Cross-Level Requirements Tracing Based on Large Language Models

Chuyan Ge[a], TianTian Wang[a], XiaoTian Yang[a], Christoph Treude[b]

[a] *Faculty of Computing, Harbin Institute of Technology, China*
[b] *School of Computing and Information Systems, Singapore Management University, Singapore*

Abstract

Cross-level requirements traceability, linking **high-level requirements (HLRs)** and **low-level requirements (LLRs)**, is essential for maintaining relationships and consistency in software development. However, the manual creation of requirements links necessitates a profound understanding of the project and entails a complex and laborious process. Existing machine learning and deep learning methods often fail to fully understand semantic information, leading to low accuracy and unstable performance. This paper presents the first approach for cross-level requirements tracing based on large language models (LLMs) and introduces a data augmentation strategy (such as synonym replacement, machine translation, and noise introduction) to enhance model robustness. We compare three fine-tuning strategies—LoRA, P-Tuning, and Prompt-Tuning—on different scales of LLaMA models (1.1B, 7B, and 13B). The fine-tuned LLMs exhibit superior performance across various datasets, including six single-project datasets, three cross-project datasets within the same domain, and one cross-domain dataset. Experimental results show that fine-tuned LLMs outperform traditional information retrieval, machine learning, and deep learning methods on various datasets. Furthermore, we compare the performance of GPT and DeepSeek LLMs under different prompt templates, revealing their high sensitivity to prompt design and relatively poor result stability. Our approach achieves superior performance, outperforming GPT-4o and DeepSeek-r1 by 16.27% and 16.8% in F1 score on cross-domain datasets. Compared to the baseline method that relies on prompt engineering, it achieves a maximum improvement of 13.8%.

Keywords: Requirements Tracing, Large Language Models, Fine-tuning, Data Augmentation, Software Requirements

1. INTRODUCTION

Cross-level requirements traceability, linking high-level and low-level requirements, is essential for maintaining relationships and consistency in software engineering. Standards such as IEEE Std.830 and CMMI stress the importance of requirements traceability in software development [1]. Establishing trace links between requirements at various levels is essential for supporting activities such as verification, validation, and change management, ensuring the correctness of system development.

Traditional methods of manual requirements tracing are time-consuming, resource-intensive, and error-prone due to subjective biases and semantic gaps[2]. Therefore, researchers have investigated the use of automated tracing techniques based on information retrieval (IR) [3, 4], machine learning (ML) [5, 6], and deep learning techniques (DL) [7, 8, 9]. However, these methods require extensive processes of information extraction and metrics computation. Moreover, they often fail to deeply understand and capture the semantics of requirements texts. This limitation is particularly pronounced when hierarchical relationships are ambiguously defined, resulting in diminished stability and accuracy.

Large language models (LLMs), through extensive pre-training on vast corpora, have acquired rich semantic knowledge, enabling them to capture the semantic information of requirements texts more effectively. Therefore, we believe that LLMs will bring new prospects to cross-level requirements tracing. Some current work has already explored the potential of LLMs for other requirements engineering tasks. Anamaria et al. [10] investigate the capabilities of GPT-3.5 and GPT-4 to assess requirements coverage, employing various prompting strategies. However, their approach only examines the impact of prompt engineering on LLMs and does not delve deeply into techniques such as fine-tuning strategies.

In addition to the aforementioned issues, methods cited in previous research often require invoking a series of algorithms for pre-processing of requirements texts. This involves tasks such as tokenization, stemming, and stop-word removal, adding complexity to the pre-processing stage. Furthermore, the selection of different pre-processing techniques and IR technologies can significantly impact the results. In Section 4.4, we discuss the influence of different IR technologies on requirements tracing results. Moreover, machine learning or deep learning methods also involve extracting features from requirements texts. For instance, in the work of Mills et al. [5], 14 IR-based features, 112 query quality metrics, and 5 artifact features are utilized. The extraction process of these feature metrics in practical applications consumes significant time and effort. Subsequently, feature selection techniques are employed to refine the extracted features, making the overall process quite intricate.

During our preliminary research, we observe that most existing studies focus on establishing traceability links between requirements and code [11, 12], test cases [13, 14], and various components. However, comparatively less attention has been

paid to the cross-level requirements traceability aspect.

To address the aforementioned challenges, this paper introduces a novel cross-level requirements tracing method based on LLMs. We simplify the data pre-processing steps. Unlike previous work that typically involves various similarity measurements on a series of requirements texts, the requirements text data is tokenized for further analysis, with excessively long texts being summarized when necessary. To enhance the model's robustness, a series of data augmentations are performed on the entire dataset. A binary classification approach is employed to implement the proposed method, with LLaMA fine-tuned on pre-processed datasets. A series of intra-project, cross-project within the same domain, and cross-domain dataset experiments is conducted. Our empirical results demonstrate that, compared to traditional methods, the LLM-based approach exhibits superior performance in requirements tracing tasks. This research makes the following contributions:

- This research pioneers the application of LLMs to cross-level requirements traceability, introducing a novel approach that expands the boundaries of this field.
- The cross-level requirements traceability datasets are organized across three domains. The organized datasets include six single-project datasets, three cross-project datasets within the same domain, and one cross-domain dataset. Data augmentation techniques, including random synonym replacement, machine translation, and noise injection, are applied to expand the cross-domain dataset, resulting in over 2,400 requirement texts, enhancing data diversity and generalization.
- A systematic analysis is conducted on three fine-tuning strategies—LoRA, P-Tuning, and Prompt-Tuning—across three different scales of LLaMA models (1.1B, 7B, and 13B), providing insights into their performance and applicability.
- Seven information retrieval (IR)-based requirements traceability methods, five machine learning models, and three deep learning models are systematically reproduced. The proposed approach outperforms traditional IR and machine learning methods, improving F1 scores by 56.07% and 26.37%, respectively.
- The performance of GPT, DeepSeek and Claude LLMs are evaluated under different prompt templates, revealing high sensitivity to prompt design and poor stability. The proposed approach surpasses GPT-4o and DeepSeek-r1 by 16.27% and 16.8% in F1 score on cross-domain datasets, and exceeds the Claude LLM-based prompt engineering baselines from Rodríguez et al. [15] by up to 13.8%.

2. RELATED WORKS

This section summarizes the related work in the field of demand tracing in three areas: information retrieval-based methods, rule-based methods, and learning-based methods.

2.1. Information retrieval based methods

IR based methods, such as the Vector Space Model (VSM) and Latent Semantic Indexing (LSI)[16] are widely applied in cross-level requirement traceability research due to the characteristics of requirement texts. These methods typically assess traceability relations between two texts by calculating the similarity of their content. The underlying assumption is that the higher the similarity between two texts, indicating more shared or similar vocabulary, the greater the likelihood of a traceability relation between them.

Common information retrieval similarity metrics include VSM, Latent Semantic Analysis (LSA), Latent Dirichlet Allocation (LDA), Jensen-Shannon (JS) divergence, Okapi BM25, Language Model with Dirichlet smoothing (LM-Dirichlet), and Jelinek Mercer smoothing (LM-JM), among others. VSM, known for its intuitive and straightforward approach, has been widely adopted since its introduction and is often used as a baseline in studies[17, 18]. Hayes et al.[19] augmented traditional VSM using Standard Rocchio feedback technique for requirement tracing. By optimizing queries with user feedback on retrieval results, they demonstrated significant improvements in accuracy and recall in linking high-level to low-level requirements, thus successfully enhancing VSM's performance.

In real-world applications, discrepancies in the terminologies used to describe the same concept across different components or by different individuals can lead to term mismatch issues and instances of polysemy. These challenges pose difficulties for traditional information retrieval methods. To address these issues, researchers alleviateed term mismatch problems among cross-level requirements artifacts by constructing lexical databases and annotating the relationships between terms[20, 21]. However, the lack of readily available lexical databases in most projects and the substantial overhead of manual annotation hinder the application of these methods.

2.2. Rules-based methods

In the realm of requirement traceability research, alongside methods handling similarity-based methods, some researchers also explore rule-based methodologies from a logical standpoint for requirement tracing. These methodologies focus on analyzing the logical relationships between requirements, such as preconditions, constraints, and triggering conditions, to infer connections between them. Guo et al.[22] introduce DoCIT for communication and control systems in transportation, using rules to extract "action units" from software artifacts, a knowledge base for linking similar concepts, and heuristics to create trace links, outperforming traditional VSM. However, rule-based methods may struggle with natural language texts from open-source systems due to inconsistent formatting. Khlif et al.[23] proposed a traceability method for software development, involving natural language pre-processing to organize texts into XML, mapping UML diagram elements via traceability rules, and linking requirement descriptions to code, ensuring consistency across development stages.

These reseraches illustrates the flexibility and adaptability of rule-based methods when combined with appropriate pre-

processing steps, allowing them to handle a variety of text formats and contribute effectively to software development processes. However, rule-based methods typically demand precise formatting for input artifacts, a requirement that natural language texts, particularly those from open-source systems characterized by inconsistent formatting and irregular writing styles, often do not satisfy.

2.3. Learning-based methods

With the rise of artificial intelligence technologies such as machine learning and deep learning, researchers have explored the practical effectiveness of learning-based methods in the field of requirement traceability. They use requirement texts and existing or annotated requirement traceability relations as training sets to train machine learning or deep learning models to identify more requirement traceability relations. These methods typically start by extracting query quality(QQ) features from components. Subsequently, they use various IR-based methods to calculate the similarity between components, generating rankings. Some researchers treat this as a binary classification problem, categorizing the relationship between a pair of software artifacts as positive (indicating a traceability relation exists) or negative (indicating no traceability relation exists). Bayesian algorithms, Random Forest, and support vector machines are commonly used classifiers by researchers[24, 5, 25].

Mills et al.[5] proposed an active learning approach that utilized 131 features including 14 IR-based features, 112 query quality metrics, and 5 artifact features to establish traceability links. They implemented a random forest classifier, which outperformed standard methods, reducing the training data required for effective traceability link recovery.

Additionally, some researchers utilize deep learning based methods to train text vectors that better represent the semantic information of requirements[26]. These text vectors are then used for information retrieval to more accurately identify traceability links. In response to the traceability of cross-level requirements, Tian et al.[1] performed second-phase pre-training on BERT using a project document corpus to facilitate project-related knowledge transfer, resulting in superior text embeddings. In addition to traditional features, the authors designed 11 heuristic features based on the requirement metadata in the open-source system. Experimental results demonstrate the effectiveness of their proposed method, with the highest achieved F1 score reaching 80.5%. However, their work still faces certain challenges. For instance, information such as assignee and creator details may not always be fully transparent or accessible, potentially impacting the accuracy and completeness of the analysis. Furthermore, their experimental results exhibit significant variations across different projects, with some projects achieving as low as 53.6% F1 score.

The dataset's significance is crucial for high-performance machine learning or deep learning models. Feature extraction is a key step, as different features can impact the model's performance in traceability link recovery. Therefore, selecting appropriate feature extraction methods that capture the correlation and semantic information among software artifacts is vital. Additionally, learning-based methods require numerous features to assess traceability, adding to the workload of requirement traceability.

In contrast, LLMs offer a promising alternative by enabling deeper semantic analysis with minimal data processing. Recent work has explored this advantage in requirements engineering. Anamaria et al.[10] also applied LLM to requirements engineering, They investigate the capabilities of GPT-3.5 and GPT-4 in assessing requirements coverage, employing various prompting strategies. The results reveal that different prompt templates significantly impact the models' performance on downstream tasks. GPT-3.5, using a zero-shot prompting strategy, accurately identifies complete coverage in four out of five datasets, highlighting the substantial potential of LLMs in requirements engineering. However, this work does not explore fine-tuning techniques, which could further enhance the models' effectiveness in this domain.

3. OVERVIEW OF APPROACH

An overview of the approach is depicted in Figure 1, illustrating the process of data collection, text augmentation, and fine-tuning strategies. Data is first collected from three domains— aerospace, computer, and medical— resulting in the construction of six single-project datasets, three cross-project datasets within the same domain, and one cross-domain dataset. To address the issue of information loss due to the input length limitations of LLMs, text augmentation techniques are applied to longer requirement texts. Several data augmentation techniques are investigated, including random synonym replacement, machine translation, and noise introduction (such as random word order shuffling and word deletion). These techniques enrich the datasets and enhance model robustness. Comparative experiments are conducted using three fine-tuning strategies—LoRA, P-tuning, and Prompt-Tuning—across different scales of LLaMA models (1.1B, 7B, and 13B). The effectiveness of each fine-tuning approach in improving requirement tracing performance is assessed.

3.1. Data Pre-processing

Integrating domain knowledge into LLMs effectively necessitates the creation of trustworthy and reusable domain-specific datasets. As depicted in Figure 2, considering the various challenges of training LLMs, we initially condense lengthy texts through summarization techniques to retain their key information. Subsequently, we apply a series of text data augmentation techniques to enhance dataset diversity and improve model robustness. Finally, we concatenate high-level and low-level requirement texts and transform them into vector form using a tokenizer before inputting them into the LLM. In the following sections, each step is elaborated upon in detail.

3.1.1. Dataset collection

The datasets been selected are from aerospace, computer, and medical domains. **CM1** and **Modis**[1] are open-source re-

