Automatic Requirements Classification Based on Graph Attention Network

GANG LI¹, CHENGPENG ZHENG², MIN LI¹, AND HAOSEN WANG²
¹Shandong Computer Science Center, Qilu University of Technology (Shandong Academy of Sciences), Jinan 250014, China
²Information Management and Artificial Intelligence, Zhejiang University of Finance and Economics, Hangzhou 310018, China
Corresponding author: Min Li (lim@sdas.org)

ABSTRACT Requirements classification is a significant task for requirements engineering, which is time-consuming and challenging. The traditional requirements classification models usually rely on manual pre-processing and have poor generalization capability. Moreover, these traditional models ignore the sentence structure and syntactic information in requirements. To address these problems, we propose an automatic requirements classification based BERT and graph attention network (GAT), called DBGAT. We construct dependency parse trees and then utilize the GAT for mining the implicit structure feature and syntactic feature of requirements. In addition, we introduce BERT to improve the generalization ability of the model. Experimental results of the PROMISE datasets demonstrate that our proposed DBGAT significantly outperforms existing state-of-the-art methods. Moreover, we investigate the impact of graph construction methods on non-functional requirements classification. DBGAT achieved the best classification results on both seen (F1-scores of up to 91%) and unseen projects (F1-scores of up to 88%), further demonstrating the strong generalization ability.

INDEX TERMS Requirements classification, non-functional requirements, GAT, BERT.

I. INTRODUCTION
The comprehensive and accurate descriptions of functional and non-functional requirements are essential for requirements engineering [1]. Even though the controversy over a clear definition of functional and non-functional requirements, the need for requirements classification remains [2], [3]. With the expansion of the scale and complexity of software systems, software requirements specification (SRS) documents describing system requirements have become very large. Thus, distinguishing the mixed functional and non-functional requirements in an SRS is time-consuming and challenging work [4].

Data with machine learning have achieved high classification accuracy [7], [8]. For example, Kurtanović and Maalej [9] used methods such as semantic similarity, custom dictionaries, and data pre-processing to improve their model’s classification performance. In recent years, convolutional neural networks [10] (CNNs) and recurrent neural networks [11] (RNNs) have shown a strong capacity in the field of requirements classification. Dekhtyar and Fang [12] proposed a CNN classifier joined with Word2Vec [13] embeddings that significantly boosted precision over that of machine-learning methods without sacrificing recall rate. Rahimi et al. [14] developed a two-stage automatic classification system using long short-term memory (LSTM) that is more robust than a single-stage classification system.

Although these studies have achieved great performance in requirements classification, they still have two limitations. First, most of the existing methods based on requirements classification techniques ignore structural features and syntactic information. Traditional requirements classification methods rely heavily on feature engineering. The BoW model treats the text as a collection of several words, and each word appears in the collection independently without establishing a relationship with other words [6]. Word2Vec learns a
A model of semantic knowledge in an unsupervised manner from many text expectations. It integrates the contextual information of words into word vectors to obtain more features [15]. Therefore, it can be found that the traditional feature-extraction techniques only obtain the features of the word itself and the requirements sentences’ shallow information. It is challenging to obtain grammatical and syntactic information. Second, most models have poor generalization ability, especially models that use data preprocessing. Research into requirements classification is ongoing, and many previous models have achieved high performance. However, When applied to unknown projects, model performance drops sharply [16]. Napier’s data show that only 16% of companies use automated techniques for requirements analysis [17], demonstrating that the existing requirements classification model is difficult to apply to real projects.

To address the two limitations mentioned above, we propose an automatic requirements classification method called DBGAT, which is based on a graph attention network and introduces the pre-training model BERT [18] to initialize node embedding to obtain richer feature information. In recent years, the wave of research at the intersection of the graph neural networks (GNNs) [19] and natural language processing (NLP) has affected various NLP tasks, such as knowledge graphs [20], information extraction [21], dialog systems [22], and natural language inference [23]. GNNs are employed to model the graph structure of natural language sentences. For requirements classification tasks, a graph attention network (GAT) [24] can capture sentence structure and grammatical information based on the dependency graph and add it to the node attributes for storage. Then, in the aggregation process, the attention mechanism is used to capture the different attentions of neighbor nodes to the central node. To obtain more feature information, we use BERT for the initial embedding of nodes. BERT is a pre-trained mask language model. Previous studies have demonstrated that BERT can significantly improve classification performance [25], [26], far exceeding the performance of traditional feature-extraction methods. Experiments show that DBGAT can effectively solve the problem of requirements classification and has advantages that other research methods cannot match. The main contributions of this paper are the following.

1) We propose an automatic requirements classification based on graph attention network (DBGAT), which effectively captures the structure information in requirement documents and provides a new research perspective for requirements classification.

