Defect Correction Method for Software Requirements Text Using Large Language Models

Li Zhang, Liubo Ouyang*, Zhuoqun Xu
College of Computer Science and Electronic Engineering, Hunan University, Changsha, China
{zhangli0208, oylb, zhuoqunxu}@hnu.edu.cn

2024 International Joint Conference on Neural Networks (IJCNN) | 979-8-3503-5931-2/24/$31.00 ©2024 IEEE | DOI: 10.1109/IJCNN60899.2024.10651284

***Abstract*—Text defect correction holds a pivotal role within the domain of text processing. In the face of the rapid advancement of Natural Language Processing (NLP) technologies, significant challenges persist in achieving precision and efficiency, particularly within specialized domains such as software requirements. For instance, traditional methods require labor-intensive verification, and the generative Large Language Models(LLMs) paradigm suffers from illusions and knowledge deficits. So for professional software requirement text and defect correction processing, we construct a Chinese requirement text defect corpus containing 215 Chinese requirement texts and 10,049 manually labeled sentences and propose a contextual prompt pattern based on LLMs, which mainly includes the following steps: (1) Requirements text defect detection classification using fine-tuned BERT. (2) Extracting primary key information from requirement text using fine-tuned UIE. (3) The above results are embedded as contextual information in extensible prompt templates, respectively, and defects are corrected by interacting with generative LLMs. The experimental results show that by utilizing prompt learning and in-context learning techniques for requirement text defect correction, we achieved an outstanding improvement (30.3%) in defect correction effectiveness.**

***Index Terms*—prompt learning, large language models, software requirements**

I. Introduction

In the software development lifecycle, the requirements analysis phase holds integral importance. Software requirements texts describe the desired behavior and characteristics of a software system. They play a decisive role in guiding and constraining the development of the software system. Due to natural language's inherent duality and extensibility, requirements texts often contain deficiencies that can negatively impact software development, leading to anomalies within the software system, deviations from requirements, and missing functionalities. The conventional manual approach to reviewing requirements text requires substantial human resources, consumes significant time, and fails to ensure comprehensive search coverage [1] [2] [3] [4]. Therefore, automatic detection and correction of defects in requirement texts is a crucial challenge.

In recent years, the rapid advancements of Language Large Models(LLMs) have given rise to a tuning paradigm known as "prompt-based" [5] that utilizes LLMs by combining template-based prediction and generation techniques. The goal is to develop and optimize prompts that leverage the capabilities of LLMs across various applications and research domains. The core concept involves integrating the preceding context with the current prompt to understand LLMs' abilities and limitations better, ultimately improving their performance in specialized area tasks.

In recent research, various approaches have been explored in text error correction. Zhang et al. [6] applied soft-masked BERT [7] to address spelling correction, while Hong et al. [8] employed a DAE-decoder paradigm for spelling error detection in Chinese text. Liu et al. [9] introduced word sound and word shape information using a classification-based method to enhance BERT's modeling, yielding improved prediction results. Additionally, Huang et al. [10] incorporated phonological and morphological knowledge for spell checking, and ChineseBERT [11] integrated glyphs and pinyin into BERT inputs via image and CNN sequence vectors. These research efforts have provided valuable insights and practical approaches to advance text error correction techniques. However, when addressing software requirement text, it is essential to go beyond the scope of merely identifying spelling and syntax errors. We hope that the defect correction will consider a combination of factors, including the main content of the text and the type of defective sentence.

In this paper, we propose a contextual prompt pattern based on LLMs. It automatically retrieves defects in the requirement text through fine-tuned BERT while extracting primary key information from requirement text using fine-tuned UIE [12], and then embeds the above results as contextual information in an extensible prompt template and corrects the defects by interacting with generative LLMs. We stimulate the potential of LLMs by introducing a prompt template upstream of LLMs that is relevant to the requirement text task, allowing LLMs to be better adapted to the downstream task at the same time that contextual cues enhance the generative capabilities of LLMs within a specific domain (requirement text defect correction).

II. Related Work

Text Processing. In recent years, there has been a growing interest among researchers in the field of text processing in text classification, text information extraction, and text error correction. In text classification detection, Jani and Islam [13] have pioneered a novel methodology that synergistically amalgamates Case-Based Reasoning (CBR) with neural network paradigms to discern defects and critically evaluate the quality of software requirement specifications. Rosadini et al. [14] were the inaugural researchers to deploy defect detection

*Corresponding author

techniques predicated on Natural Language Processing (NLP) across a voluminous corpus of industrial requirements, meticulously annotated by domain experts. Their empirical findings substantiate the superior efficacy of NLP techniques over conventional manual methodologies traditionally employed in industry for requirements analysis. In addition, Vaish and Sharma [15] proposed a semi-automated approach to detect defects and significantly improve the quality of documentation and development software.

Recent research has shown that joint extraction models outperform traditional pipeline methods regarding text information extraction. Wei et al. [16] introduced CasRel, which utilizes annotation-based techniques to extract entities and relationships simultaneously. Eberts and Ulges [17] employed Transformer and BERT for pre-training and adopted a span-based approach for joint extraction. Furthermore, Wang et al. [18] proposed TPLinker, a fragment-based method that transforms the joint extraction task into a token pair linking problem.