[1] http://promise.site.uottawa.ca/SERepository/datasets/

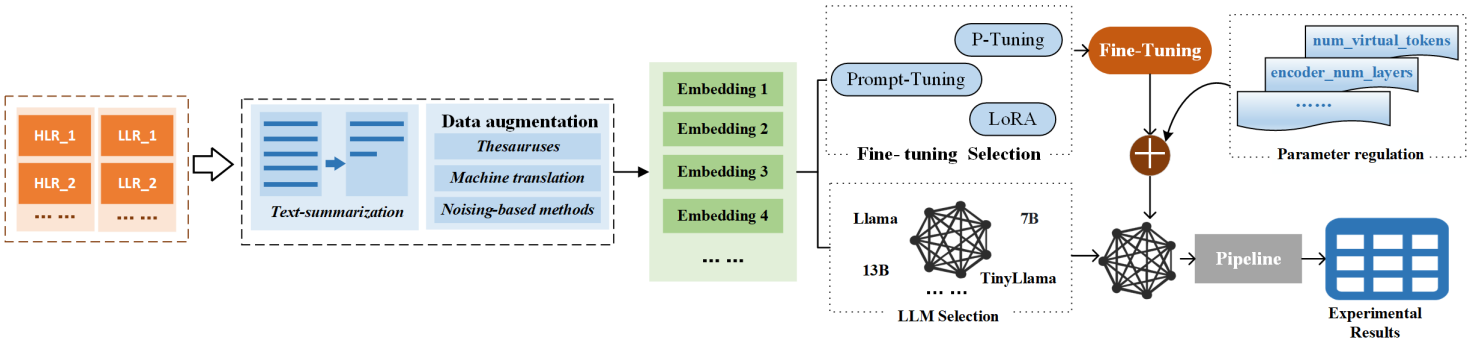


Figure 1: Overview of Approach

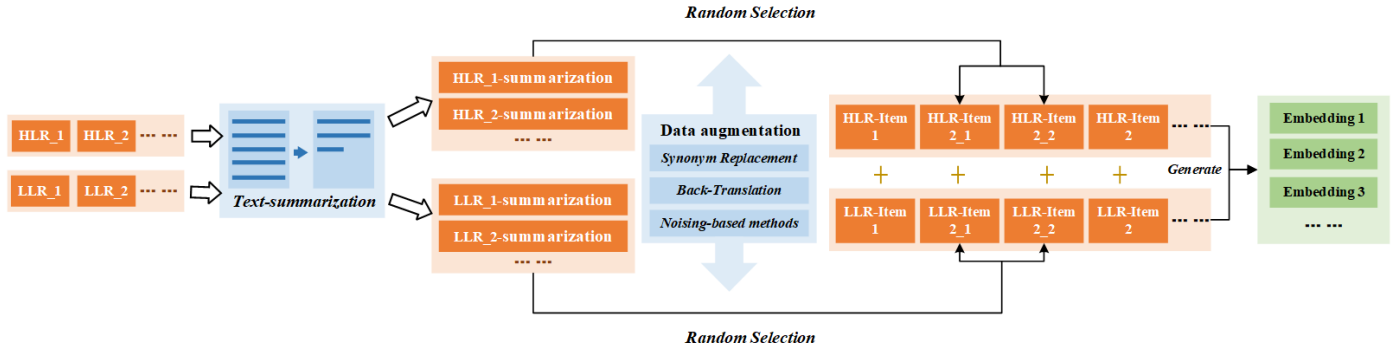


Figure 2: Data Preprocessing Workflow

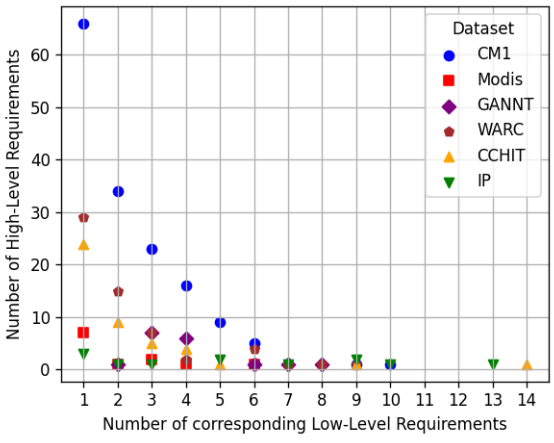


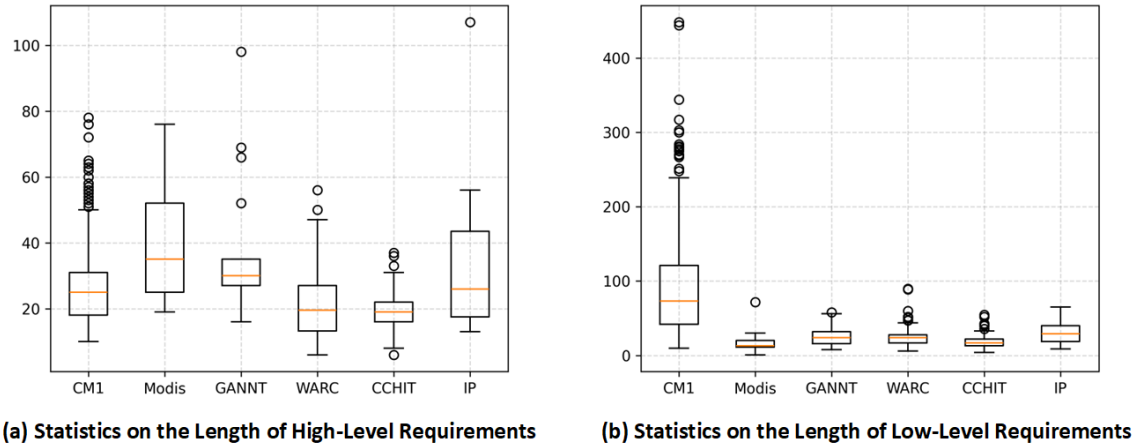
Figure 3: Distribution of high-level requirements based on the number of corresponding LLRs for each dataset.

quirement datasets provided by NASA in the aerospace domain. The **GANNT** dataset[2] describes the requirements of the GanttProject open-source project management system, while the **WARC** dataset[2] includes functional and non-functional system requirements (HLR) from the medical domain, along with their traces to the software requirements specification (LLR). The **InfusionPump(IP)** dataset details the software system of a medical infusion pump, and the **CCHIT** dataset[2] focuses on healthcare information technology standards (HLR) developed by the Certification Commission for Healthcare Information Technology (CCHIT), tracing requirements from the WorldVistaA system (LLR).

We analyze the distribution of HLR and their corresponding LLR across multiple datasets (see Figure 3). Each dataset uses different colors and marker styles. The chart shows the distribution of HLR and their corresponding LLR for each dataset. In the CM1 dataset, out of 64 HLR, only one has a corresponding LLR, with most HLR linked to more than 6 LLR. The distributions in the Modis and GANNT datasets are relatively uniform, with most GANNT HLR linking to 3 to 8 LLR, and most Modis HLR linking to 1 to 4 LLR. The CCHIT dataset contains the HLR with the most corresponding LLR, reaching 14. In the IP dataset, this number reaches up to 13.

Additionally, we analyze the length of the requirements in

[2]http://sarec.nd.edu/coest/datasets/



(a) Statistics on the Length of High-Level Requirements **(b) Statistics on the Length of Low-Level Requirements**

Figure 4: Token Statistics for requirements for each dataset.

different datasets and count the number of tokens after processing them with the tokenizer. Since our method converts both words and symbols into individual tokens, the actual statistics represent the number of words and symbols in the demand statements. The high-level and low-level requirements are respectively recorded in Figure 4 . Notably, there are many lengthy entries in multiple categories. The lengths of high-level requirements across various projects are relatively balanced, with none exceeding 100. However, in the low-level requirements, there are numerous lengthy entries, especially in the ”CM1” category. Specifically, the word count range in the ”CM1” category is quite broad, with the minimum being 11 and the maximum reaching 448, the highest among all categories. Additionally, there are several entries with word counts exceeding 100, such as 448, 444, 303, and 280. This indicates that this category contains a large number of lengthy demand texts, possibly involving more detailed and complex requirements.

When generating candidate links, we first take the high-level and low-level requirements that already have traceability links in the open-source dataset and pair them as positive samples, assigning them a label of 1. Then, we randomly match high-level requirement entries with low-level requirement entries to generate negative samples. In our dataset, the number of non-traceable pairs is significantly larger than that of traceable ones. To mitigate data imbalance, we randomly remove a portion of the non-traceable samples. During this process, we ensure that every high-level and low-level requirement from the original dataset is retained in the final dataset, regardless of whether it has a corresponding traceability link. Additionally, to maintain a balanced dataset, we adjust the final positive-to-negative sample ratio to approximately 4:6.This choice is informed by prior work in traceability link recovery tasks. For example, Zhang et al. [27] employed a similar data distribution where the proportion of positive samples is around 38%, which is close to our setting. Their empirical results demonstrated that such a ratio maintains sufficient negative diversity while alleviating the impact of class imbalance on model training.

To facilitate cross-domain analysis, we further merge all single-project datasets into a single cross-domain dataset, enabling a comprehensive evaluation of data augmentation techniques across different domains. When constructing cross-project datasets, we randomly generate links between requirement entries from multiple projects as negative samples. Based on this strategy, we construct 10 datasets, including 6 single-project datasets, 3 cross-project datasets within the same domain, and 1 cross-domain dataset. Specifically, CM1 and Modis represent the aviation domain, GANNT and WARC belong to the computer science domain, and CCHIT and IP correspond to the medical domain.

3.1.2. Text-summarization methods

Although the LLaMA model we employ can process relatively long text inputs, excessively long sequences significantly increase training time and computational complexity. Additionally, LLMs tend to focus more on the initial portion of an input while potentially overlooking later content, making it challenging to capture key information in lengthy texts. However, in our dataset, the length of requirement texts varies significantly, with some entries being particularly long (see Section 3.1.1). Overly long texts can further disperse the model's attention, making it difficult to extract essential information. Our analysis shows that low-level requirements tend to be lengthier, and directly segmenting these texts may lead to information loss, ultimately reducing the accuracy of requirements traceability.

To solve this problem, we apply two text summarization techniques for analysis. First, we use *TextRank* [28], and then we apply a LLM for text summarization. Inspired by the *PageRank* algorithm, *TextRank* applies a graph-based ranking approach for extractive summarization. It constructs an undirected weighted graph where sentences act as nodes, and the similarity between sentences determines the edge weights. This method enables the selection of key sentences that best represent the original text, improving the efficiency of requirement analysis.

As depicted in Figure 5, the requirement text is first split into sentences. If the number of resulting sentences is less than the specified top n, all sentences are returned as the summary. Then, a TF-IDF matrix is generated for the sentences, and the cosine similarity between every pair of sentences is calculated

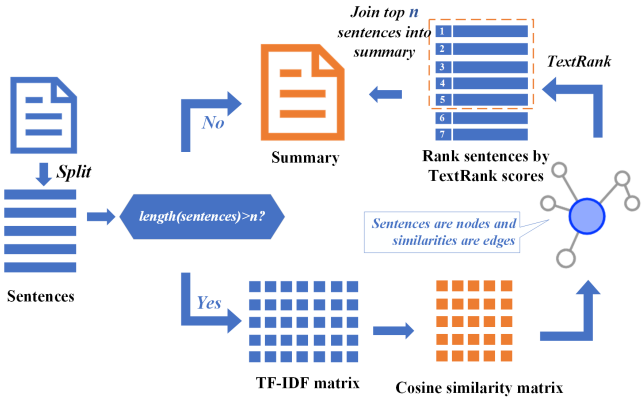


Figure 5: Text Summarization Algorithm

to obtain a similarity matrix. Next, a graph is constructed, where sentences are represented as nodes, and edges reflect the cosine similarities between the sentences. In this graph, the *TextRank* algorithm is applied to compute scores for each sentence. The sentences are then sorted in descending order based on their *TextRank* scores, and the top n sentences with the highest *TextRank* scores are selected to form the summary. Finally, the selected sentences are concatenated to generate the summary, which is returned as the output.

We also apply a LLM-based methord for text summarization. Specifically, we use the BART (Bidirectional and Auto-Regressive Transformer) model[29], which is pre-trained on large-scale text corpora and fine-tuned on the CNN/Daily Mail dataset, optimized for text summarization tasks. The BART model follows an encoder-decoder architecture, where the encoder processes the input text and captures its semantic representations, while the decoder generates a concise and informative summary. To apply the model efficiently, Hugging Face's pipeline function is used with specific settings. The max_length is set to 130 to ensure the summary does not exceed 130 tokens. The min_length is set to 30 to prevent overly short outputs. Additionally, do_sample is disabled to produce more deterministic and stable summaries. During execution, the input text is fed into the BART model, where the encoder extracts the semantic information, and the decoder generates a summary that balances conciseness and informativeness. In case of an error during summarization, the function returns the original text to ensure robustness.

Figure 6 illustrates an example of text summarization, showing the original requirement text, the summary with TextRank applied, and the text segmented directly. It is evident that TextRank effectively preserves important details from the original text by extracting key content and maintaining logical relationships. In contrast, direct segmentation can lead to information loss. For instance, during segmentation, the sentence "The control panel also enables the clinician to request a bolus and enter the bolus duration" is truncated, resulting in the loss of complete details about bolus requests and duration entry. This information loss affects the coherence of the text and the comprehensive understanding of the control panel's functions. In comparison, the summarized version retains key information while enhancing conciseness: "The control panel also enables the clinician to request a bolus and enter the bolus duration. It allows an authorized clinician to start and stop infusion, display the prescription, and use a scanner to read the drug container for confirmation. A clinician uses this panel to enter configurations and view alarms and warnings." Since high-level requirements are generally concise, we retain their original form to prevent any potential loss of critical information.

3.1.3. Data augmentation

During the requirements traceability process, data volumes are usually limited. We implement various data augmentation algorithms to address the potential robustness issues of LLMs in low-sample situations. These techniques help expand and diversify the training dataset, ultimately enhancing the model's resilience. Data augmentation encompasses a spectrum of methods, from simple rule-based techniques to advanced learnable generation-based methods, all aimed at ensuring the integrity and validity of the augmented data [30].

3.1.3.1. Synonym Replacement. Several methods aim to refresh the expression of the original text by substituting words with their synonyms and hypernyms, thereby finding a new mode of expression while striving to keep the semantics of the original text as intact as possible. A popular method for text augmentation, known as EDA (Easy Data Augmentation Techniques) proposed by Wei et al. in 2019[31], also employs *WordNet* to substitute words in a sentence with their synonyms by randomly selecting n words that are not considered stop words. We utilize the nltk library, which provides a vast lexical resource and synonym relationships.

As shown in Algorithm 1, systematically enhances sentences by carefully replacing words with their synonyms, drawn from the extensive database. Subsequently, we apply a series of text augmentation techniques to enhance dataset diversity and improve model Subsequently, we apply a series of text augmentation techniques to enhance dataset diversity and improve model By integrating the principle of synonym and hypernym replacement, Algorithm 1 not only enriches the text's expression but also maintains the original semantics as closely as possible, offering a refined method for text augmentation.

3.1.3.2. Machine translation. Beyond merely altering individual words, we incorporate the method of back-translation to induce changes in sentence structures. This method leverages grammatical differences among languages to transform passive to active voice or convert simple sentences into complex ones.

With the rise of machine translation models and accessible online APIs, services like Google and DeepL are frequently used for this purpose. In this work, we select Pygtrans Translate[3], a Python-based library leveraging the robust Google Translate API. This library offers simplicity, supports translations across multiple languages, and boasts high customizability to

[3]https://pygtrans.readthedocs.io/zh-cn/latest/pygtrans.html

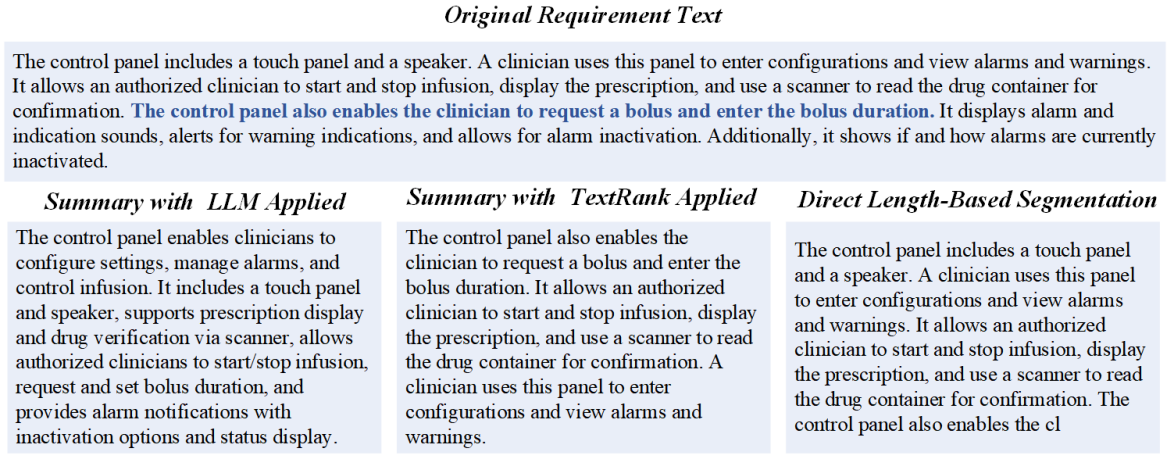


Figure 6: An Example of Text Summarization Algorithm

Algorithm 1 Synonym Replacement Algorithm

Input: sentence ▷ Input sentence
Output: augmented_sentence ▷ Sentence with synonyms replaced

```
words ← split(sentence)
n ← max(1, ⌊count_words(sentence)/5⌋)
num_replaced ← 0
for word in random_words do
    synonyms ← get_synonyms(word)
    if synonyms is not empty then
        synonym ← random_choice(synonyms)
        words ← replace(words, word, synonym)
        num_replaced ← num_replaced + 1
        if num_replaced ≥ n then
            break
        end if
    end if
end for
augmented_sentence ← join(words)
return augmented_sentence
function GET_SYNONYMS(word)
    return [lemma.name() for syn in wordnet.synsets(word) for lemma in syn.lemmas()]
end function
function COUNT_WORDS(sentence)
    return len(split(sentence))
end function
```

Figure 7: An Example of Data augmentation by Machine translation

suit diverse project needs. With Pygtrans Translate integrated, we develop a back-translation function. This function translates the original text into a target language (e.g., French or Chinese) and then back into the original language (e.g., English) using the *translate*() method. Figure 7 shows an example of the transformation of requirement. Extensive testing is conducted to ensure the accuracy and effectiveness of the back-translation process, validating that core meanings are preserved while introducing structural variations.