2) Experimental results show that the proposed DBGAT achieves great classification results and strong generalization ability in the requirements classification task.

II. RELATED WORK

Functional and non-functional requirements are a well-studied topic in requirements engineering. Functional and non-functional requirements are modeled as goals and soft goals, respectively [27]–[29]. Glinz et al. [3] considered non-functional requirements as to quality constraints and used a quality-oriented approach to model non-functional requirements. Some researchers argue that the distinction between functional and non-functional requirements is artificially defined [30] and that many non-functional requirements also include functional requirements [31]. Automatic extraction and classification of requirements from textual natural language documents have been a popular research topic in requirements engineering. We next present work related to requirements classification from three aspects: traditional machine learning, deep learning, and graph neural networks.

A. TRADITIONAL REQUIREMENTS CLASSIFICATION METHODS

Information retrieval [5] is a general method for classifying requirements in structured and unstructured documents. This method obtains a higher recall value but has poor accuracy when performing multi-classification of NFR subclasses. Ezzati-Karami and Madhavji [6] combined three feature extraction techniques (POS tagging, BoW, and TF-IDF) with a supervised machine-learning algorithm and derived three combined models through experiments. Their accuracy and recall are both greater than 85% based on validation using an industry dataset. In addition, several researchers have enhanced the performance of machine-learning models [7] by pre-processing all context-based users/customers and products in the project that are defined as canonical representations in a specific dictionary [8]. Nevertheless, this approach is not suitable for other unknown projects because the processing method must be re-processed for different styles of projects [25]. Kurtanović and Maalej [9] proposed a supervised machine-learning method that uses auxiliary classifiers, such as metadata, vocabulary, and semantic similarity for classification [32], which in NFR subclass multi-classification tasks (of only four NFR subclasses: usability, security, operability, and performance) achieve performance ranging between 72% and 90%.

B. DEEP LEARNING BASED REQUIREMENTS CLASSIFICATION METHODS

The rapid development of deep learning has produced many language models for NLP in recent years. CNNs [10] and RNNs [11] have been increasingly applied in requirements classification tasks. Navarro-Almanza et al. [33] used a CNN to classify the 12 NFR subclasses in the PROMISE dataset, with an F1-score of 77%. Dekhtyar and Fang [12] utilized TensorFlow-guided learning and a CNN for NFR classification and introduced Word2Vec embeddings [13] to delegate every word in the requirements, which provides an accurate and measurable improvement in classification accuracy. Baker et al. [34] proposed a CNN model that
can effectively classify NFRs into five different subclasses while achieving F1-scores ranging between 82% and 92%. In addition, they invited five security experts to classify the requirements of the security field, but the effect of manual classification was lower than the automatic requirements classification model. Rahimi et al. [14] used a combination of three holistic approaches and four deep learning models (LSTM, BiLSTM, GRU, and CNN) to build an NFR classifier and develop a two-stage classification system. Hey et al. [25] proposed NoRBERT based on the pre-trained model BERT, which achieved excellent results. In addition, Kici et al. [26] proposed distillation BERT, which is superior to LSTM and BiLSTM in requirements classification tasks, through experimental comparison.

### C. GRAPH NEURAL NETWORK

In order to capture the relationship between text or words, various graph-based approaches have been developed for text classification [15], [35]. Graph-structure data can code complex semantic relationships in sentences and learn more feature information to improve the classification effect. Many researchers [36], [37] have presented sophisticated neural network models to process arbitrarily structured graphs by incorporating regular grid structures in CNNs. Kipf and Welling [36] proposed a scalable semi-supervised learning method called graph convolutional network (GCN), which can run neural networks directly on graphs and achieve advanced classification results in several tasks. Some authors explore different composition approaches combined with graph neural networks for text classification. They construct graphs by dependencies between sentences and documents [38], [39] or for relationships between words in sentences. Yao et al. [40] proposed a GCN-based text classification algorithm that constructs independent text graphs for the corpus through word co-occurrence and document-word relations, which method outperformed the then state-of-the-art algorithms on several text classification tasks without external word embeddings or knowledge. Recently, the wave of research at the intersection of GNNs and NLP has affected various NLP tasks, e.g., knowledge graphs [20], information extraction [21], dialog systems [22], and natural language inference [23].

### III. METHODOLOGY

The DBGAT structure and the components are depicted in Figure 1. Firstly, dependency parse trees are used to build graph structure of requirements sentences. Then BERT initializes the nodes/edges in the graph, and GAT captures more node features through training. Finally, use the classifier for requirements classification.