In contrast to traditional textual error correction methods [19] [20], the approach proposed by Rozovskaya and Roth [21] is a very effective method for correcting errors in textual ambiguities. Zhang et al. [22] have unveiled three groundbreaking text correction stratagems—namely Drag-n-Drop, Drag-n-Throw, and Magic Key—that promise to refine the disciplinary process. Additionally, Holtmann et al. [23] have introduced a Controlled Natural Language (CNL) framework for documenting requirements. Based on CNL, automated requirements validation was developed to identify inconsistent and incomplete requirements and perform automated correction operations. These studies represent valuable efforts and practical contributions toward advancing text-processing technology.

Prompt Learning and In-Context Learning. Prompt learning involves leveraging pre-trained LLMs to generate text within a given prompt context. Generative LLMs predict the next potential output word by considering input features and previous inputs. Also, task-specific use of small models as LLM adapters better facilitate adaptation to various downstream tasks [24]. Zhu et al. [25] proposed a short text categorization method with prompt learning by incorporating knowledge graphs and made significant progress on five well-known datasets. Hu et al. [26] have concentrated their scholarly inquiries on integrating exogenous knowledge into the discourse participant, thereby establishing Knowledge-based Prompt Tuning (KPT), which augments and stabilizes the efficacy of prompt tuning.

With the great success of LLMs, In-Context Learning (ICL) [27] has become a hot topic in NLP. The core of ICL lies in the model's ability to quickly adapt and solve new tasks with given textual task instructions and few-shot learning [28] [29], showing great potential and flexibility. Rubin et al. [30] proposed an efficient method for retrieving prompts for ICL using annotated data and an LM while evaluating the effectiveness of this method on three sequence-to-sequence tasks. Sun et al. [31] proposed "SQLPrompt", tailored to improve the few-shot prompting capabilities of Text-to-SQL for LLMs. Besides, Liu et al. [32] demonstrated that the latter has higher accuracy and lower computational cost by rigorously comparing few-shot ICL and Parameter Efficient Fine-Tuning (PEFT) and also proposed a simple recipe based on the T0 model, called T-Few, which can be applied to new tasks without the need for task-specific adjustments or modifications.

III. Method

In this section, we leverage Chinese NLP technology to propose a defect correction method within the context of Chinese software engineering requirement text. The main process is illustrated in the Fig. 1 below:

A. Dataset and Defect Classification Standard

Given the lack of an openly accessible dataset designed explicitly for identifying defects in Chinese requirement texts, we first need to carry out relevant construction and optimization work on the requirement text defects dataset, which provides essential data support for the subsequent training and tasks and ensures the dataset's balance and stability.

The salient features of Chinese requirement texts encompass (1) Brevity and high information density, with each requirement description typically constrained to a maximum of 50 words. (2) Incorporating numerous grammatical symbols, such as delimiters, paragraph markers, and subsection identifiers. (3) The scope of the text content is pretty specialized, predominantly encompassing detailed explications about functional, data, and performance requirements, among others.

Meanwhile, the definitions and examples of fuzzy, unverifiable, and incomplete requirements are shown in TABLE I.

Consequently, combining the above features, we carefully curated a dataset comprising 10,049 samples from 215 Chinese requirement texts spanning diverse fields. Table II presents the categorization of domain categories and detailed characteristics across the 215 requirement texts. The dataset encompasses a range of requirement types, including correct requirements, fuzzy requirements, unverifiable requirements, and incomplete requirements. To facilitate practical model training, validation, and testing, we partitioned the corpus into training, validation, and testing datasets, adhering to a ratio of 7:1:2.

Data Pre-processing. Our initial step involves pre-processing the text data, including (1) Establishing word granularity. In this paper, we adopt words as the granularity unit and employ the Jieba segmentation tool to process the text, integrating specific keywords from the software requirements field to augment segmentation precision. (2) Eliminating irrelevant characters. This involves purging the original text of subsection numbers, extraneous characters, and scrambled codes to maintain the integrity of word order. (3) Constructing a dictionary. We create a dictionary based on the segmentation results, assigning each word a unique ID, and extend the dictionary for new words to ensure the stability of word IDs. (4) Standardizing sample length. We restrict the sample length