3.1.3.3. Noising-based methods. The semantics of natural language are sensitive to the order of text, yet slight changes in order remain readable for humans. Therefore, random swapping of words or even sentences within a reasonable range can be used as a data augmentation method. More specifically, this involves randomly deleting words in a sentence or deleting sentences in a document. As for word-level deletion, Wei et al. [31] randomly remove each word in the sentence with probability p.

In our study, we implement the random noise introduction method detailed in [30], which involves randomly swapping the positions of two words and randomly deleting words as outlined in the cited paper. This approach introduces mild noise that minimally impacts the semantics, allowing for a subtle deviation from the original data. Both discrete and continuous noise types are utilized to maintain the method's effectiveness. The primary goal of these techniques is to enhance the model's robustness.

Random Word Swap (RWS) algorithm, as shown in Algorithm 2, accepts a sentence and an integer n as input, representing the number of word swaps to perform. The execution steps of the algorithm are as follows: Firstly, the input sentence is split into a list of words. Then, for the specified number n, two different words are randomly selected and their positions are swapped. Finally, the swapped word list is recombined into a new sentence, and the augmented sentence after n word swaps is returned.

The Random Deletion (RD) algorithm, as illustrated in

Algorithm 2 Random Word Swap (RWS)

Input: sentence (a string); n (an integer specifying number of swaps);
Output: new_sentence (a string with n words swapped);

```
words ← SPLIT(sentence);
new_words ← COPY(words);
for i from 1 to n do
    random_idx_1 ← RANDOM(0, LENGTH(new_words) - 1);
    random_idx_2 ← random_idx_1;
    counter ← 0;
    while (random_idx_2 = random_idx_1) and (counter ≤ 3) do
        random_idx_2 ← RANDOM(0, LENGTH(new_words) - 1);
        counter ← counter + 1;
    end while

    if counter ≤ 3 then
        SWAP(new_words[random_idx_1], new_words[random_idx_2]);
    end if

end for
new_sentence ← JOIN(new_words);
return new_sentence;
```

Algorithm 3, receives a sentence and a floating-point number p as input, indicating the probability of word deletion. The execution steps of the algorithm are outlined as follows: Initially, the input sentence is split into a list of words. If the sentence contains only one word, the original sentence is returned directly as deletion cannot be performed. Then, for each word in the sentence, a random decision is made based on the given probability p to determine whether the word is retained. If no words are retained, a random word from the original sentence is selected and returned as the augmented sentence. Lastly, the retained words are concatenated into a new sentence, which is returned. Through such algorithmic design, we can achieve random word swapping and deletion operations on sentences in natural language processing tasks, thereby enhancing the diversity and robustness of the dataset.

Algorithm 3 Random Deletion (RD)

Input: sentence (a string); p (a float representing deletion probability);
Output: new_sentence (a string after random deletions);

```
words ← SPLIT(sentence into words);
if LENGTH(words) = 1 then
    return sentence;
end if
new_words ← EMPTY LIST;
for each word in words do
    r ← RANDOM NUMBER between 0 and 1;
    if r > p then
        APPEND word to new_words;
    end if
end for
if LENGTH(new_words) = 0 then
    rand_int ← RANDOM INTEGER between 0 and (LENGTH(words) -
1);
    return words[rand_int];
end if
new_sentence ← JOIN(new_words into a string);
return new_sentence;
```

In order to enhance the robustness of the model while ensuring the quality of the dataset, we employ the following augmentation techniques on all dataset samples: Firstly, utilizing paraphrasing-based methods, we randomly select 10% of the requirement texts, enabling us to introduce variations in sentence structures and expressions. This technique promotes linguistic diversity and aids in mitigating overfitting. Secondly, employing machine translation on another 10% of the requirement texts serves to simulate linguistic diversity and broaden the vocabulary, thus further enriching the dataset. Subsequently, we implement random word swaps on 10% of the requirement texts, facilitating the introduction of lexical diversity and enhancing the model's tolerance to minor variations in wording. Finally, random deletion is applied to 10% of the requirement texts, simulating noise and enhancing the model's adaptability in handling missing information. The resulting newly generated samples from these augmentation methods are seamlessly integrated into the dataset, ensuring a comprehensive training corpus that encompasses diverse variations. This approach not only maintains the integrity of the dataset but also contributes to fostering a more robust model that is better equipped to generalize and perform effectively across various scenarios.

To ensure the semantic integrity and linguistic quality of the augmented data, we conduct a manual validation process involving three key scoring dimensions: semantic consistency, grammatical fluency, and relevance to the original requirement.

Each augmented requirement x' is compared with its original counterpart x, and evaluated using the following scoring criteria:

- **Semantic Consistency**:

$$S_{\text{sem}}(x, x') = \begin{cases} 2, & \text{Nearly identical meaning.} \\ 1, & \text{Partial meaning reflection.} \\ 0, & \text{Significant divergence.} \end{cases} \tag{1}$$

- **Grammatical Fluency**:

$$S_{\text{gram}}(x') = \begin{cases} 2, & \text{Grammatically correct.} \\ 1, & \text{Minor issues.} \\ 0, & \text{Ungrammatical or unnatural.} \end{cases} \tag{2}$$

- **Relevance to Original**:

$$S_{\text{rel}}(x, x') = \begin{cases} 2, & \text{Highly relevant.} \\ 1, & \text{Loosely related.} \\ 0, & \text{Irrelevant.} \end{cases} \tag{3}$$

The total score for an augmented sample is calculated as:

$$S_{\text{total}} = S_{\text{sem}} + S_{\text{gram}} + S_{\text{rel}} \tag{4}$$

An augmented sample is accepted for downstream tasks if it scores S_{total} at least 5 points out of 6. Otherwise, it is flagged for further review or discarded.

3.2. Large Language Model and Fine-tuning

3.2.1. Large Language Model-LLaMA

LLMs have significantly advanced natural language processing (NLP) by enabling efficient adaptation to a wide range of tasks with minimal human intervention. Among them, LLaMA, developed by Meta, stands out as a powerful decoder-only Transformer model that incorporates several architectural improvements for augmented efficiency and performance. These include replacing LayerNorm with RMSNorm, adopting Group

Project	Domain	No. High-requirement		No. Low-requirement		No. positive samples in the augmented dataset	No. negative samples in the augmented dataset	No. samples in the augmented dataset
		Original Datasets	Augmented Datasets	Original Datasets	Augmented Datasets			
CM1	Aerospace	235	545	220	461	515	668	1183
Modis	Aerospace	19	36	49	38	34	39	73
GANNT	Project management	17	86	69	136	87	190	277
WARC	Web Tools	64	148	89	176	190	168	358
CCHIT	Healthcare	45	131	91	181	141	211	352
IP	Healthcare	12	66	55	117	90	139	229
Total	/	**392**	**1012**	**573**	**1109**	**1057**	**1415**	**2472**

Table 1: Summary of Projects and Requirements

Query Attention (GQA)—a generalization of Multi-Query Attention (MQA)—and introducing Rotary Positional Embedding (RoPE) in place of traditional positional encoding. The 7B-parameter LLaMA model demonstrates strong reasoning capabilities, achieving up to 97.7% accuracy in mathematical tasks [32].

In our study, we leverage LLaMA for binary classification in cross-level requirements traceability, aiming to identify semantically or functionally related elements between high-level stakeholder requirements and low-level system requirements. This task demands the ability to model long-range dependencies and handle varied linguistic expressions across different abstraction layers. LLaMA's reasoning strength and efficient architectural design (notably GQA and RoPE) make it well-suited for aligning scattered yet related information in lengthy and heterogeneous requirements documents, thereby enabling accurate and scalable automated traceability analysis.

3.2.2. *Fine-tuning*

To further optimize model performance and efficiency, we integrate widely adopted fine-tuning techniques, including LoRA [33], Prompt-Tuning [34], and P-Tuning [35]. Figure 8 illustrates the logic behind these fine-tuning methods and their role in improving task-specific adaptation.

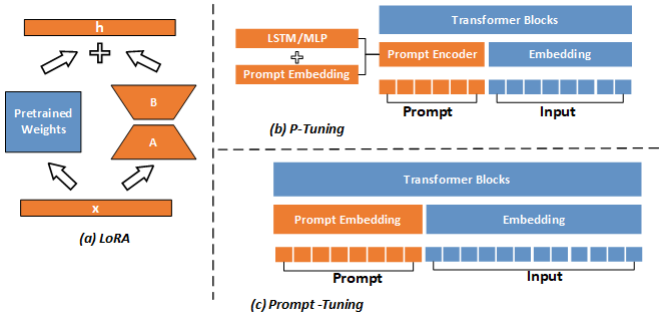


Figure 8: Fine-Tuning Techniques

3.2.2.1. *LoRA.*

3.2.2.2. *P-Tuning v2.*

P-Tuning v2 operates by adding prompts of length p tokens to all layers of the language model in the form of sequences of continuous prefix tokens. These continuous embeddings are represented by the vector $H_m = [h_0, \ldots, h_p]$, where each $h_i \in \mathbb{R}^d$ corresponds to the dimensionality of token embeddings for each layer m in the Transformer-based model. These embeddings are prefixed to the embeddings of each Transformer layer $E_m = [E_{\text{CLS}}, E_1, \ldots, E_n]$, where n is the maximum sequence length. Thus, each layer of the LLM can attend to the prompts independently.

In the fine-tuning process, we prepend 128 trainable virtual tokens to the input sequence, which are mapped into task-specific representations using a 16-layer MLP encoder. The encoder has a hidden size of 128 and employs MLP as the reparameterization method, with a dropout rate of 0.3 to enhance generalization. During training, these virtual tokens undergo nonlinear transformations via the MLP encoder and are concatenated with the original input text embeddings to form the final input to the Transformer model. The pre-trained model then performs forward propagation for sequence classification. Throughout this process, only the virtual tokens and their MLP encoder parameters are optimized, while the pre-trained model's original weights remain unchanged.

3.2.2.3. *Prompt Tuning.*

Prompt Tuning enhances pre-trained LLMs' performance by adding learnable prompts to the input. This method reformulates tasks into a "fill-in-the-blank" format, where the model predicts a masked token in a sentence. For a text sequence $X = \{x_1, x_2, \ldots, x_n\}$ and a label Y, Prompt Tuning introduces prompt embeddings $P = \{p_1, \ldots, p_{n_p}\}$, where each $p_i \in \mathbb{R}^D$ is a vector with embedding dimension D.

The process involves embedding X into the model's token space, inserting P into the sequence, and encoding it using the pre-trained encoder Φ. The encoded representation is denoted as Z:

$$Z = \Phi(P, X) \tag{5}$$

Model predictions are made with a trainable classification head h_θ, with output $h_\theta(Z)$ representing the probability distribution over possible labels. The loss function for training is:

$$L_{\text{pred}} = \text{cross\_entropy}(h_\theta(Z), Y) \tag{6}$$

The encoder parameters remain frozen during training, optimizing only the prompts and classification head. This approach allows the prompts to better capture task-specific information from X, improving model performance on downstream tasks.

During the fine-tuning process, Prompt Tuning is also applied to a sequence classification task, where the core idea is to introduce trainable virtual tokens into the input sequence to adapt the pre-trained model to the specific task. Specifically, 12 virtual tokens are added to the front of the input sequence

and initialized through a task-specific text prompt. we present a detailed scenario description:

Consider the following scenario where "high requirement text" and "low requirement text" represent high-level and low-level requirements of the software system, respectively. Your task is to ascertain the traceability between these two requirements.

Compared to P-Tuning, Prompt Tuning differs in that it only uses trainable virtual tokens for task adaptation, without introducing additional complex structures. In contrast, P-Tuning v2 uses a more complex parameterization method to perform nonlinear transformations of the virtual tokens. Moreover, P-Tuning allows trainable prompts to be inserted at different positions in the input sequence, whereas Prompt Tuning only inserts the virtual tokens at the beginning of the sequence.

While all three strategies—LoRA, P-Tuning, and Prompt-Tuning—enable pre-trained models to adapt to new tasks while retaining existing knowledge, they employ different methods. LoRA Tuning adjusts the model's internal parameters, P-Tuning integrates task-specific prompts, and Prompt-Tuning uses a trainable prompt vector. In this study, we evaluate the effectiveness of these techniques on our LLM to identify the most suitable approach for requirement traceability tasks, focusing on achieving a balance between performance, resource utilization, and the ability to generalize across different datasets.

4. EXPERIMENTAL EVALUATION

This section introduces the experimental setup and then presents the research questions and evaluation metrics. The experimental results are illustrated. For the reproducibility of the evaluation, all models and datasets are available at https://github.com /Gechuyan/Cross-Level-Requirements-Tracing-Based-on-Large-Language-Model .

4.1. Evaluation criteria and baseline method selection

This paper adopts commonly used evaluation indicators in the field of demand tracking, namely Accuracy, Recall, Precision, and F1 value. The recall rate evaluated the comprehensiveness of the automatic algorithm's recognition of trace links, that is, the percentage of the total number of correct trace links identified. The F-measure is a harmonic average of the recall rate and accuracy. These metrics can be computed by Equations 7-10.

$$\text{Accuracy} = \frac{TP + TN}{TP + FP + TN + FN} \tag{7}$$

$$\text{Precision} = \frac{TP}{TP + FP} \tag{8}$$

$$\text{Recall} = \frac{TP}{TP + FN} \tag{9}$$

$$F - \text{measure} = 2 * \frac{\text{Precision} * \text{Recall}}{\text{Precision} + \text{Recall}} \tag{10}$$

where true positive (TP) indicates the number of positive samples correctly classified as positive, true negative (TN) indicates the number of negative samples correctly classified as negative, false-positive (FP) indicates the number of negative samples incorrectly classified as positive, and false negative (FN) indicates the number of positive samples incorrectly classified as negative.

4.2. Research Questions

To evaluate the effectiveness of our proposed method, we investigate the following research questions (RQs):

RQ1 How do different-sized LLaMA models perform under various fine-tuning techniques?

RQ2 How does our approach compare with various baselines and existing state-of-the-art methods?

- RQ2.1 How does our method compare with traditional methods, including IR-based, ML-based, and DL-based models?
- RQ2.2 How do DeepSeek and GPT perform under different prompt templates?
- RQ2.3 How does the proposed approach compare with state-of-the-art methods, relying on prompt engineering?

RQ3 Is our data augmentation approach effective?

RQ4 How effective is the summarization technique in improving model performance?

RQ1 and RQ2 focus on assessing the overall performance of our approach and comparing it against baseline methods. RQ3 and RQ4 are designed as ablation studies to analyze the impact of specific components and data processing techniques.

4.3. Experimental Setup

The overall experimental procedure is illustrated in Figure 9. To systematically evaluate our proposed method, we designed experiments corresponding to each research question (RQ), ensuring a rigorous and comprehensive analysis.