#### A. DEPENDENCY GRAPH CONSTRUCTION

The dependency graph is widely used to capture the dependency relations between different objects in the given sentences. Formally, given a paragraph, one can obtain the dependency parsing tree (e.g., syntactic dependency tree or semantic dependency parsing tree) by using various NLP parsing tools. Then one may extract the dependency relations from the dependency parsing tree and convert them into a dependency graph. We assume that the input is a corpus composed of m sentences, and a sentence of length n is indicated as \( \text{sent}_i = \{s_1, s_2, \ldots, s_n\} \). To formalize this problem mathematically, we give the mathematical symbol \( G(V, E) \), of the graph, where \( V \) is the set of nodes denoted \{\( w_1, w_2, \ldots, w_n \)\} and \( E \) is the set of edges denoted \( \{(w_i, w_j) : i, j \in n\} \).

First, we build a syntactic parse tree that reveals the syntactic structure of requirements, as shown in Figure 2. The first relationship line in the syntactic parse tree points from the word “administrators” to the word “Only,” and the “advmod” line in the middle of the two words indicates that the syntactic information of these two words is an adverbial modification relationship. The rich syntactic information in the sentence can be used to capture more characteristics of the requirements sentence. Next, we use the features captured by the syntax parse tree to construct the dependency graph.

We set the words in the syntactic parse tree as the nodes of the dependency graph, and the grammatical relations between the words are set as the edges of the dependency graph. Each dependency graph contains the number of nodes,
node-information set, and edge-information set. The node information set is represented by \((w_1, w_2, \ldots, w_k)\). Each piece of node information is a dictionary including node value (Word token), node position ID (the Word’s position ID in the original sentence), and node ID (the node token’s ID that will be used in Graph).

The construction of the dependency graph is mainly done to save the syntactic features between words in the syntactic parse tree as the edge information of the dependency graph. We save the relationship between two nodes using the node ID if there is a connection between node \(w_i\) and node \(w_j\). For example, \((w_5, w_6)\) indicates a connection between node \(w_5\) and node \(w_6\). This method can simplify the post-order calculation and save the syntactic structure relationship between each word. It can capture more feature information of requirements sentences and calculate faster.

**B. INITIALIZE NODE EMBEDDING WITH BERT**

The generalization ability of the requirements classification model is essential and determines whether the model can maintain stable classification accuracy in practical applications. Since the training data available in non-functional requirements fields are very limited, it is necessary to use limited data to capture more features. BERT is proposed by Google, which adds bidirectional encoding representation to Transformer. The bidirectional encoding representation enables the model to process a word vector while considering the meanings of the words before and after the word to achieve contextual semantic understanding, which improves the model’s performance in the field of complex semantic understanding. Thus BERT has a natural advantage in improving generalization capabilities [41]. So we use BERT to initialize the nodes of the dependency graph so that the dependency graph can get richer feature information.

To use the BERT model, we reconstruct the requirements sentence into a form suitable for BERT input, i.e., we add “[CLS]” and “[SEP]” at the beginning and end of the requirements sentence, respectively. The adjusted input sequence is

\[
x = \{x_0, x_1, \ldots, x_n, x_{n+1}\}
\]

where \(x_0\) and \(x_{n+1}\) are vector representations of “[CLS]” and “[SEP]”, respectively. \(\{x_1, x_2, \ldots, x_n\}\) is a vector representation of the word sequence \((w_1, w_2, \ldots, w_k)\). We generate sequence \(t\) after \(x\) is initialized by BERT, and its length is the same as sequence \(x\), which is expressed as follows:

\[
t = \{t_0, t_1, \ldots, t_n, t_{n+1}\}
\]

where \(t_0\) is called a “BERT pooling” vector and \(\{t_1, t_2, \ldots, t_n\}\) are the output context vector representations of the input sequence. In this work, we use \(\{t_1, t_2, \ldots, t_n\}\) for pooling and feature fusion.

**C. DEPENDENCY GRAPH EMBEDDING WITH GAT**

After dependency-graph construction and node-embedding initialization, we use the GAT model [42] to learn the network embedding of requirements statements. The specific framework is shown in Figure 3. Based on the initialization of node embedding by the BERT pre-training model, bidirectional node embedding is obtained by aggregating the information from the forward and backward neighborhoods of nodes in the graph. Then, graph embedding is constructed based on the learned node embedding to capture the information of the entire graph.

We use a bidirectional node-embedding algorithm to induce node embedding, which generates bidirectional node embedding by aggregating the local forward and backward neighborhood information of nodes in a K-hop graph. The embedding generation process of nodes proceeds as follows.

1) In the previous step, we used BERT to initialize the embedding and converted the text attributes of the node \(w\) into the feature vector. Then, according to the edge
of the Softmax layer and then apply the Softmax layer for classification; the input is converted into a confidence space and then use the Softmax function to output the probability distribution of the requirements candidate subclasses. The formula for running the forward aggregation is

$$h_k^w = \max \left( \left\{ \sigma \left( W_{pool} h_{l_{\text{in}}}^k + b \right), u \in N_k^w \right\} \right)$$  (3)

where $\max$ denotes the element-wise max operator, $\sigma$ is a non-linear activation function, and $W_{pool}$ is a pooling parameter matrix.