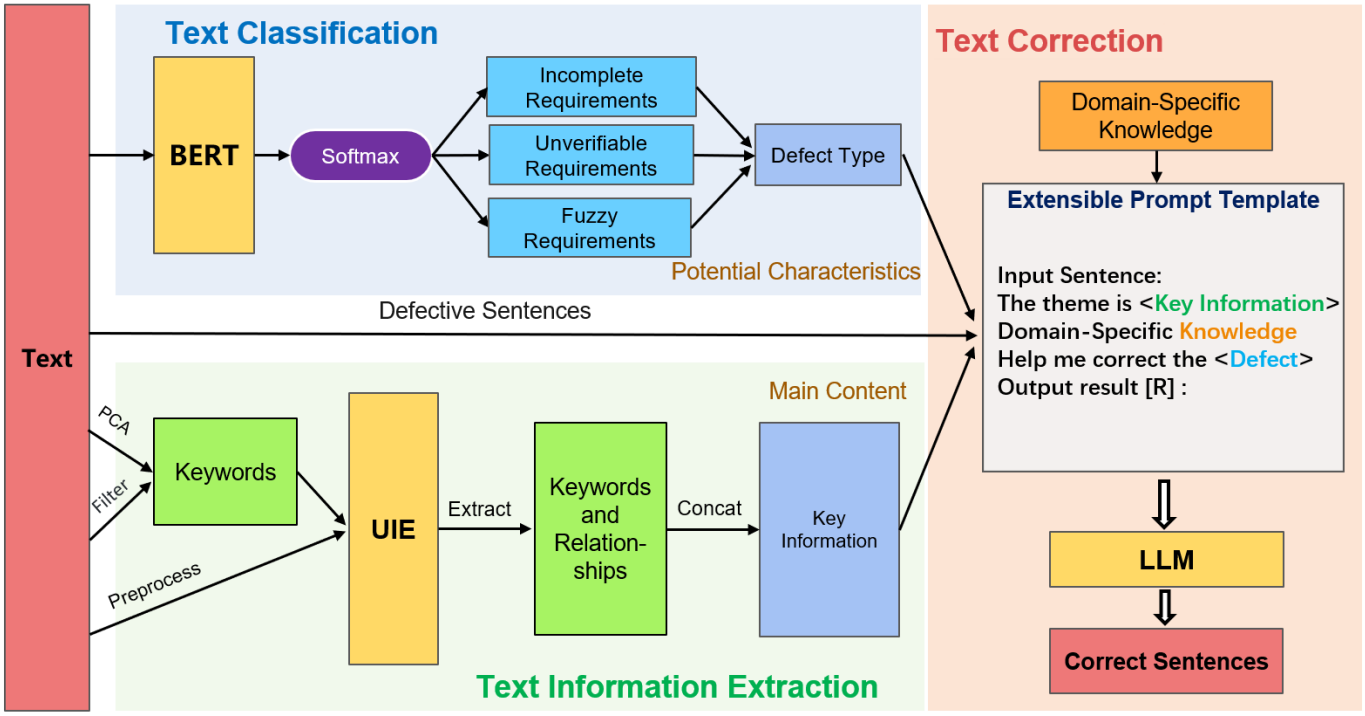

Fig. 1. Illustration of our approach. The left figure shows an overview of our **defect detection** module and **key information** extraction module, and the right figure shows the detailed structure of our **extensible prompt template** (colors represent information sources). Here, all information and the defective sentences are fed into LLM to obtain the correct sentences.

TABLE I
TYPES OF DEFECTS, CLASSIFICATION STANDARDS, AND EXAMPLE SENTENCES (DEFECTS ARE REPRESENTED BY RED FONT)

Defect Types	Classification Standards	Example Sentences (English Version)
Fuzzy Requirements	Requirements that contain domain-specific specialized vocabulary or vague words without clear specialization	"The system supports a variety of operating environments", "The system is robust to cyber attacks"
Unverifiable Requirements	Requirements that are not humanly controllable, cannot be verified, or are extremely costly to verify	"Can open files up to 100 Gigabytes in a millisecond", "The server can categorize 3,000 images in a nanosecond"
Incomplete Requirements	Requirements that are incompletely expressed or have unplanned elements	"The system automatically generates a problem report when it encounters lags, crashes", "Be sure to consider scalability during development"

to 250 words and ensure uniformity in input size through padding or truncation.

Data Augmentation. Given the limited size of the dataset, which could potentially compromise the quality of model training, we resort to two data augmentation strategies: Easy Data Augmentation (EDA) and Back Translation. In the EDA phase, we primarily employ four techniques: Synonym Replacement, Random Insertion, Random Swap, and Random Deletion. The probabilities for Random Insertion and Random Deletion are set at 0.04 to guarantee consistency between the enhanced and original data. In the back-translation phase, we employ the strategy of back-translation and sentence shuffling. Simultaneously, to ensure the quality of the enhanced data, we compare the resultant sentences with the original ones and further filter out sentences that are either excessively short or long or bear excessive similarity or identicality to the original sentences.

B. Text Defect Detection

Defect Checking. The pre-trained LLM already possesses a robust feature extraction capability, which can be further enhanced by incorporating fully connected layers, softmax classification layers, and other components to accomplish the text classification task. The approach enables attaining high-quality results tailored to the specific task requirements. This paper uses a Chinese pre-trained BERT model to construct a

TABLE II

DISTRIBUTION OF CORPUS DOMAIN CATEGORIES AND RELATED INFORMATION. **PER** IS THE PERCENTAGE OF EACH CATEGORY, **LENGTH** IS THE AVERAGE LENGTH OF ALL TEXTS BELONGING TO THE CATEGORY, AND **EXAMPLES** REPRESENTS REAL INSTANCES OF REQUIREMENT TEXTS BELONGING TO THE THEMATIC CATEGORY.