In RQ1, to investigate the impact of different fine-tuning techniques on LLaMA models of various sizes, we select three model versions: 1.1B, 7B, and 13B. Each model is fine-tuned using three distinct strategies: LoRA, Prompt-Tuning, and P-Tuning.

To prepare the training and evaluation datasets, we first randomly sample 20% of the original dataset as the test set, which is held out and remains untouched during all fine-tuning stages. The remaining 80% of the data is used for training and validation. Data augmentation techniques described in Section 3.1.3 are applied to this 80% portion, and the augmented dataset are combined with the original 80% to construct the final training pool.

From this combined dataset, four different training-validation splits—2:8, 5:5, 8:2, and 9:1—are explored to assess how the amount of training data affects fine-tuning performance. Fine-tuning experiments are conducted on a variety of scenarios, including six single-project datasets, three cross-project datasets

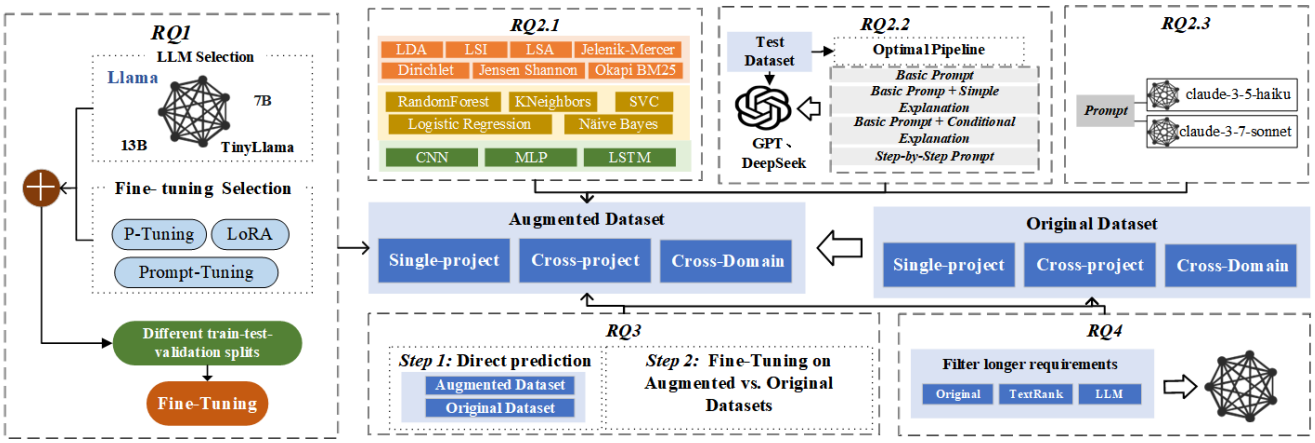


Figure 9: Overview of Experiment

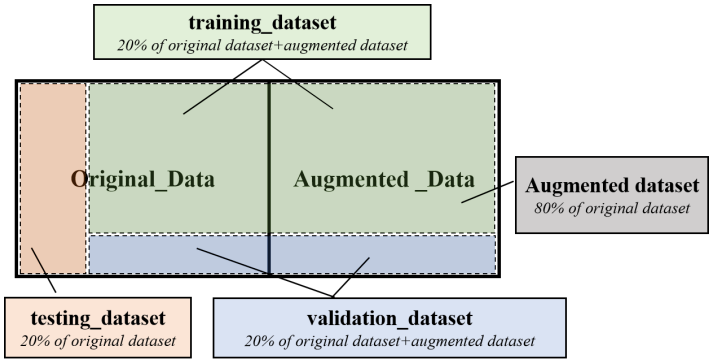


Figure 10: Composition of Datasets Used for Fine-Tuning

within the same domain, and one cross-domain dataset. To provide a clear understanding of the data processing workflow, we summarize the dataset construction process in Figure 10.

To address RQ2, which aims to evaluate the effectiveness of our proposed approach in comparison with a range of baselines and state-of-the-art methods, we design a comprehensive experimental framework composed of three stages. All experiments are conducted using the same datasets as those used in RQ1 to ensure consistency. To address RQ2.1, our investigation begins with an empirical evaluation of seven IR-based methods. In this phase, we apply seven distinct IR techniques to assess their effectiveness in traceability tasks. In the second part, we evaluate the performance of machine learning algorithms in conjunction with our approach. We compute the QQ value using the formulas provided in the online appendix, accessible at http://github.com/TLR2019/resource/tree/master. Additionally, we utilize seven IR-based calculation techniques, as outlined in the study by Du et al. [25], to extract ranking features for two key components. These extracted features serve as inputs for various machine learning models. In the third part, we assess the effectiveness of our data processing methodology when applied to traditional deep learning architectures. To this end, we conduct experiments using classic models such as CNNs, LSTMs, and MLPs, evaluating their performance within our refined data-processing framework.

In RQ2.2, our approach is compared with GPT-3.5, GPT-4o, and DeepSeek-r1, and four prompt templates—Basic Prompt, Basic Prompt + Simple Explanation, Basic Prompt + Conditional Explanation, and Step-by-Step Prompt—are applied based on the method described in [10].To ensure a fair comparison, we use the same random seed as in LLM fine-tuning and conduct experiments on the same test dataset. Since our approach leverages Prompt-Tuning, which directly fine-tunes the model on task-specific data, it eliminates the need for additional prompt engineering techniques. This allows the model to adapt to the task without manually adjusting prompt templates for different configurations.

In RQ2.3, to benchmark the proposed approach against state-of-the-art techniques, the method proposed by Rodríguez et al. [15] is adopted, which applies different prompt template strategies to identify the optimal prompt and LLM, relying on prompt engineering. This comparison aims to assess the effectiveness of the proposed method in requirement traceability tasks.

First, data augmentation techniques, including synonym replacement, machine translation, and noise-based methods (such as random deletion and random word swap), are applied to all entries in the single-project datasets. The augmented versions are saved as separate datasets. Predictions are then performed using LLaMA models of different sizes, and the results are compared on both the original and augmented datasets. This step is aimed at assessing the sensitivity of LLMs to augmented data and determining whether the augmentation methods effectively enhance dataset diversity. In the second step, LLMs are fine-tuned on all 10 datasets, both before and after augmentation. This experiment is designed to evaluate whether the data augmentation techniques improve the model's performance.

In RQ4, to verify the impact of text summarization techniques on the model's ability to identify traceability links, we design an ablation experiment. We select requirement entries with text lengths exceeding 800 characters and construct three datasets: (1) Original, which retains the original text without any summarization; (2) TextRank, which applies the TextRank algorithm to generate summaries; and (3) LLM, which utilizes

Table 2: Configuration of models

Model	Hyper-parameter	Value
Random forest	n_estimators	100 (default)
KNeighbors	n_neighbors	5
Naive Bayes	alpha	1.0
Logistic	penalty	l2
	tol	1e-4
	max_iter	None
SVC	kernel	rbf (default)
	C	1.0
CNN	Filters in convolution layer	{64, 128, 256, 512}
	Kernel size in convolution layer	3
	Number of units(Dense)	{256, 128}
LSTM	Number of units	{32, 64, 128}
	Dropout rate	0.2
GPT-3.5,GPT-4o	temperature	1 (default)
	max_tokens	256 (default)
DeepSeek-r1	temperature	0.7 (default)
	max_tokens	4096 (default)
claude-3	temperature	1 (default)
	top_p	0.9 (default)
	max_tokens	1024 (default)
MLP	Number of units	{256, 128, 64}
	Activation function	relu
	Dropout rate	0.2
LoRA	r	8
	lora_alpha	128
	lora_dropout	0.1
P-tuning	num_virtual_tokens	128
	encoder_num_layers	16
PromptTuning	num_virtual_tokens	12

a LLM to produce summaries. In the experiment, we employ three different-sized LLaMA models (1.1B, 7B, and 13B) for the requirement traceability task and compare the performance of different summarization strategies.

Table 2 presents the configurations of all models. All fine-tuning experiments are conducted on four NVIDIA RTX A4000 GPUs. To ensure the broad applicability and reliability of our research findings, we conduct experiments on datasets from various projects, encompassing both intra-domain cross-projects and cross-domain scenarios. Specifically, cross-domain experiments test performance in diverse scenarios, while intra-domain cross-project experiments provide more precise evaluations within specific domain projects. By combining these methods, we gain a comprehensive understanding of the methods' strengths and weaknesses, thereby enhancing the credibility and persuasiveness of our results.

4.4. Experiment results and analysis

4.4.1. Results for RQ1

In delving into the performance of the LLaMA model on different datasets, we first need to make a clear point: while the size of the model reflects its capabilities to some extent, large-scale experiments have confirmed that the quality and size of the training dataset tend to have a much more significant impact on model performance. As shown in the studies on BERT, the characteristics of the dataset are often the primary determinants of the model's performance, not just the size of the model [36].

Based on this perspective, we compare the performance of three different sizes of LLaMA models (1.1B TinyLLaMA, 7B, and 13B) across multiple datasets, both with and without fine-tuning. Additionally, we consider fine-tuning strategies such as LoRA, P-Tuning, and Prompt-Tuning to enhance model performance on specific tasks through parameter adjustments.

To ensure a consistent evaluation, 20% of the original (non-augmented) dataset is reserved as the test set across all experiments, while the remaining data is used for training and validation under varying splits (9:1, 8:2, 5:5, and 2:8). Given the large number of models, fine-tuning strategies (LoRA, P-tuning, Prompt tuning), and data split configurations involved, we provide a high-level comparative overview of model performance using box plots (Figure 11). Each box plot illustrates the distribution of F1 scores achieved by different LLaMA models under a specific fine-tuning technique across the various train-validation ratios.

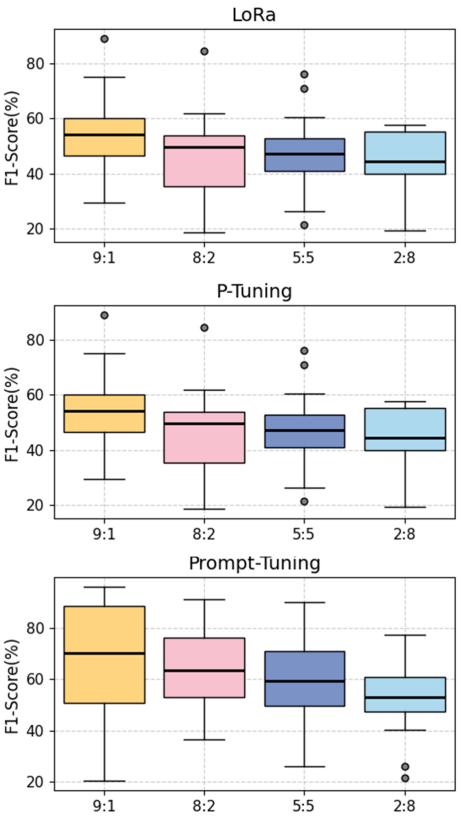


Figure 11: Overall Result in RQ1

The figure clearly reveals that **as the proportion of training data increases (e.g., from 2:8 to 9:1), the overall performance across all fine-tuning methods tends to improve**, evidenced by a higher median and a more compact interquartile range within the box plots. This trend indicates that with more training samples available, the fine-tuning methods can better exploit the capacity of the model, resulting in more stable and superior F1 performance. In particular, under the 9:1 setting, all methods show a more concentrated and elevated distribution of F1 scores, further confirming the positive impact of increased training data on fine-tuning effectiveness.

The complete experimental results are available at https://github.com/Gechuyan/Cross-Level-Requirements-Tracing-B

ased-on-Large-Language-Model.

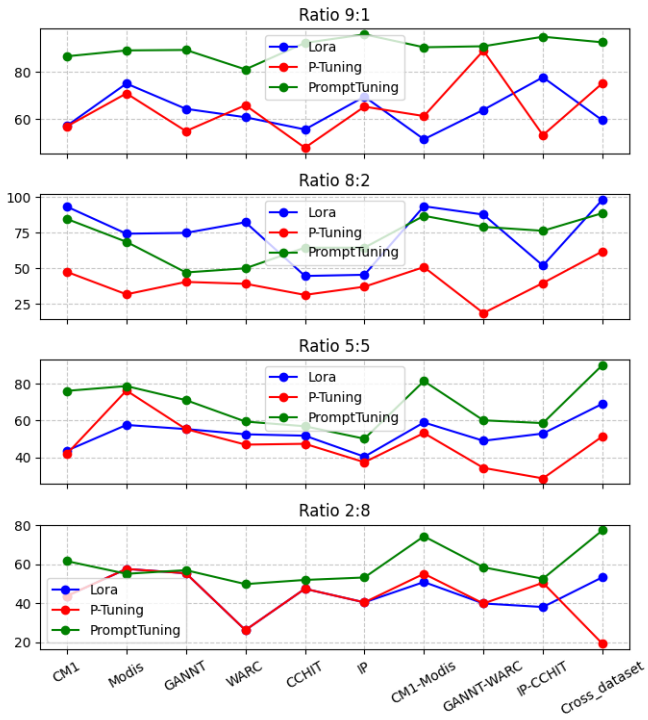


Figure 12: Performance of 7B-LLaMA Comparison of Different Strategies

Figure12, presents a comparison of F1 scores for the 7B-sized model across different datasets, evaluating various fine-tuning techniques under different training-test ratios. As illustrated in the figure, model performance generally declines as the training data ratio decreases. This trend is particularly evident when comparing Prompt-Tuning with LoRA and P-Tuning under varying conditions.

When the training set proportion is high (9:1, 8:2), Prompt-Tuning consistently outperforms LoRA and P-Tuning, maintaining strong generalization across datasets. At a 9:1 ratio, Prompt-Tuning achieves the highest accuracy across all datasets, reaching 95.82% on IP, 92.47% on Cross-project, and 92.21% on CCHIT. These results represent a 6-35% improvement over LoRA and P-Tuning in most cases. Even at an 8:2 ratio, Prompt-Tuning maintains a competitive edge, with 88.70% on Cross-project, 86.83% on CM1-Modis, and 79.09% on GANNT-WARC. These results suggest that Prompt-Tuning efficiently utilizes available labeled data to learn robust feature representations, particularly for large-scale models like LLaMA-13B.

As the amount of training data decreases, the performance of all models declines, but Prompt-Tuning exhibits greater resilience compared to LoRA and P-Tuning. At a 5:5 ratio, LoRA and P-Tuning experience severe degradation, with P-Tuning dropping to 34.44% on GANNT-WARC, while Prompt-Tuning remains significantly higher at 60.16%. At an even more extreme 2:8 ratio, while most models fail to maintain accuracy, Prompt-Tuning still scores 74.37% on CM1-Modis and 77.43% on Cross-project, whereas LoRA and P-Tuning drop below 55% in many cases. This robustness of Prompt-Tuning can be attributed to its architectural characteristics. Unlike LoRA and P-Tuning, which introduce additional trainable parameters into the model's internal layers or embeddings, Prompt-Tuning only optimizes a small set of soft prompt vectors while keeping the pre-trained model entirely frozen. This minimalist adaptation reduces the risk of overfitting, especially under data-scarce conditions, and helps retain generalization capabilities learned from pre-training. In contrast, the larger number of trainable parameters in LoRA and P-Tuning may require more data to converge effectively, leading to performance degradation when training samples are limited.

These observations indicate that Prompt-Tuning is not only superior when data is abundant but also significantly more robust in low-data scenarios. This advantage is likely due to its ability to better leverage pre-trained knowledge and adapt to new tasks with minimal data, making it particularly valuable for real-world applications where labeled data is scarce.