Thematic Categories	Per	Length	Examples
Software Development	16.2%	7006	EasySRS Document Editing System, Intelligent Query And Decision Support System
Health Care	14.0%	7826	Hypertension Management System, Intelligent Re-Examination Reminder System
Science & Education	21.8%	9664	Innovative Course Management System, Teacher-Student Mutual Selection System
E-commerce	14.0%	7632	Second-Hand Trading Platforms For Universities, Convenience Store Data Analysis System
Financial	3.5%	6169	Pocket Accounts Reimbursement System, Financial Inventory Management System
Transportation & Logistics	6.0%	8797	Automobile Repair Store Reservation Service App, Drip Guide Software
Social Media & Entertainment	14.0%	10272	"Xiangyu" Social Media Platform, Municipal Hotline APP
Others	10.5%	9850	Digital Museum Of Chinese Couplets, Sign Language Recognition App

text classification model. Subsequently, we apply the model to detect defects in requirement text, thereby leveraging the strengths of pre-trained LLMs for effective defect detection.

After constructing the dataset, we implemented it based on the Keras framework to build the text classification model. We first guaranteed the length consistency of the input text, and at the same time, we finely configured the model parameters and serialized them through a tokenizer to ensure the standardization of the input sequences. We chose the output vectors corresponding to the [CLS] tokens in the BERT model as the inputs to the Softmax classification layer for text category prediction. In the optimization process of the model, the Adam optimization algorithm was chosen, and the loss function was set to categorical_crossentropy while the learning rate was set to 1e-5 to promote model convergence. After several experimental iterations, by statistically analyzing the training results under different sequence lengths, we determine the optimal tokenize length of 70 and the most suitable batchsize of 16, which provides a reliable parameter basis for the subsequent model training.

C. Text Information Extraction

Key Information Extraction. This module adopts the Jieba library for sentence segmentation and word tokenization of software requirement texts. First, we define the requirement text as T. We use the Jieba.tokenize(T) function to segment it and filter out irrelevant words using the Chinese stopword list *Stopwords* to get the set of sentences $S = \{s_1, s_2, \ldots, s_n\}$.

Next, we build a word frequency dictionary D, where each element corresponds to the frequency of one kind of non-stop word, $D = \{w_1 : f_1, w_2 : f_2, \ldots, w_n : f_n\}$, where D is computed as $D = \text{CountFrequency}(S)$. We convert the elements in D to vector form and weigh them using Principal Component Analysis (PCA) to obtain $D' = \text{PCA}(\text{Vectorize}(D))$. On this basis, we calculate the highly relevant words to the whole text, perform semantic similarity ranking, and filter out invalid words. Finally, we select a representative set of keywords $K = \text{ExtractKeywords}(D')$ that represent the text, where $K = \{k_1, k_2, \ldots, k_n\}$.

We input these selected keywords K together with the requirement text T into the fine-tuned UIE model, and we can obtain other related keywords $\text{related}_k = \text{UIE.template\_entry}(K, T)$. Then, we input the related keywords related_k and the original text T into the fine-tuned UIE model according to the relationship extraction template UIE.template_relation(K, T) defined by the UIE model, to extract the relationship $R = \text{UIE}(\text{related}_k, T)$ between the keywords. Finally, by stitching the set of related keywords related_k and the set of keywords relationships R, we can obtain the key information set KI, $KI = \text{Concat}(\text{related}_k, R)$. In this way, we have transformed the original requirement text T into a more refined and representative key information set KI, which provides the basis for the subsequent analysis.

D. Text Defect Correction

By leveraging data collected from the entire network, LLMs such as GPT, LaMDA, and LLaMA have been trained on a diverse corpus encompassing a wealth of knowledge. We recognize the substantial potential of these models in addressing challenges related to software engineering requirements text. However, to effectively apply LLMs in specific tasks within this domain, they must engage in Man-machine conversation, comprehend the specific problem context, and transfer that understanding to downstream tasks through transfer learning.

To enhance the capabilities of LLMs in requirement text defect correction and rectify defective sentences in requirement texts with greater accuracy and efficiency. Our approach is to pre-build extensible prompt templates and design a library of requirement text defect correction templates so that they can be computed based on defect types and contextual information, and key information and defect sentences can be added and entered into the LLMs (ERNIE Bot, GPT-3.5, GPT-4.0) to get the correct content as expected. As LLMs perform textual defect correction tasks, these templates can serve as adapters to conform LLMs to the domain context required for downstream tasks while playing an essential role in understanding software engineering requirements and related professional knowledge.

The extensible prompt format involves transforming requirement text defects into correct sentences and incorporating relevant contextual clues and keywords into the prompt text. The approach aims to extract the core content of the requirement text efficiently and convey its context effectively. We believe that these context-rich prompts provide valuable information for requirement text defect correction tasks, effectively harnessing the problem-solving capabilities of LLMs in

the domain of requirement texts. Fig. 1 shows the extensible prompt format. We have defined the prompt template-based generation methods:

(1) Main Content: We obtain the key information set KI based on the results in the previous section, and for $kI_j \in KI$, we compute the Term Frequency (TF) and Inverse Document Frequency (IDF) of all the words w_k in kI_j, where the TF-IDF value is calculated as $TF-IDF(w_k, kI_j, KI) = TF(w_k, kI_j) \cdot IDF(w_k, KI)$. We choose the highest scoring phrase as the final result, calculated as KI_max $= \max_{w_k \in kI_j} TF - IDF(w_k, kI_j, KI)$. We believe KI_max effectively summarizes the text's main points and comprehensively represents the entire text.

(2) Potential Characteristics: We perform a down-sampling of the prompt's extension content by incorporating the identified defective sentences and their corresponding defect types from the initial text defect detection step as auxiliary information. In particular, we define the requirement text T, the set of defect sentences $S = \{s1, s2, ..., sn\}$, defect types $D = \{d1, d2, ..., dn\}$, where defect $d_i \in \{Fuzzy, Unverifiable, Incomplete\}$. Each sentence s_i corresponds to a defect type d_i, computed as $S, D =$ BERT.detect(M, T), where $M =$ BERT.train$(T,$ Adam, loss, learning-rate, tokenize, batchsize$)$. The various hyperparameters in M have been described in the previous section. We take the defect sentence s_i and its defect type d_i as potential features that, together with the main content of the requirement text described above, form the input to the LLMs.

(3) Domain-Specific Knowledge: Considering the specialization of the requirement text, we choose to use the relevant specialized domain knowledge in the requirement text as supplementary prompt template information, specialized terms such as system architecture, software updates, streamline with software engineering terms, software requirements specification, etc. We define the set $DSK = \{dsk1, dsk2, ..., dskn\}$ representation and choose the phrase $dski$ that best fits the correction direction to be added to the template.

In summary, throughout the defect correction process, we define the prompt template as PT, the defective sentence set as $DS = \{ds_1, ds_2, ..., ds_n\}$, and the set of optimized sentences as $CS = \{cs_1, cs_2, ..., cs_n\}$. The correct sentences are computed as $CS =$ LLM(PT), where $PT =$ Embed(DS, KI_max, D, DSK), and their values are also computed above. After the rapid summarization of the textual context (KI_max, D) and auxiliary information (DSK), they are embedded in the preset template fed into the LLMs to obtain the optimized sentences.

IV. Experiments

A. Evaluation of Requirement Text Defect Detection

After constructing the BERT model, we conducted relevant training on the dataset. The BERT's performance in terms of accuracy, precision for various types of requirements, recall, and F1 values are shown in Table III. The BERT model exhibits superiority across all indicators, which achieves 95% accuracy in the defect detection module.

B. Key Information Extraction Evaluation

The UIE model leverages actual Chinese requirement text for data sampling and undergoes manual annotation to construct a dataset comprising 2,073 sentences. The dataset primarily encompasses keywords, related keywords, and relationships for information extraction. Subsequently, we divided the dataset into a training dataset and a testing dataset in a 4:1 ratio, utilizing the training set to fine-tune the UIE model. The results of applying our retrained model to the testing dataset have been presented. Our method has achieved 90.5% accuracy in extracting keywords, 88.4% success in extracting related keywords, and 85.2% success in extracting related relationships.

C. Prompt Indicator Evaluation

We carefully selected 254 sentences from the defective requirements text sentences, ensuring a normal distribution of defect types and covering various text titles. These sentences were then corrected by three professionals with a background in software engineering research. The correction process aimed to maintain the original sentences' accuracy, semantics, and structure as much as possible. We utilized this benchmark dataset as an evaluation metric for the prompt, aiming to compare the effectiveness of using various prompt defect correction modules with the direct input of corrections from LLMs. The related information, examples, and comparisons of various prompt templates are shown in Fig. 2. This paper used three LLMs, ERNIE Bot, GPT-3.5, and GPT-4.0, for the experiments. In addition, we employed the following metrics for evaluation:

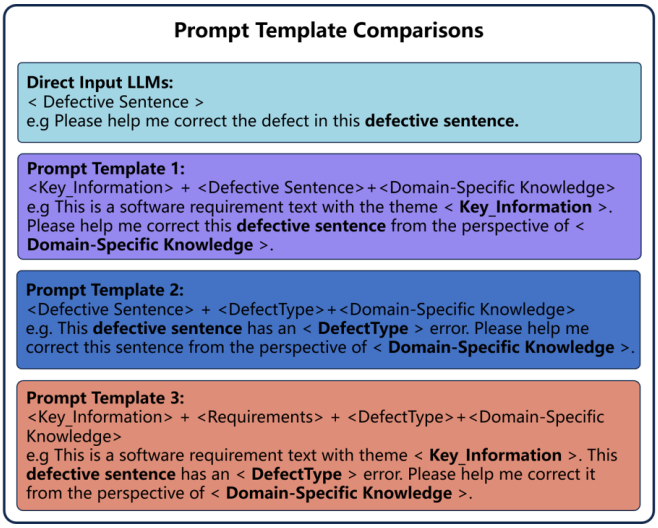

Fig. 2. The related information, examples, and comparisons of various prompt templates. Corresponding from top to bottom are **Direct Processing**, **Prompt Processing(Add Key Information)**, **Prompt Processing(Add DefectType)**, and **Prompt Processing(All)**.