Table 3 presents the experimental results under a 9:1 training-validation ratio. Our chosen TinyLLaMA is fine-tuned on the UltraChat dataset, synthetic dialogues generated by ChatGPT. Since these dialogues are not closely related to our task, the performance is suboptimal. The performance of the TinyLLaMA model without fine-tuning is nearly the worst across all datasets. Although the performance of the model improves after fine-tuning, the overall advantage remains insignificant. Notably, the F1 scores of the TinyLLaMA model exceed 70% on the GANNT dataset after fine-tuning with the LoRA technique, and surpass 60% on the WARX dataset following fine-tuning with the Prompt-tuning technique.

In comparison, the overall performance of the 13B model is superior to that of the TinyLLaMA, with F1 scores consistently above 50%. However, the improvement from fine-tuning is not significant and, in some cases, even shows a decline. For example, on the Modis dataset, the F1 score of the 13B model without fine-tuning exceeds 60%, but drops to 48.08% after P-tuning and 33.33% after PromptTuning. We hypothesize that this decline is due to the insufficient volume of our dataset to effectively support the optimization of the 13B model. Additionally, the data augmentation techniques we employ may introduce noise, further impacting model performance. These findings suggest that the 13B model exhibits poor generalization ability for this specific task.

The 7B model outperforms the TinyLLaMA in direct classification tasks and demonstrates performance comparable to the 13B model. Notably, the performance of the 7B model significantly improves after fine-tuning. In experiments across various datasets, the 7B model fine-tuned with Prompt-tuning consistently achieves the best performance, with F1 scores exceeding 90% on multiple datasets. Additionally, the 7B model fine-tuned with Prompt-tuning demonstrates relatively stable performance across various dataset groups. In three cross-project datasets within the same domain, all metrics exceed 90%, even surpassing the performance on single-project datasets. This is likely because the larger data volume in cross-project datasets allows for more comprehensive training of the LLM, highlight-

Model	Fine-Tuning	CM1				Modis			
		Accuracy	**Precision**	**Recall**	**F1_Score**	**Accuracy**	**Precision**	**Recall**	**F1_Score**
7B	Without-Tuning	51.69%	53.11%	45.27%	48.88%	47.22%	81.82%	45.76%	58.70%
	Lora	57.98%	57.18%	57.98%	57.13%	75.00%	75.00%	75.00%	75.00%
	P-Tuning	57.14%	57.25%	57.14%	56.87%	75.00%	82.14%	75.00%	70.83%
	PromptTuning	**86.55%**	**86.55%**	**86.55%**	**86.55%**	**87.50%**	**90.63%**	**87.50%**	**89.04%**
13B	Without-Tuning	44.92%	66.34%	41.64%	51.16%	45.83%	93.94%	45.59%	61.38%
	Lora	59.66%	35.60%	59.66%	44.59%	75.00%	56.25%	75.00%	64.29%
	P-Tuning	53.78%	28.92%	53.78%	37.61%	62.50%	39.06%	62.50%	48.08%
	PromptTuning	61.34%	37.63%	61.34%	46.64%	50.00%	25.00%	50.00%	33.33%
1.1B	Without-Tuning	56.09%	0.93%	33.33%	1.81%	54.17%	0.00%	0.00%	0.00%
	Lora	69.44%	48.23%	69.44%	56.92%	62.50%	39.06%	62.50%	48.08%
	P-Tuning	59.66%	35.60%	59.66%	44.59%	62.50%	39.06%	62.50%	48.08%
	PromptTuning	57.98%	33.62%	57.98%	42.56%	37.50%	14.06%	37.50%	20.45%
		GANNT				**WARC**			
		Accuracy	**Precision**	**Recall**	**F1_Score**	**Accuracy**	**Precision**	**Recall**	**F1_Score**
7B	Without-Tuning	37.68%	80.23%	30.80%	44.52%	51.82%	**90.00%**	52.78%	66.54%
	Lora	75.00%	56.25%	75.00%	64.29%	61.11%	60.75%	61.11%	60.74%
	P-Tuning	67.86%	46.05%	67.86%	54.86%	66.67%	72.22%	66.67%	65.83%
	PromptTuning	**89.29%**	89.20%	**89.29%**	**89.24%**	**80.56%**	81.44%	**80.56%**	**81.00%**
13B	Without-Tuning	31.16%	**100.00%**	31.16%	47.51%	54.34%	40.00%	60.80%	48.25%
	Lora	67.86%	46.05%	67.86%	54.81%	67.86%	46.05%	67.86%	54.81%
	P-Tuning	71.43%	51.02%	71.43%	59.52%	71.43%	51.02%	71.43%	59.52%
	PromptTuning	64.29%	41.33%	64.29%	50.31%	64.29%	41.33%	64.29%	50.31%
1.1B	Without-Tuning	55.43%	27.91%	28.24%	28.07%	48.74%	6.32%	70.59%	11.59%
	Lora	78.57%	83.33%	78.57%	71.87%	41.67%	17.36%	41.67%	24.51%
	P-Tuning	75.00%	56.25%	75.00%	64.29%	61.11%	37.35%	61.11%	46.36%
	PromptTuning	64.29%	41.33%	64.29%	50.32%	63.89%	63.82%	63.89%	63.85%
		CCHIT				**IP**			
		Accuracy	**Precision**	**Recall**	**F1_Score**	**Accuracy**	**Precision**	**Recall**	**F1_Score**
7B	Without-Tuning	48.43%	47.86%	38.29%	42.54%	60.53%	15.73%	48.28%	23.73%
	Lora	55.56%	55.56%	55.56%	55.56%	69.57%	69.36%	69.57%	69.46%
	P-Tuning	50.00%	64.40%	50.00%	47.81%	65.22%	65.22%	65.22%	65.22%
	PromptTuning	**91.67%**	**92.75%**	**91.67%**	**92.21%**	**95.65%**	**95.99%**	**95.65%**	**95.82%**
13B	Without-Tuning	59.83%	5.00%	46.67%	9.03%	42.11%	83.15%	38.74%	52.85%
	Lora	52.78%	27.85%	52.78%	36.46%	65.22%	42.53%	65.22%	51.49%
	P-Tuning	55.56%	30.86%	55.56%	39.68%	39.13%	77.87%	39.13%	52.09%
	PromptTuning	75.00%	74.07%	75.00%	74.53%	69.57%	69.91%	69.57%	69.74%
1.1B	Without-Tuning	40.46%	39.18%	89.29%	54.46%	38.60%	38.77%	98.88%	55.70%
	Lora	69.44%	64.70%	69.44%	66.99%	56.52%	57.68%	56.52%	57.09%
	P-Tuning	66.67%	44.44%	66.67%	53.33%	56.52%	31.95%	56.52%	40.82%
	PromptTuning	58.33%	49.68%	58.33%	53.66%	65.22%	42.53%	65.22%	51.49%
		CM1-Modis				**GANNT-WARC**			
		Accuracy	**Precision**	**Recall**	**F1_Score**	**Accuracy**	**Precision**	**Recall**	**F1_Score**
7B	Without-Tuning	59.17%	20.06%	44.63%	27.68%	38.99%	96.47%	39.11%	55.66%
	Lora	52.59%	50.28%	52.59%	51.41%	63.89%	63.88%	63.89%	63.88%
	P-Tuning	60.74%	61.74%	60.74%	61.24%	88.89%	89.01%	88.89%	88.95%
	PromptTuning	**90.37%**	**90.38%**	**90.37%**	**90.37%**	**90.28%**	**91.32%**	**90.28%**	**90.80%**
13B	Without-Tuning	41.28%	88.45%	39.45%	54.56%	39.97%	99.65%	39.77%	56.85%
	Lora	62.96%	62.83%	62.96%	62.89%	58.33%	68.12%	58.33%	62.85%
	P-Tuning	55.56%	54.39%	55.56%	54.97%	60.25%	60.28%	60.25%	60.26%
	PromptTuning	80.00%	80.06%	80.00%	80.03%	80.56%	80.56%	80.56%	80.56%
1.1B	Without-Tuning	60.43%	58.33%	2.61%	5.00%	43.62%	39.04%	74.91%	51.33%
	Lora	81.48%	81.49%	81.48%	81.48%	80.56%	83.38%	80.56%	81.95%
	P-Tuning	47.41%	41.33%	47.41%	44.16%	58.33%	34.03%	58.33%	42.98%
	PromptTuning	60.00%	56.09%	60.00%	57.98%	69.44%	48.23%	69.44%	56.92%
		CCHIT-IP				**Cross-project**			
		Accuracy	**Precision**	**Recall**	**F1_Score**	**Accuracy**	**Precision**	**Recall**	**F1_Score**
7B	Without-Tuning	48.97%	70.00%	41.49%	52.10%	47.11%	31.72%	33.04%	32.37%
	Lora	77.59%	77.74%	77.59%	77.66%	60.38%	58.50%	60.38%	59.43%
	P-Tuning	53.45%	52.89%	53.45%	53.16%	75.09%	75.17%	75.09%	75.13%
	PromptTuning	**94.83%**	**94.91%**	**94.83%**	**94.84%**	**92.45%**	**92.49%**	**92.45%**	**92.47%**
13B	Without-Tuning	52.24%	52.17%	41.81%	46.42%	53.95%	69.70%	45.02%	54.71%
	Lora	60.34%	60.52%	60.34%	60.43%	66.04%	67.39%	66.04%	66.71%
	P-Tuning	63.79%	54.38%	63.79%	58.71%	58.11%	55.38%	58.11%	56.71%
	PromptTuning	70.69%	71.17%	70.69%	70.93%	83.77%	84.18%	83.77%	83.97%
1.1B	Without-Tuning	55.17%	41.28%	30.87%	35.32%	40.05%	37.61%	76.33%	50.39%
	Lora	79.31%	79.59%	79.31%	79.45%	84.91%	85.03%	84.91%	84.97%
	P-Tuning	46.55%	21.67%	46.55%	29.57%	41.89%	56.08%	41.89%	47.96%
	PromptTuning	48.27%	23.31%	48.27%	31.44%	60.75%	66.27%	60.75%	63.39%

Table 3: Result of RQ1-Experimental Results with a 9:1 Train-Validation Split

ing its strong capability to learn domain-specific knowledge. Furthermore, in cross-domain datasets, the 7B model fine-tuned with Prompt-tuning achieves metrics exceeding 92%, indicating strong generalization ability across different domains.

In conclusion, our 7B model, utilizing the Prompt-tuning technique, demonstrates superior performance across various datasets in our experiments. Furthermore, our findings suggest that increased model size does not necessarily correlate with improved results.

> Higher training data ratios (such as 9:1) further enhance performance, while lower ratios (like 5:5 and 2:8) lead to a decline. The results indicate that increasing model size does not always lead to better performance. The 7B model with Prompt-tuning achieved the best results.

4.4.2. Results for RQ2

This section presents the experimental results addressing RQ2, which aims to evaluate the effectiveness of our approach in comparison with a range of baselines and state-of-the-art methods. The hyperparameter configurations used for the involved models are summarized in Table 2.

4.4.2.1. Results for RQ2.1. This section compares the performance of our proposed method with traditional methods. Our method is evaluated against seven IR-based methods, five machine learning methods, and two deep learning methods. All experiments are conducted on ten datasets augmented through data enhancement as those employed in RQ1, with the training-to-test ratio set to 9:1 for both machine learning and deep learning models.

Specifically, we reproduce seven IR-based methods: Latent Dirichlet Allocation (LDA), Latent Semantic Indexing (LSI), Okapi BM25 (BM25), Jensen-Shannon divergence (JS), Latent Semantic Analysis (LSA), Jelinek-Mercer smoothing (JM), and Dirichlet smoothing (DIR). The similarity between high-level and low-level requirements is computed, and the top 10 ranked requirement pairs are considered traceable. Figure 13 presents a heatmap illustrating the requirement traceability performance of these methods across different datasets, where darker colors indicate better performance.

The Precision metric evaluates the proportion of true positive samples among the predicts positive samples. In this metric, our proposed LLM fine-tuning method demonstrates superior overall performance. The LSI-based method, although achieving over 90% precision in the WARC, CCHIT, IP, and cross-project datasets from the aerospace and computer domains, performs poorly in some datasets, such as achieving only 2.75% precision in the CM1 dataset and 17.65% in the cross-domain dataset, indicating poor stability. The Jelenik-Mercer method shows the worst performance, with precision exceeding 60% only in the cross-domain dataset and the CCHIT and IP datasets, while the precision in other datasets is below 30%. Recall is another crucial metric for evaluating the effectiveness of methods, as it reflects the ability to correctly predict all positive samples. In our experiments, the seven IR-based methods significantly underperform compared to our LLM fine-tuning method. The recall rates are particularly poor in cross-project and cross-domain datasets. For instance, in the CM1-Modis dataset from the aerospace cross-project dataset, the Jelenik-Mercer method has the lowest recall at just 0.52%, and the highest recall achieved by the LSA method is only 15.54%. In the cross-domain dataset, our proposed method achieves a recall rate of 92.45%, whereas the best-performing IR-based method, LSA, achieves only 23.02%. The comparison of recall rates reveals that traditional IR-based methods for calculating similarity do not effectively identify traceability relationships between requirements. Furthermore, their performance deteriorates as the volume of requirement data increases.

In contrast, proposed methods based on LLM can better understand the semantics and context of requirement texts. Through the robust representation capabilities of deep learning models, they can capture deeper associations between requirements. LLMs can learn more language and domain knowledge during training, providing stronger generalization ability and maintaining high accuracy and recall rates on cross-project datasets. Additionally, LLMs can integrate contextual information to make comprehensive judgments, reducing false positives and improving the overall performance of requirement tracing.

Machine learning methods are also widely used in requirement traceability tasks. We reproduced the classical machine learning approach from Du et al. [25] and compared the performance of various models, including Random Forest (RF), K-Nearest Neighbors (KNN), Naïve Bayes (NB), Logistic Regression (LR), and Support Vector Classifier (SVC).

Figure 14 illustrates the performance of different datasets in ML experiments, where a heatmap is used to visualize the experimental results. In our machine learning experiments, we find that overall performance is superior to the IR-based experiments. Meanwhile, our proposed LLM fine-tuning method remains the best performer. In this set of experiments, five machine learning classifiers achieve accuracy greater than 60% across all datasets. We observe that Random Forest and SVC achieve a 100% recall on the Modis and aerospace cross-project dataset CM1-Modis, but their corresponding precision is relatively low at 75% and 54.55%, respectively. This typically indicates that while the model predicts many positive samples, a significant portion of these are false positives. Simultaneously, Random Forest,KNN, and SVC classifiers achieve 100% precision on the CM1 dataset, but their corresponding recall rates are only 75%, 56.25%, and 62.5%. This situation indicates that while all predicts positive samples are correct (i.e., no false positives), the models fail to identify all actual positive samples.

The SVC achieve a 100% recall rate and an F1 score of 91.14% on the GANNT-WARC dataset in the computer domain, slightly higher than our proposed method (91.14% vs. 90.8%). However, this classifier demonstrated poor stability, with an F1 score of only 52.63% on the GANNT dataset and failing to reach 77% in other single-project dataset experiments. Additionally, it is observed that the performance of each machine learning model on cross-domain datasets is relatively poor, with the highest F1 score from SVC being only 66.1%, which is 26.37% lower than our proposed method (92.47% vs. 66.1%).