1. To commence, we compared the semantic characteristics among the original text sentences, sentences processed

TABLE III
BERT'S PERFORMANCE IN TERMS OF ACCURACY, PRECISION FOR VARIOUS TYPES OF REQUIREMENTS, MICRO-AVERAGE RECALL RATE, AND F1 VALUES

Model	Accuracy	Precision					Micro-Average Recall Rate	F1 Metrics
		Correct	Fuzzy	Unverifiable	Incomplete	Average		
BERT	0.9636	0.957	0.952	0.998	0.959	0.966	0.963	0.964

TABLE IV
SIMILARITY AND AVERAGE SCORES OF DIFFERENT TYPES OF TEXT IN SEVERAL DIFFERENT LLM ENVIRONMENTS

Text Type	ERNIE Bot		GPT-3.5		GPT-4.0	
	Average Similarity	Average Score	Average Similarity	Average Score	Average Similarity	Average Score
Original	0.941	0.211	0.941	0.211	0.941	0.211
Direct Processing	0.948	0.225	0.949	0.231	0.952	0.239
Prompt (Add Key Information)	0.949	0.228	0.952	0.241	0.951	0.236
Prompt (Add DefectType)	0.951	0.236	0.954	0.254	0.955	0.257
Prompt Processing (All)	0.964	0.293	0.967	0.298	0.972	0.315

TABLE V
AVERAGE ERROR LEVEL SCORES AND AVERAGE SATISFACTION SCORES FOR DIFFERENT TYPES OF TEXT IN SEVERAL DIFFERENT LLM ENVIRONMENTS

LLM	Text Type	Average Error Score	Average Satisfaction Score
Others	Original	7.082	2.792
ERNIE Bot	Direct Processing	6.476	3.358
	Prompt (Add Key Information)	5.520	4.567
	Prompt (Add DefectType)	4.408	5.165
	Prompt (All)	2.764	7.520
GPT-3.5	Direct Processing	6.398	3.587
	Prompt (Add Key Information)	5.544	4.569
	Prompt (Add DefectType)	4.365	5.014
	Prompt (All)	1.857	8.412
GPT-4.0	Direct Processing	6.224	3.655
	Prompt (Add Key Information)	4.708	4.782
	Prompt (Add DefectType)	3.875	5.222
	Prompt (All)	1.452	8.653
Others	Expert Processing	0.715	9.365

directly by LLMs, and sentences processed using various add-information prompt templates with accurately annotated sentences provided by domain experts. For each sentence, we generated a semantic vector expression and calculated the cosine similarity (cos) between the sentence and the correct sentence using "(1)". A higher degree of semantic similarity signifies a closer resemblance to the accurate sentence. Statistical analysis determined the average similarity between text variations and correct sentences in several LLM environments. The experimental results are presented in the first column under each LLM label in Table IV.

$$\cos\theta = \frac{A \cdot B}{\|A\| \cdot \|B\|} \tag{1}$$

2. We compared the manually annotated accurate sentences with the word segmentation of the corrected sentences, filtering them to retain only the contextually relevant keywords. The comparison involved assessing the number of keywords to ensure consistency in the main content and entities of the sentences.

3. We employed cross-entropy, denoted as L(y, p), to evaluate the correctness of the marked correct sentences against the original text, LLMs directly processed sentences, and sentences processed by various add-information prompt templates. The cross-entropy calculation involved "(2)".

$$L(y,p) = -\sum[y \cdot \log(p) + (1-y) \cdot \log(1-p)] \tag{2}$$

Additionally, we calculated the BLEU value between two sentences using the precision_n metric, where precision_n is defined as count_n / total_n. In this equation, count_n represents the number of n-grams that overlap between two sentences, while total_n represents the total number of n-grams in the correct sentences. Ultimately, we computed a

score using "(3)", which reflects the closeness of the corrected sentences to the marked correct sentences.

$$\text{score} = \beta \cdot L(y, p) + (1 - \beta) \cdot \text{precision\_n} \tag{3}$$

A higher score indicates a more excellent proximity. We performed statistical analysis on the average scores of various texts and correct sentences in several different LLM environments, and the experimental results are shown in the second column under each LLM label in Table IV.

4. To assess the performance of our method, we recruited 20 participants who represent potential users of the system. Each participant was given six different sentences to evaluate. The assessment criteria encompassed the original sentence, the associated text title, relevant contextual information, sentences processed directly by the LLMs, and sentences processed using various add-information prompt templates. Participants were instructed to rate the level of errors and their satisfaction with the modifications for each type of text separately. The error score ranged from 0 to 10, with a higher score indicating greater error in the sentence. The satisfaction score ranged from 0 to 10, with a higher score indicating a higher level of satisfaction with the correction, considering factors such as fluency, semantic accuracy, grammar precision, and topic coherence. We conducted statistical analysis to examine the average error levels for different types of texts in several different LLM environments and the average satisfaction score before and after the modifications in several different LLM environments. The final results are presented in Table V.

Based on the comparative analysis results, it was observed that integrating prompt learning and in-context learning technologies for text defect correction significantly improved the average score compared to the original text (39.4%), direct input to the LLMs for processing (30.3%) and text processing using the single add-information prompt template (28.5% & 21.4%) in several different LLM environments. Meanwhile, The integrity of the corrected text and user satisfaction have substantially improved. We are confident that using the method can significantly enhance the efficiency of processing software requirement texts and aid in tasks such as visualization, extraction of crucial content, defect identification, and defect correction.

V. Conclusion

This paper presents a requirement text defect correction method applicable to the software engineering domain. The method accurately identifies the defects in the requirement text while extracting the key information. Subsequently, these results are integrated as contextual information into our extensible prompt templates, and deficiencies in the requirement text are corrected by interacting with the LLMs. The experimental evaluation and performance analysis demonstrate the efficacy of our method in processing requirement texts, significantly enhancing the quality of requirement texts. In the future, we plan to extend the application of prompts to various aspects of requirements engineering. By doing so, our ultimate objective is to achieve effective and high-quality automation processing within the software engineering domain.

References

[1] F. Chantree, B. Nuseibeh, A. D. Roeck, and A. Willis, "Identifying nocuous ambiguities in natural language requirements," in 14th IEEE International Requirements Engineering Conference (RE'06). IEEE, 2006, pp. 59–68.

[2] H M. Jani, "Applying case-based reasoning to software requirements specifications quality analysis system," in the 2nd International Conference on Software Engineering and Data Mining. IEEE, 2010, pp. 140–144.

[3] A. Ferrari and S. Gnesi, "Using collective intelligence to detect pragmatic ambiguities," in 2012 20th IEEE International Requirements Engineering Conference. IEEE, 2012, pp. 191–200.

[4] A. A. Alshazly, A. M. Elfatatry, and M. S. Abougabal, "Detecting defects in software requirements specification," in Alexandria Engineering Journal, 2014, vol. 53, pp. 513–527.

[5] P. Liu, W. Yuan, and J. Fu et al., "Pre-train, prompt, and predict: A systematic survey of prompting methods in natural language processing," in ACM Computing Surveys, 2023, vol. 55, pp. 1–35.

[6] S. Zhang, H. Huang, J. Liu, and H. Li, "Spelling error correction with soft-masked bert," in arXiv preprint arXiv:2005.07421., 2020.

[7] J. Devlin, M. W. Chang, K. Lee, and K. Toutanova, "Bert: Pre-training of deep bidirectional transformers for language understanding," in arXiv preprint arXiv:1810.04805., 2018.

[8] Y. Hong, X. Yu, and N. He et al., "Faspell: A fast, adaptable, simple, powerful Chinese spell checker based on dae-decoder paradigm," in Proceedings of the 5th Workshop on Noisy User-generated Text (W-NUT 2019), Association for Computational Linguistics, 2019, pp. 160–169.

[9] S. Liu, T. Yang, T. Yue, F. Zhang, and D. Wang, "Plome: Pre-training with misspelled knowledge for Chinese spelling correction," in Proceedings of the 59th Annual Meeting of the Association for Computational Linguistics and the 11th International Joint Conference on Natural Language Processing(Volume 1: Long Papers). Association for Computational Linguistics, 2021, pp. 2991–3000.

[10] L. Huang, J. Li, and W. Jiang et al., "Phmospell: Phonological and morphological knowledge guided chinese spelling check," in Proceedings of the 59th Annual Meeting of the Association for Computational Linguistics and the 11th International Joint Conference on Natural Language Processing(Volume 1: Long Papers), Association for Computational Linguistics, 2021, pp. 5958–5967.

[11] Z. Sun, X. Li, and X. Sun et al., "Chinesebert: Chinese pretraining enhanced by glyph and pinyin information," in Proceedings of the 59th Annual Meeting of the Association for Computational Linguistics and the 11th International Joint Conference on Natural Language Processing, Association for Computational Linguistics, 2021.

[12] Y. Lu, Q. Liu, and D. Dai et al., "Unified structure generation for universal information extraction," Association for Computational Linguistics, 2022.

[13] H. M. Jani and A. B. M. T. Islam, "A framework of software requirements quality analysis system using case-based reasoning and neural network," in 2012 6th International Conference on New Trends in Information Science, Service Science and Data Mining, 2012, pp. 152–157.

[14] B. Rosadini, A. Ferrari, and G. Gori et al., "Using nlp to detect requirements defects: An industrial experience in the railway domain," in International Working Conference on Requirements Engineering: Foundation for Software Quality, Springer, 2017, pp. 344–360.

[15] N. Vaish and A. Sharma, "Semi-automated system based defect detection in software requirements specification document," in 2018 5th IEEE Uttar Pradesh Section International Conference on Electrical, Electronics and Computer Engineering, IEEE, 2018.

[16] Z. Wei, J. Su, Y. Wang, and Y. Chang, "A Novel Cascade Binary Tagging Framework for Relational Triple Extraction," in proceedings of the 58th Annual Meeting of the Association for Computational Linguistics, 2020.

[17] M. Eberts and A. Ulges, "Span-based Joint Entity and Relation Extraction with Transformer Pre-training," arXiv preprint arXiv:1909.07755., 2019.

[18] Y. Wang, B. Yu, and Y. Zhang et al., "TPLinker: Single-stage Joint Extraction of Entities and Relations Through Token Pair Linking," in Proceedings of COLING 2020.