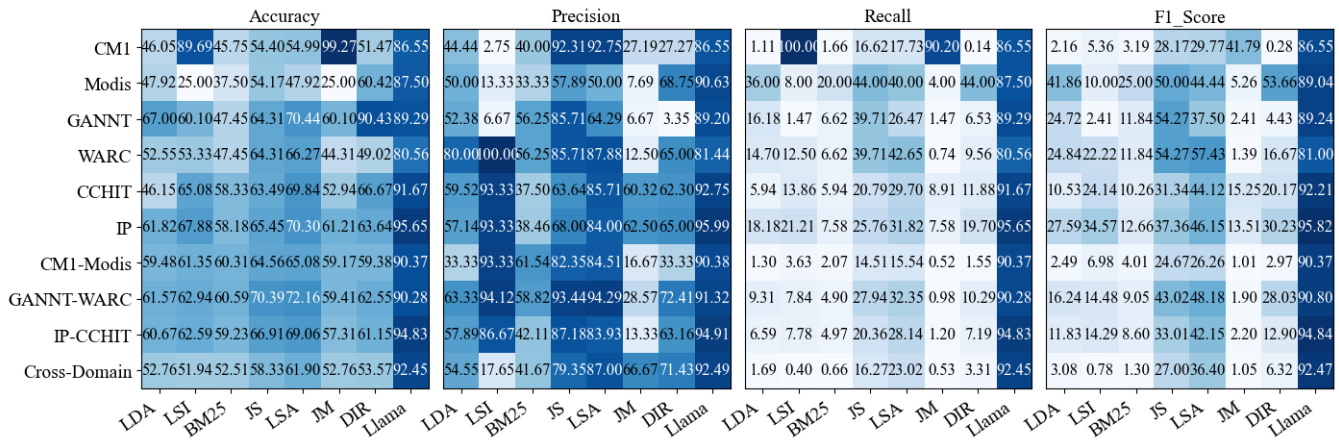


Figure 13: Result of RQ2.1:Comparison with IR-Based-Methods

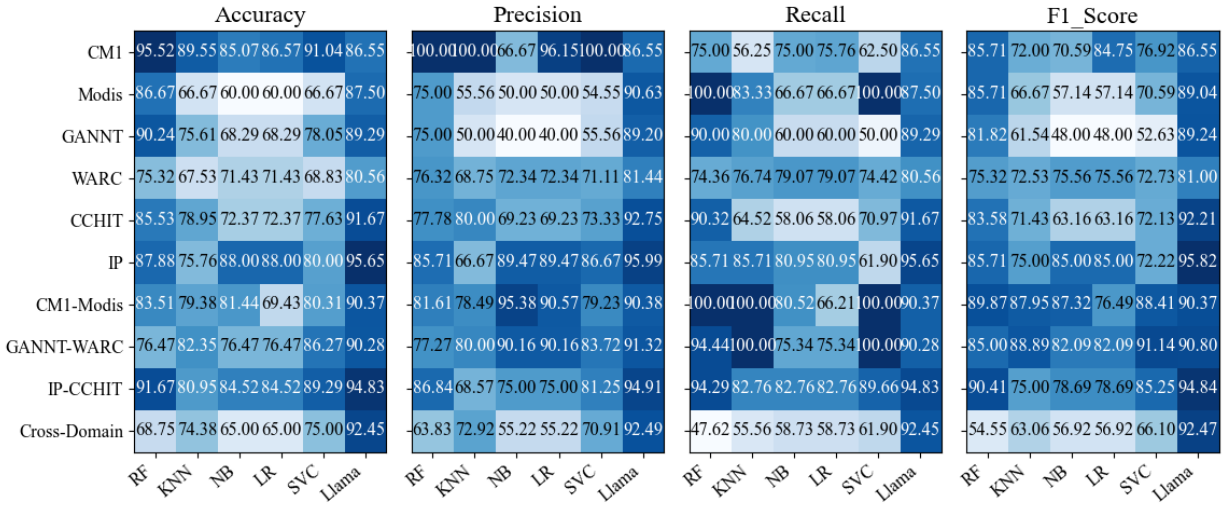


Figure 14: Result of RQ2.1:Comparison with ML-Based-Methods

Although these traditional machine learning methods performed well on some specific datasets, our proposed LLM fine-tuning method consistently demonstrated the best performance across all datasets, exhibiting extremely high precision and recall rates with minimal fluctuation across various metrics. This indicates its superior stability and generalization capability. This advantage is particularly evident on cross-project and cross-domain datasets, showing that our model maintains high efficiency when handling different types of data.

Figure 15 displays the results of the comparative experiments involving deep learning models. In our deep learning-based experiments, the same data processing methods as those used by the LLM group are employed, including tokenizing the requirement texts for vectorization. The results indicate that the fine-tuned LLaMA model consistently achieves the best performance in this set of experiments.

It is found that traditional deep learning models do not achieve metrics above 70% on any dataset, although their performance remains relatively stable across different datasets. The CNN model performs well on the CCHIT and IP datasets, with F1-scores of 56.36% and 54.26%, respectively. This indicates that the CNN model effectively captures features in medical data. However, its F1-score drops to 44.37% on cross-project datasets, highlighting its limited generalization ability in cross-project tasks. The LSTM model achieves F1-scores above 50% on three cross-project datasets, but its F1-score drops to 44.77% on cross-domain datasets, indicating that while the LSTM model is relatively stable on cross-project data, its generalization ability across domains is poor.

On the CM1 dataset, the recall rates of the CNN, MLP, and LSTM models are 55.7%, 55.27%, and 60.34%, respectively, but their precision rates are only 31.02%, 30.55%, and 36.41%, respectively. This high recall but low precision phenomenon indicates that the models can identify most positive samples but also produce a large number of false positives. On cross-domain datasets, the recall and precision of the CNN model are 60.57% and 36.68%, respectively, again demonstrating the model's strong ability to identify positive samples but with a high false positive rate.

1. IR-based methods generally suffer from low recall. While LSA performs best among them, it remains far behind our fine-tuned 7B model with Prompt-Tuning.
2. The 7B model fine-tuned with Prompt-tuning outperforms the best SVC model from traditional machine learning methods by 26.37% in F1 score, despite the SVC model's instability.
3. Deep learning models perform poorly and fail to capture semantic information effectively. The 7B model fine-tuned with Prompt-tuning still achieves the best performance.

4.4.2.2. Results for RQ2.2. In this experiment, we investigate the impact of different prompt templates on the performance of state-of-the-art LLMs, including GPT-3.5, GPT-4o, and DeepSeek-R1, in requirements traceability tasks.

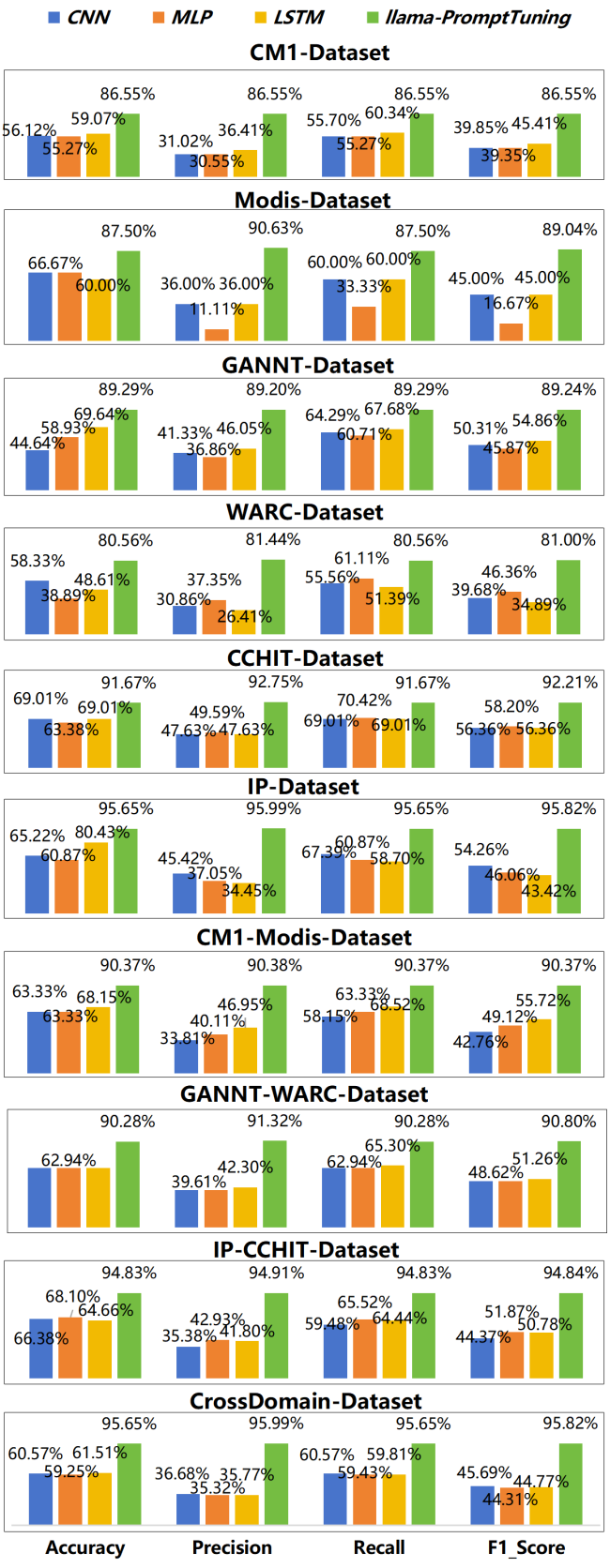


Figure 15: Result of RQ2.1:Comparison with DL-Based-Methods

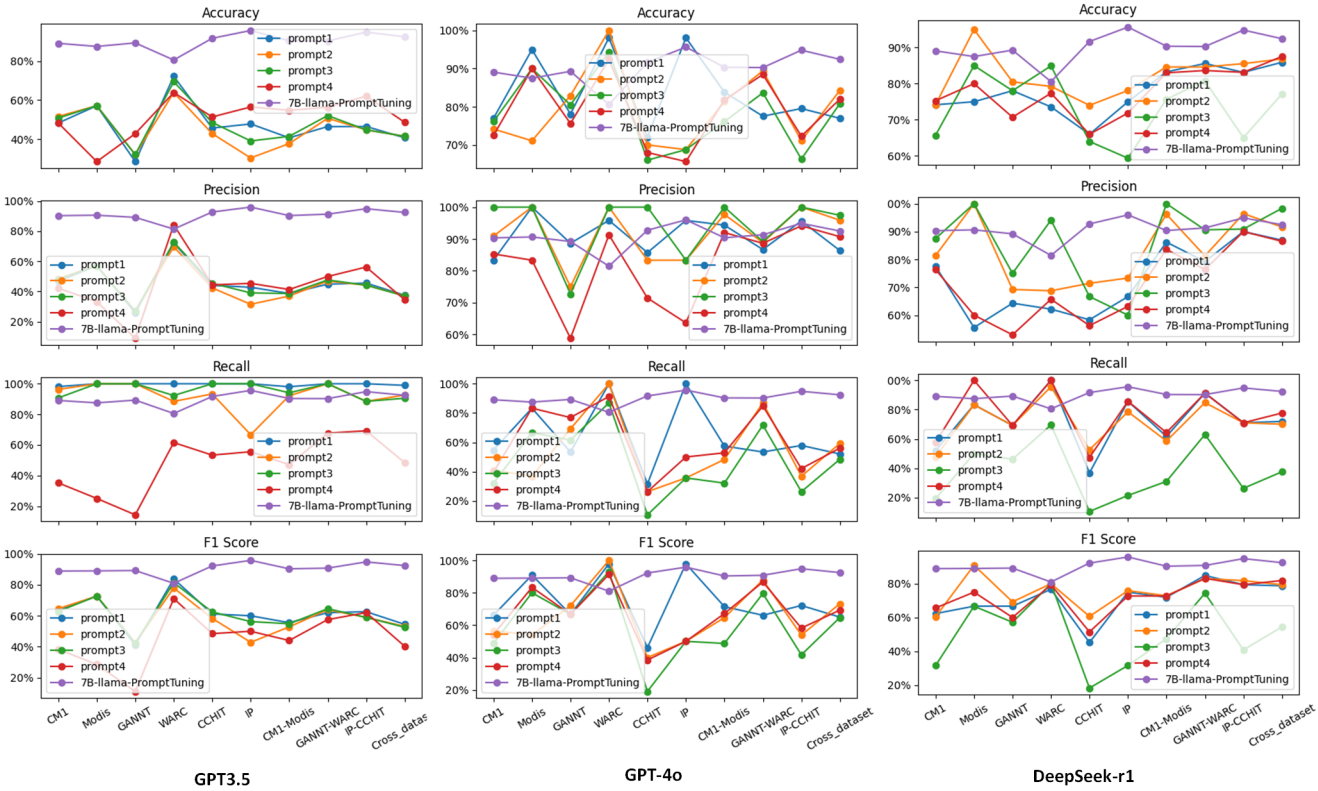


Figure 16: Result of RQ2.2:Performance of GPT-3.5, GPT-4o, and DeepSeek-r1 under different prompt templates

Based on the prompt design methods outlined in the paper[10], we select four different prompt template designs: Basic Prompt, Basic Prompt + Simple Explanation, Basic Prompt + Conditional Explanation, and Step-by-Step Prompt. These four prompt templates are illustrated at bellow:

- **Prompt-1** Basic Prompt: Consider the following scenario where 'high requirement' and 'low requirement text' represent high-level and low-level requirements of the software system, respectively. Answer with Yes or No: Does the low-level requirement trace back to the high-level requirement?
- **Prompt-2** Basic Prompt + Simple Explanation: Consider the following scenario where 'high requirement' and 'low requirement text' represent high-level and low-level requirements of the software system, respectively. Answer with Yes or No: Does the low-level requirement trace back to the high-level requirement? Please explain your answer in 10 words.
- **Prompt-3** Basic Prompt + Conditional Explanation: Consider the following scenario where 'high requirement' and 'low requirement text' represent high-level and low-level requirements of the software system, respectively. Answer with Yes or No: Does the low-level requirement trace back to the high-level requirement? If the answer is no, please explain your answer in 10 words.
- **Prompt-4** Step-by-Step Prompt: Consider the following scenario where the 'high requirement' and the 'low requirement text' represent high and low level requirements of the software system, respectively. Let's think step by step: 1. Identify the key elements of the high level requirement; 2. Match these elements with the low-level requirement; 3. Highlight any gaps; 4. Infer implicit traceability; Finally, answer with Yes or No if there is a traceability relationship, and provide a brief rationale in about 10 words.

The results of our experiments are shown in a line graph in Figure 16. Our experimental results indicate that the GPT-3.5 model exhibits the lowest sensitivity to prompt templates, as its performance remains relatively stable across different prompts. However, its overall effectiveness is limited. Specifically, when applying Prompt 1 to the CM1 dataset, GPT-3.5 achieves a high recall rate of 98.15% but with a relatively low precision of 46.90%. On the Modis dataset, its recall reaches 100%, yet its precision is only 57.14%.

For Prompt 2, the recall rate on the Modis, GANTT, and GANTT-WARC datasets remains at 100%, but precision does not exceed 50%, with the GANTT dataset performing particularly poorly at only 26.92%. Using Prompt 3, recall rates consistently exceed 90% across all datasets except for IP-CCHIT, which has a recall of 88.46%. The highest precision is observed on the WARC dataset (73.73%), followed by Modis (57.14%), while the remaining datasets exhibit precision below 50%. These results suggest that although GPT-3.5 effectively identifies pos-

itive samples, it also generates a high number of false positives.

GPT-4o demonstrates superior performance compared to GPT-3.5, but its stability is inconsistent. For example, in the CCHIT dataset, it achieves a precision of 100% but with a recall of only 10.53%. A similar pattern is observed in the IP-CCHIT dataset, indicating that the model struggles to maintain a balance between precision and recall. For the DeepSeek-R1 model, the third prompt template shows the least stability, whereas the other templates exhibit relatively consistent performance. Notably, when applying the second prompt to the Modis dataset, the model achieves 100% precision and an F1-score of 90.91%, slightly outperforming our proposed approach. However, this high level of performance is not maintained across other datasets.