[19] C. Plumb, E. C. Butterfield, and D. J. Hacker et al., “Error correction in text: Testing the processing-deficit and knowledge-deficit hypotheses[J],” Reading and Writing, 1994, pp. 347-360.
[20] D. J. Hacker, C. Plumb, and E. C. Butterfield et al., “Text revision: Detection and correction of errors[J],” Journal of Educational Psychology, 1994, 86(1): 65.
[21] A. Rozovskaya and D. Roth, “Generating confusion sets for context-sensitive error correction,” in Proceedings of the 2010 conference on empirical methods in natural language processing, 2010.
[22] M. Zhang, H. Wen, and J. O. Wobbrock, “Type, then correct: Intelligent text correction techniques for mobile text entry using neural networks,” in Proceedings of the 32nd Annual ACM Symposium on User Interface Software and Technology, 2019, pp. 843-855.
[23] J. Holtmann, J. Meyer, and M. Detten, “Automatic validation and correction of formalized, textual requirements,” in 2011 IEEE Fourth International Conference on Software Testing, Verification and Validation Workshops, IEEE, 2011, pp. 486-495.
[24] T. Schick, and H. Schütze, “It’s not just size that matters: Small language models are also few-shot learners,” arXiv preprint arXiv:2009.07118 (2020).
[25] Y. Zhu, Y. Wang, J. Qiang, and X. Wu, “Prompt-learning for short text classification[J],” IEEE Transactions on Knowledge and Data Engineering, 2023.
[26] S. Hu, N. Ding, and H. Wang et al., “Knowledgeable prompt-tuning: Incorporating knowledge into prompt verbalizer for text classification[J],” arXiv preprint arXiv:2108.02035, 2021.
[27] S. Min, X. Lyu, A. Holtzman, M. Artetxe, M. Lewis, H. Hajishirzi, and L. Zettlemoyer, “Rethinking the role of demonstrations: What makes in-context learning work?” ArXiv preprint, abs/2202.12837. 2022.
[28] T. Brown, B. Mann, and N. Ryder et al., “Language models are few-shot learners,” In Advances in Neural Information Processing Systems, volume 33, pp. 1877–1901, Curran Associates, Inc. 2020.
[29] T. Gao, A. Fisch, and D. Chen, “Making pre-trained language models better few-shot learners,” arXiv preprint arXiv:2012.15723 (2020).
[30] O. Rubin, J. Herzig, and J. Berant, “Learning to retrieve prompts for in-context learning[J],” arXiv preprint arXiv:2112.08633, 2021.
[31] R. Sun, S. Ö. Arik, and R. Sinha et al., “SQLPrompt: In-Context Text-to-SQL with Minimal Labeled Data[J],” arXiv preprint arXiv:2311.02883, 2023.
[32] H. Liu, D. Tam, and M. Muqeeth et al., “Few-shot parameter-efficient fine-tuning is better and cheaper than in-context learning[J],” Advances in Neural Information Processing Systems, 2022, 35: 1950-1965.
[33] J. Winkler, J. Grönberg, and A. Vogelsang, “Optimizing for recall in automatic requirements classification: An empirical study,” in 2019 IEEE 27th International Requirements Engineering Conference (RE), 2019, pp. 40–50.
[34] A. Rashwan, O. Ormandjieva, and R. Witte, “Ontology-based classification of non-functional requirements in software specifications: A new corpus and svm-based classifier,” in 2013 IEEE 37th Annual Computer Software and Applications Conference, 2013, pp. 381–386.
[35] J. Fischbach, B. Hauptmann, L. Konwitschny, D. Spies, and A. Vogelsang, “Towards causality extraction from requirements,” in 2020 IEEE 28th International Requirements Engineering Conference (RE), 2020, pp. 388–393.
[36] T. Hey, J. Keim, A. Koziolek, and W. F. Tichy, “Norbert: Transfer learning for requirements classification,” in 2020 IEEE 28th International Requirements Engineering Conference (RE), 2020, pp. 169–179.
[37] F. Dalpiaz, D. Dell’Anna, F. Aydemir, and S. Çevikol, “Requirements classification with interpretable machine learning and dependency parsing,” in 2019 IEEE 27th International Requirements Engineering Conference (RE), 2019, pp. 142–152.
[38] M. Eberts, and A. Ulges, “Span-Based Joint Entity and Relation Extraction with Transformer Pre-Training,” in 2020 European Conference on Artificial Intelligence, IOS Press, 2020.
[39] A. Hassouna, and L. Tahvildari, “An effort prediction framework for software defect correction[J],” Information and Software Technology, 2010, 52(2): 197-209.
[40] C. Arora, M. Sabetzadeh, L. Briand, and F. Zimmer, “Automated Checking of Conformance to Requirements Templates Using Natural Language Processing,” in IEEE Transactions on Software Engineering, vol. 41, no. 10, pp. 944-968, doi: 10.1109/TSE.2015.2428709.
[41] J. Hernández-González, and D. Rodriguez et al., “Learning to classify software defects from crowds: a novel approach[J],” Applied Soft Computing, 2018, 62: 579-591.
[42] Y. Liu, M. Ott, and N. Goyal et al., ”Roberta: A robustly optimized bert pretraining approach.” ArXiv preprint arXiv:1907.11692 (2019).