Our proposed approach, 7B PromptTuning, consistently outperforms the other three models across all metrics. It demonstrates an exceptional ability to balance precision and recall, achieving high recall rates while minimizing false positives. This makes 7B PromptTuning a more reliable choice for ensuring traceability across evaluation datasets.

> Different LLMs exhibit varying sensitivities to prompt templates. Compared to GPT-4o and DeepSeek-r1, 7B LLaMA with PromptTuning improves F1 by 16.27% and 16.8% respectively on cross-domain datasets. It also shows higher recall and fewer false positives, indicating better stability in requirement traceability tasks.

4.4.2.3. Results for RQ2.3. In the RQ2.2 experiment, we found that simple prompt engineering techniques alone do not significantly improve the predictive accuracy of LLMs. The study by [15] represents the most advanced research on cross-level requirement traceability to date, as it explores various prompt strategies to enhance LLM performance. Specifically, the authors tested OpenAI's text-davinci-003 model for predicting traceability links and demonstrated that, despite requiring slight modifications in prompts, its performance was comparable to Claude Instant (claude-instant-1.0) from Anthropic.

Given the comprehensiveness of this study, we adopt its methodology as our baseline for comparison. However, upon attempting to fully reproduce the authors' work, we discovered that claude-instant-1.0 is no longer available. As a result, we employ two more recent LLMs, claude-3-5-haiku-20241022 and claude-3-7-sonnet-20250219, for our evaluation.

Following the prompt engineering strategy outlined in [15], we apply the same prompt template: ***Below are artifacts from the same software system. Ignoring different levels of abstraction, can (2) be one of the hierarchical decompositions of (1)? Consider whether (2) implements a part of (1).***

Table 4 presents a comparison between our approach and this baseline method, highlighting the limitations of prompt-based techniques in improving LLM accuracy for requirement traceability tasks.

7B-PromptTuning achieves the highest accuracy across all datasets, reaching a maximum of 95.65% (IP dataset) and a minimum of 80.56% (WARC dataset), significantly outperforming the Claude Baseline methods. Additionally, in terms of F1 Score, 7B-PromptTuning consistently surpasses the Claude Baseline models across all datasets, demonstrating its stability and generalization capability. Furthermore, in terms of Precision and Recall, 7B-PromptTuning maintains a better balance, avoiding the issue observed in the Claude Baseline models, where some models achieve extremely high recall at the cost of low precision.

Dataset	Model	Accuracy	Precision	Recall	F1 Score
CM1	claude-3-7-sonnet	79.78%	82.46%	64.38%	72.31%
	claude-3-5-haiku	68.54%	58.76%	78.08%	67.06%
	7B-PromptTuning	86.55%	86.55%	86.55%	86.55%
Modis	claude-3-7-sonnet	70.00%	50.00%	100.00%	66.67%
	claude-3-5-haiku	60.00%	42.86%	100.00%	60.00%
	7B-PromptTuning	87.50%	90.63%	87.50%	89.04%
GANNT	claude-3-7-sonnet	73.17%	55.56%	76.92%	64.52%
	claude-3-5-haiku	46.34%	36.36%	92.31%	52.17%
	7B-PromptTuning	89.29%	89.20%	89.29%	89.24%
WARC	claude-3-7-sonnet	77.36%	77.36%	91.30%	77.78%
	claude-3-5-haiku	62.26%	53.85%	91.30%	67.74%
	7B-PromptTuning	80.56%	81.44%	80.56%	81.00%
CCHIT	claude-3-7-sonnet	67.39%	55.56%	31.25%	40.00%
	claude-3-5-haiku	58.14%	45.45%	62.50%	52.63%
	7B-PromptTuning	91.67%	92.75%	91.67%	92.21%
IP	claude-3-7-sonnet	71.88%	66.67%	71.43%	68.97%
	claude-3-5-haiku	62.50%	54.17%	92.86%	68.42%
	7B-PromptTuning	95.65%	95.99%	95.65%	95.82%
CM1-Modis	claude-3-7-sonnet	81.30%	82.54%	59.77%	69.33%
	claude-3-5-haiku	69.51%	54.29%	87.36%	66.96%
	7B-PromptTuning	90.37%	90.38%	90.37%	90.37%
GANNT-WARC	claude-3-7-sonnet	87.50%	83.67%	89.13%	86.32%
	claude-3-5-haiku	83.65%	75.44%	93.48%	83.50%
	7B-PromptTuning	90.28%	91.32%	90.28%	90.80%
IP-CCHIT	claude-3-7-sonnet	81.48%	92.00%	63.89%	75.41%
	claude-3-5-haiku	79.49%	77.78%	77.78%	77.78%
	7B-PromptTuning	94.83%	94.91%	94.83%	94.84%
Cross-project	claude-3-7-sonnet	86.61%	88.37%	72.61%	79.72%
	claude-3-5-haiku	82.05%	69.61%	90.45%	78.67%
	7B-PromptTuning	92.45%	92.49%	92.45%	92.47%

Table 4: Result of RQ2.3:Comparison between the proposed approach and state-of-the-art approaches

Claude-3-7-Sonnet performs relatively well on some datasets, such as WARC and CM1, achieving an F1 Score between 72.31% and 77.78%. However, it still falls short compared to 7B-LLaMA with PromptTuning. Notably, this model attains a Recall of 100.00% on the Modis dataset but only a Precision of 50.00%, indicating a tendency to overpredict positive samples, leading to a high false positive rate. Claude-3-5-Haiku performs even worse than Claude-3-7-Sonnet, particularly on the CM1, GANNT, and CCHIT datasets, where its accuracy drops as low as 46.34% (GANNT dataset), highlighting its weaker generalization ability. Additionally, while this model exhibits extremely high recall on some datasets (e.g., 92.31% for GANNT and 92.86% for IP), its low precision suggests an excessive bias toward predicting positive samples, resulting in a high number of false positives.

The Claude Baseline methods overall exhibit a trend of high Recall but low Precision, particularly in the Modis and GANNT datasets, where Precision is significantly lower than Recall. This pattern indicates a predictive strategy that prioritizes high recall to cover all relevant samples but sacrifices precision, leading to more false positives. As a result, while the Claude models can ensure high Recall in certain cases, their Precision and F1 Score remain considerably lower than those of 7B-PromptTuning.

7B-PromptTuning demonstrates more stable performance,

consistently outperforming baseline models in terms of Accuracy and F1 Score while maintaining a balanced trade-off between Precision and Recall, reducing both false positives and false negatives. Its performance remains steady across all datasets without significant fluctuations, highlighting its superior generalization capability. Additionally, as the Claude models are proprietary and subject to potential API access restrictions or service discontinuation, their long-term reliability is uncertain.

> 7B-LLaMA with PromptTuning significantly outperforms the baseline models, including Claude-3-7-Sonnet and Claude-3-5-Haiku, across all datasets. In contrast, the baseline models demonstrate inconsistent performance, with notably high false positive rates and lower generalization capability, particularly on datasets like CM1 and GANNT.

4.4.3. Results for RQ3

This section explores the impact of data augmentation techniques on model performance. We first assess the effectiveness of different augmentation methods in enhancing dataset diversity, followed by evaluating their influence on fine-tuned LLMs across multiple datasets.

Although LLMs perform well in natural language understanding, they sometimes struggle with synonym recognition, active-passive transformations, and handling erroneous inputs. Moreover, different LLMs exhibit noticeable variations in practical recognition tasks.

To evaluate the sensitivity of LLMs to data augmentation, we applied four commonly used techniques—random synonym replacement, machine translation, random deletion, and random word swapping—to six project datasets, generating five augmented versions per dataset and forming a total of 30 data groups. We then used LLaMA models of varying scales (1.1B, 7B, and 13B) to assess prediction consistency. The results reveal that in over 68% of the cases, fewer than 10 samples retained identical predictions across all augmented and original versions, indicating that the models are sensitive to even minor textual perturbations. Notably, the 13B model showed higher prediction consistency with the original data but still failed to maintain robustness across all datasets. Conversely, the 7B model, despite achieving the best overall accuracy for RQ1, exhibited the weakest ability to differentiate augmented data. For instance, in the CM1 dataset, nearly half of the true positive and false positive predictions by the 7B model remained unchanged after augmentation, whereas this ratio dropped below 10% for other datasets. These results suggest that while data augmentation increases diversity, current LLMs—regardless of scale—struggle to generalize over perturbed inputs, underscoring the need for targeted fine-tuning. The complete experimental data are available at the GitHub repository (https://github.com/Gechuyan/Cross-Level-Requirements-Tracing-Based-on-Large-Language-Model).

To further demonstrate the effectiveness of data augmentation, we apply both the augmented dataset and the unaugmented dataset to LLMs for fine-tuning. Three different model sizes are evaluated using three distinct fine-tuning methods. The experimental results are illustrated in Table 5. From the experimental results, it is evident that augmented datasets exhibit significant performance differences compared to original datasets under various methods.

It is worth noting that in various datasets, the datasets augmented through the PromptTuning technique exhibit the most significant improvement compared to the original datasets. In single-project datasets, the recall in the IP project increases from 41.18% to 95.05%, and the F1 score improves from 46.7% to 95.82%. In intra-domain cross-project datasets, the most notable improvement is observed in the medical field IP-CCHIT dataset, where the F1 score increases from 50.74% to 94.84%. The P-Tuning technique generally achieves better performance with the augmented datasets across most datasets. However, some instances show decreased performance after data augmentation. For example, in the CCHIP dataset, when fine-tuning with P-Tuning, the precision and recall of the original dataset are 11.91% (76.31%-64.4%) and 7.69% (57.69%-50%) higher, respectively, than those of the augmented dataset. Nevertheless, for this dataset, the performance of other fine-tuning techniques on the augmented data is higher than that on the original data. Furthermore, in the IP dataset, the recall of the original dataset is slightly higher than that of the augmented dataset, but the precision is 20.76% (70.59%-49.83%) higher.

This indicates that data augmentation techniques exhibit varying performance across different datasets and fine-tuning methods. While data augmentation significantly improves model performance in most cases, there are instances where the performance of the augmented dataset is lower than that of the original dataset. This might be due to the introduction of certain noise or biases by the augmentation techniques, affecting the model's learning effectiveness. For instance, in the CCHIT dataset, when using the P-Tuning method, the precision and recall of the augmented dataset are lower than those of the original dataset, indicating that the P-Tuning method might be more sensitive to noise in this scenario.

Despite this, overall, data augmentation techniques play a crucial role in enhancing the robustness and generalization ability of models. In the IP dataset, although the recall of the original dataset is slightly higher, the precision of the augmented dataset is significantly improved, indicating the clear advantage of data augmentation in enhancing model precision. Especially in the PromptTuning technique, the augmented datasets show more significant performance improvements, further validating the effectiveness of data augmentation in this method.

> 1.Data augmentation significantly affects LLM performance, with models showing different adaptability.
> 2.Data augmentation effectively improves model performance, with PromptTuning showing more pronounced gains.

4.4.4. Results for RQ4

To assess the impact of text summarization on model performance, we conduct an ablation study using requirement items

Datasets	Method	Augmented Datasets				Original Datasets			
		Accuracy	Precision	Recall	F1_Score	Accuracy	Precision	Recall	F1_Score
CM1	LoRA	57.98%	57.18%	57.98%	57.13%	55.22%	57.25%	55.22%	56.21%
	P-Tuning	57.14%	57.25%	57.14%	56.87%	55.24%	55.24%	55.24%	55.24%
	PromptTuning	**89.08%**	**90.34%**	**89.08%**	**88.91%**	76.12%	76.12%	76.12%	76.12%
Modis	LoRA	75.00%	75.00%	75.00%	75.00%	60.00%	60.00%	60.00%	60.00%
	P-Tuning	75.00%	82.14%	75.00%	70.83%	60.00%	60.00%	60.00%	60.00%
	PromptTuning	**87.50%**	**90.63%**	**87.50%**	**89.04%**	80.00%	64.00%	80.00%	71.11%
GANNT	LoRA	75.00%	56.25%	75.00%	64.29%	47.62%	45.24%	47.62%	46.40%
	P-Tuning	67.86%	46.05%	67.86%	54.86%	57.14%	32.65%	57.14%	41.56%
	PromptTuning	89.29%	**89.20%**	89.29%	**89.24%**	**90.48%**	81.86%	**90.48%**	85.95%
WARC	LoRA	61.11%	60.75%	61.11%	60.74%	50.00%	51.63%	50.00%	50.80%
	P-Tuning	66.67%	72.22%	66.67%	65.83%	34.62%	38.52%	34.62%	36.47%
	PromptTuning	**80.56%**	**81.44%**	**80.56%**	**81.00%**	61.54%	63.65%	61.54%	62.58%
CCHIT	LoRA	55.56%	55.56%	55.56%	55.56%	53.85%	28.99%	53.85%	37.69%
	P-Tuning	50.00%	64.40%	50.00%	47.81%	57.69%	76.31%	57.69%	65.71%
	PromptTuning	**91.67%**	**92.75%**	**91.67%**	**92.21%**	57.69%	60.83%	57.69%	59.22%
IP	LoRA	69.57%	69.36%	69.57%	69.46%	58.82%	62.61%	58.82%	60.66%
	P-Tuning	65.22%	65.22%	65.22%	65.22%	70.59%	49.83%	70.59%	58.42%
	PromptTuning	**95.65%**	**95.99%**	**95.65%**	**95.82%**	41.18%	53.92%	41.18%	46.70%
CM1-Modis	LoRA	52.59%	50.28%	52.59%	51.41%	60.83%	58.82%	60.82%	59.80%
	P-Tuning	60.74%	61.74%	60.74%	61.24%	61.86%	61.16%	61.86%	61.51%
	PromptTuning	**90.37%**	**90.38%**	**90.37%**	**90.37%**	74.23%	75.61%	74.23%	74.91%
GANNT-WARC	LoRA	63.89%	63.88%	63.89%	63.88%	68.63%	47.10%	68.63%	55.86%
	P-Tuning	88.89%	89.01%	88.89%	88.95%	64.71%	67.11%	64.71%	65.89%
	PromptTuning	**90.28%**	**91.32%**	**90.28%**	**90.80%**	58.82%	58.66%	58.82%	58.74%
IP-CCHIT	LoRA	77.59%	77.74%	77.59%	77.66%	54.76%	29.99%	54.76%	38.76%
	P-Tuning	53.45%	52.89%	53.45%	53.16%	57.14%	32.65%	57.14%	41.56%
	PromptTuning	**94.83%**	**94.91%**	**94.83%**	**94.84%**	47.62%	65.12%	47.62%	50.74%
Cross-project	LoRA	60.38%	58.50%	60.38%	59.43%	58.42%	57.46%	58.42%	57.94%
	P-Tuning	75.09%	75.17%	75.09%	75.13%	62.63%	61.16%	62.63%	61.89%
	PromptTuning	**92.45%**	**92.49%**	**92.45%**	**92.47%**	76.32%	75.96%	76.32%	76.14%

Table 5: Result of RQ3: Comparison of augmented and original datasets across different methods

exceeding 800 characters in length. We find that among 177 data entries, the length of the bottom-level requirements surpasses this threshold. Due to the small sample size, and to avoid potential errors, we apply four positive data augmentation techniques to expand the dataset. Three datasets are created for comparison: the original dataset without summarization, a version summarized using the TextRank algorithm, and another generated through a LLM. We utilize three different LLaMA models (1.1B, 7B, and 13B) for the requirement traceability task and compare their performance under different summarization strategies.

Model	Data Type	TP	FN	FP	TN	Accuracy	Precision	Recall	F1 Score
1.1B	Original	109	223	192	330	51.41%	36.21%	32.83%	34.44%
	TextRank	99	203	170	352	**54.73%**	36.80%	32.78%	34.68%
	LLM	214	118	319	203	48.83%	**40.15%**	**64.46%**	**49.48%**
7B	Original	150	182	236	286	**51.05%**	38.86%	45.18%	41.78%
	TextRank	286	46	428	94	44.50%	**40.06%**	86.14%	**54.68%**
	LLM	301	31	469	53	41.45%	39.09%	**90.66%**	54.63%
13B	Original	262	70	420	102	42.62%	38.42%	78.92%	51.68%
	TextRank	303	29	458	64	**42.97%**	39.82%	91.27%	55.44%
	LLM	319	13	479	43	42.39%	**39.97%**	**96.08%**	**56.46%**

Table 6: RQ4: Performance comparison of different summarization techniques across various LLaMA models.

Table 6 presents the performance comparison of different summarization techniques across LLaMA models of varying sizes (1.1B, 7B, and 13B). The results indicate that summarization techniques generally improve recall and F1 scores compared to the original dataset. For instance, in the 1.1B model, the recall increases from 32.83% (Original) to 64.46% (LLM-based summarization), while the F1 score improves from 34.44% to 49.48%. Similarly, in the 13B model, LLM-based summarization achieves a recall of 96.08%, significantly higher than the 78.92% recall observed in the original dataset. However, these improvements come at the cost of increased false positives, leading to a decrease in precision. For example, in the 7B model, LLM-based summarization results in a precision of 39.09%, which is lower than the 38.86% obtained with the original dataset.

TextRank summarization also enhances performance, though to a lesser extent than LLM-based methods. In the 7B model, TextRank improves recall from 45.18% (Original) to 86.14%, but introduces more false positives, reducing accuracy from 51.05% to 44.50%. A similar trend is observed in the 13B model, where TextRank increases recall to 91.27%, while the accuracy remains nearly unchanged. Notably, in the 1.1B model, TextRank provides a more balanced improvement, with accuracy increasing from 51.41% (Original) to 54.73% and the F1 score rising from 34.44% to 34.68%.

The performance differences between TextRank and LLM-based summarization become less pronounced as model size increases. In the 13B model, for instance, the F1 scores of TextRank and LLM summarization are 55.44% and 56.46%, respectively, indicating that both techniques provide comparable benefits at larger scales. However, LLM-based summarization requires deploying an additional language model, which increases computational costs. Given the input length limitations of small-scale datasets and the high execution time of LLM-based summarization, TextRank is preferred for practical applications.

These results suggest that long-text summarization enhances large models' ability to handle requirement traceability tasks by improving contextual understanding and facilitating more accurate traceability predictions. LLM-based summarization methods perform better on smaller models. This may be because LLMs generate more contextually sensitive summaries, allowing models with fewer parameters to retain key informa-

tion more effectively. In contrast, the TextRank method relies on extractive summarization, which may not always capture the most informative parts of the text. While larger models mitigate this difference, the impact is more pronounced in smaller models. Therefore, when selecting a summarization method, both model size and computational feasibility should be considered. Both summarization techniques are viable options, with overall similar performance.

> Integrating summarization techniques enhances requirement traceability performance. LLM-based summarization benefits smaller models, while TextRank and LLM methods perform comparably overall.

4.5. Threats to Validity

The first threat to the validity is that this study only applies different versions of the LLaMA model. While these models provide significant insights, the findings may not generalize across other LLMs, such as GPT-3.5, BERT, or T5. The exclusive focus on LLaMA models means that our conclusions might not hold true for other architectures with different underlying mechanisms and training data. Future work will include experiments with a broader range of LLMs to ensure comprehensive validity and applicability, allowing for more robust and generalizable conclusions.

The secondary threat to validity involves the data augmentation techniques employed in this study, which might introduce some noise. Although our methods theoretically preserve the semantic meaning of sentences, we do not empirically validate this aspect. This potential introduction of noise could affect the robustness and accuracy of our results. Specifically, the augmented data might contain subtle semantic shifts or artifacts that could mislead the model during training. To address this issue, we perform manual validation of the augmented requirement texts in our experiments, aiming to minimize the risk of model misguidance.

Thirdly, a significant threat to validity is that the datasets used in this study, although diverse, may not cover all possible variations and complexities encountered in real-world requirement tracing tasks. Our datasets are confined to specific domains such as aerospace, computer, and medical fields. This domain specificity might limit the applicability of our findings to other fields with different types of requirements and project structures. Expanding the datasets to include more varied domains and larger samples will help to validate the generality and robustness of our proposed method.

Finally, while our results demonstrate that large-scale fine-tuning on high-quality traceability data can significantly improve performance, we acknowledge the practical challenge of acquiring sufficiently large and accurately traced datasets in real-world settings. This is a common limitation across many ML-based approaches to software traceability. In terms of practical applicability, we note that the datasets used in this study span domains such as aerospace, computer systems, and healthcare, which are characterized by relatively mature development processes and rigorous documentation practices. In these domains, the fine-tuned models we provide are close to usable out-of-the-box, especially in environments where data schemas and requirements share structural or semantic similarities with the training data. These models can be further adapted to organization-specific contexts with minimal additional fine-tuning, reducing the labeling burden for practitioners. However, for domains with less structured documentation practices or where traceability data is scarce or noisy, additional work may be required to improve model performance. This could include leveraging semi-supervised learning, active learning, or weak supervision techniques to reduce the cost of trace acquisition. We view this as a promising direction for future work, particularly in making traceability tools more accessible across a broader range of industrial contexts. Additionally, current AI platforms like Watson support the uploading of datasets and automatic fine-tuning, which enhances the practicality and scalability of AI models. This functionality makes it easier for practitioners to adapt models to specific needs and domains with minimal manual intervention, thus reducing deployment time and effort. It also enables a wider range of industries to adopt AI-driven solutions with minimal upfront cost and technical expertise.

5. CONCLUSIONS

In this paper, we introduce a novel cross-level requirement tracing method based on LLMs. Our approach simplifies traditional preprocessing by focusing solely on text summarization and vectorization, leveraging the deep semantic understanding capabilities of LLMs. We apply data augmentation techniques across six single-project datasets, three cross-project datasets within the same domain, and one cross-domain dataset, utilizing various fine-tuning methods on different scales of LLaMA models.

Empirical results demonstrate that our approach achieves F1 scores exceeding those of traditional IR, ML, and DL methods on cross-domain datasets. Specifically, it outperforms the best-performing IR techniques, LSA and JS, by 56.07% and 65.47% , respectively. Compared to the strongest ML model, SVC, our method yields a notable 26.37% improvement in F1 score. In the DL category, it surpasses the top CNN model by 46.78% . Moreover, our approach achieves significant improvements over GPT-4o and DeepSeek-r1, with F1 score gains of 16.27% and 16.8%, respectively, on cross-domain datasets. Compared to the baseline, it achieves a maximum improvement of 13.8% . These results highlight that for requirement traceability tasks, fine-tuned large models outperform methods relying solely on prompt templates.

In future work, we will focus on further enhancing the practical applicability of our approach by integrating the trained model into industrial requirement management systems. We plan to conduct more extensive real-world experiments to evaluate its feasibility under various conditions. Additionally, we aim to refine the data labeling and model fine-tuning process by fostering collaborations between industry and academia to accumulate high-quality training data. Furthermore, we will explore methods to improve traceability prediction, particularly

in cases where certain requirement pairs are absent from the training data, ensuring the model's robustness and generalization capabilities.

6. ACKNOWLEDGEMENT

This work is supported by the High Quality Development Special Project(CEIEC-2024-ZM02-0067), National Natural Science Foundation of China(Grant No. 61977020), Key technical projects of ShenZhen (Grant No. JSGG2021110892802003), Basic research project (Grant No. JCKY2021204B025) and Natural Science Foundation of Hei Longjiang Province (Grant No. LH2019F046).

References

[1] J. Tian, L. Zhang, X. Lian, A cross-level requirement trace link update model based on bidirectional encoder representations from transformers, Mathematics 11 (3) (2023) 623.

[2] X. Chen, X. Hu, Y. Huang, H. Jiang, W. Ji, Y. Jiang, Y. Jiang, B. Liu, H. Liu, X. Li, et al., Deep learning-based software engineering: progress, challenges, and opportunities, Science China Information Sciences 68 (1) (2025) 1–88.

[3] H. Gao, H. Kuang, X. Ma, H. Hu, J. Lü, P. Mäder, A. Egyed, Propagating frugal user feedback through closeness of code dependencies to improve ir-based traceability recovery, Empirical Software Engineering 27 (2) (2022) 41.

[4] R. Al-Msie'deen, Requirements traceability: Recovering and visualizing traceability links between requirements and source code of object-oriented software systems, arXiv preprint arXiv:2307.05188 (2023).

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

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

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

[8] J. Lin, Y. Liu, Q. Zeng, M. Jiang, 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.

[9] J. Guo, M. Gibiec, J. Cleland-Huang, Tackling the term-mismatch problem in automated trace retrieval, Empir. Softw. Eng. 22 (3) (2017) 1103–1142. doi:10.1007/S10664-016-9479-8.
URL https://doi.org/10.1007/s10664-016-9479-8

[10] A.-R. Preda, C. Mayr-Dorn, A. Mashkoor, A. Egyed, Supporting high-level to low-level requirements coverage reviewing with large language models, in: Proceedings of the 21st International Conference on Mining Software Repositories, 2024, pp. 242–253.

[11] H. Gao, H. Kuang, W. K. Assunção, C. Mayr-Dorn, G. Rong, H. Zhang, X. Ma, A. Egyed, Triad: Automated traceability recovery based on biterm-enhanced deduction of transitive links among artifacts, in: Proceedings of the IEEE/ACM 46th International Conference on Software Engineering, 2024, pp. 1–13.

[12] J. Zhu, G. Xiao, Z. Zheng, Y. Sui, Enhancing traceability link recovery with unlabeled data, in: 2022 IEEE 33rd International Symposium on Software Reliability Engineering (ISSRE), IEEE, 2022, pp. 446–457.

[13] R. White, J. Krinke, R. Tan, Establishing multilevel test-to-code traceability links, in: Proceedings of the ACM/IEEE 42nd International Conference on Software Engineering, 2020, pp. 861–872.

[14] S. Wang, T. Li, Z. Yang, Exploring semantics of software artifacts to improve requirements traceability recovery: A hybrid approach, in: 26th Asia-Pacific Software Engineering Conference, APSEC 2019, Putrajaya, Malaysia, December 2-5, 2019, IEEE, 2019, pp. 39–46. doi:10.1109/APSEC48747.2019.00015.
URL https://doi.org/10.1109/APSEC48747.2019.00015

[15] A. D. Rodriguez, K. R. Dearstyne, J. Cleland-Huang, Prompts matter: Insights and strategies for prompt engineering in automated software traceability, in: 2023 IEEE 31st International Requirements Engineering Conference Workshops (REW), IEEE, 2023, pp. 455–464.

[16] A. D. Lucia, F. Fasano, R. Oliveto, G. Tortora, Recovering traceability links in software artifact management systems using information retrieval methods, ACM Transactions on Software Engineering and Methodology (TOSEM) 16 (4) (2007) 13–es.

[17] W. Wang, A. Gupta, N. Niu, L. Da Xu, J.-R. C. Cheng, Z. Niu, Automatically tracing dependability requirements via term-based relevance feedback, IEEE Transactions on Industrial Informatics 14 (1) (2016) 342–349.

[18] J. Lin, Y. Liu, Q. Zeng, M. Jiang, 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.

[19] J. H. Hayes, A. Dekhtyar, J. Osborne, Improving requirements tracing via information retrieval, in: Proceedings. 11th IEEE International Requirements Engineering Conference, 2003., IEEE, 2003, pp. 138–147.

[20] J. H. Hayes, A. Dekhtyar, S. K. Sundaram, Advancing candidate link generation for requirements tracing: The study of methods, IEEE Transactions on Software Engineering 32 (1) (2006) 4–19.

[21] G. Capobianco, A. D. Lucia, R. Oliveto, A. Panichella, S. Panichella, Improving ir-based traceability recovery via noun-based indexing of software artifacts, Journal of Software: Evolution and Process 25 (7) (2013) 743–762.

[22] J. Guo, N. Monaikul, C. Plepel, J. Cleland-Huang, Towards an intelligent domain-specific traceability solution, in: Proceedings of the 29th ACM/IEEE international conference on Automated software engineering, 2014, pp. 755–766.

[23] W. Khlif, D. Kchaou, N. Bouassida, A complete traceability methodology between uml diagrams and source code based on enriched use case textual description, Informatica 46 (1) (2022).

[24] B. Wang, R. Peng, Y. Li, H. Lai, Z. Wang, Requirements traceability technologies and technology transfer decision support: A systematic review, Journal of Systems and Software 146 (2018) 59–79.

[25] T.-b. Du, G.-h. Shen, Z.-q. Huang, Y.-s. Yu, D.-x. Wu, Automatic traceability link recovery via active learning, Frontiers of Information Technology & Electronic Engineering 21 (8) (2020) 1217–1225.

[26] J. Lin, A. Poudel, W. Yu, Q. Zeng, M. Jiang, J. Cleland-Huang, Enhancing automated software traceability by transfer learning from open-world data, arXiv preprint arXiv:2207.01084 (2022).

[27] Y. Zhang, C. Ge, S. Hong, R. Tian, C. Dong, J. Liu, Delesmell: Code smell detection based on deep learning and latent semantic analysis, Knowledge-Based Systems 255 (2022) 109737.

[28] R. Mihalcea, P. Tarau, Textrank: Bringing order into text, in: Proceedings of the 2004 conference on empirical methods in natural language processing, 2004, pp. 404–411.

[29] M. Lewis, Y. Liu, N. Goyal, M. Ghazvininejad, A. Mohamed, O. Levy, V. Stoyanov, L. Zettlemoyer, Bart: Denoising sequence-to-sequence pre-training for natural language generation, translation, and comprehension, arXiv preprint arXiv:1910.13461 (2019).

[30] B. Li, Y. Hou, W. Che, Data augmentation approaches in natural language processing: A survey, Ai Open 3 (2022) 71–90.

[31] J. Wei, K. Zou, Eda: Easy data augmentation techniques for boosting performance on text classification tasks, arXiv preprint arXiv:1901.11196 (2019).

[32] Ieee tcsc executive committee, in: Proceedings of the IEEE International Conference on Web Services, ICWS '04, IEEE Computer Society, Washington, DC, USA, 2004, pp. 21–22. doi:http://dx.doi.org/10.1109/ICWS.2004.64.
URL http://dx.doi.org/10.1109/ICWS.2004.64

[33] E. J. Hu, Y. Shen, P. Wallis, Z. Allen-Zhu, Y. Li, S. Wang, L. Wang, W. Chen, Lora: Low-rank adaptation of large language models, arXiv preprint arXiv:2106.09685 (2021).

[34] B. Lester, R. Al-Rfou, N. Constant, The power of scale for parameter-

efficient prompt tuning, arXiv preprint arXiv:2104.08691 (2021).
[35] X. Liu, K. Ji, Y. Fu, W. L. Tam, Z. Du, Z. Yang, J. Tang, P-tuning v2: Prompt tuning can be comparable to fine-tuning universally across scales and tasks, arXiv preprint arXiv:2110.07602 (2021).
[36] C. Ye, J. Yang, Y. Mao, User identification for knowledge graph construction across multiple online social networks, Alexandria Engineering Journal 73 (2023) 145–158.