3) We connect the forward representation of $w$, $h_{N_{k}^{w}}^{k-1}$, with the newly generated neighborhood vector, $h_{N_{k}^{w}}^k$, The connection vector is input into the fully connected layer, and the forward representation of the node is updated for use in the next iteration.

4) We update the backward operations, using a similar process to the forward operations in steps 2) and 3). The aggregate function formula used by backward aggregation is

$$h_k^w = \max \left( \left\{ \sigma \left( W_{pool} h_{l_{\text{in}}}^k + b \right), u \in N_k^w \right\} \right)$$  (4)

5) We repeat steps 2)-4) $K$ times and integrate the final forward and reverse representations as the final bidirectional representation of $V$ to complete the node embedding operation.

6) Finally, we perform the graph embedding operation by the node-based method. We add a new core node $w_s$ to the input graph and add all the other nodes to $w_s$ directly. Then, we generate the embedding of $w_s$, which is also the graph embedding, by aggregating the embeddings of the neighboring nodes.

**D. REQUIREMENTS CLASSIFICATION WITH MLP**

For final requirements classification, we use the maximum pooling method to aggregate the node vector output of the GAT model, input them into the multilayer perceptron (MLP) and then use the Softmax function to output the probability distribution of the requirements candidate subclasses.

We use the MLP model to calculate the probability. $r$ represents a fixed distribution, $W_1 \in R^{H \times M}$ a fully connected weight matrix, $H$ the number of hidden layers, $b_1$ a bias term, and $O \in R^{H \times 1}$ a fully connected layer output. The calculation formula is

$$O = f \left( W_1 r + b_1 \right)$$  (5)

We convert the original feature space into a confidence space and then apply the Softmax layer for classification; the input of the Softmax layer $I \in R^{C \times 1}$ are representations as follows:

$$I = W_2 O$$  (6)

| Class | Project Number | Total |
|-------|----------------|-------|
|       | 1   | 2   | 3   | 4   | 5   | 6   | 7   | 8   | 9   | 10  | 11  | 12  | 13  | 14  | 15  |
| Availability(A) | 1  | 2   | 2   | 0   | 2   | 1   | 0   | 5   | 1   | 1   | 2   | 1   | 1   | 1   | 1   |
| Legal(L) | 0   | 0  | 0  | 6  | 3  | 0  | 1   | 1   | 5   | 0   | 0  | 0  | 0  | 0  | 0  | 0  |
| Look and Feel(LF) | 1  | 4   | 0   | 2   | 3  | 3  | 2   | 0   | 6   | 0   | 7   | 2   | 2   | 4   | 3   | 2   |
| Maintainability(MN) | 0   | 0   | 0   | 0   | 0  | 4   | 0   | 2   | 1   | 1   | 3   | 2   | 2   | 2   | 2   | 1   |
| Operational(O) | 0   | 0   | 7   | 6   | 10  | 15  | 3   | 9   | 2   | 0   | 2   | 2   | 3   | 3   | 6   |
| Performance(PE) | 2   | 6   | 2   | 2   | 4   | 1   | 2   | 1   | 7   | 4   | 4   | 3   | 5   | 0   | 1   | 1   |
| Scalability(PO) | 0   | 3   | 4   | 0   | 3   | 4   | 0   | 4   | 0   | 0   | 0   | 1   | 2   | 0   | 0   | 0   |
| Security(SI) | 1   | 3   | 10  | 7   | 3   | 5   | 2   | 15  | 0   | 1   | 3   | 3   | 2   | 2   | 6   |
| Usability(US) | 3   | 6   | 8   | 4   | 5   | 13  | 0   | 10  | 0   | 2   | 2   | 3   | 6   | 6   | 1   | 6   |
| Feud/Tolerance(FT) | 0   | 4   | 0   | 0   | 0   | 2   | 0   | 2   | 0   | 0   | 0   | 2   | 0   | 0   | 0   | 0   |
| Portability(PO) | 0   | 1   | 0   | 0   | 0   | 0   | 0   | 0   | 0   | 0   | 0   | 0   | 0   | 0   | 0   | 0   |

Total NFRs: 625
Total Functional: 20
Total Security: 20
Total Operability: 30
Total Performance: 35
Total: 150

where $W_2 \in R^{C \times H}$ represents the conversion matrix and $C$ the number of classes. If $C$ is to classify the four subclasses of NFR, then $C = 4$. The confidence of each class sample is denoted $I$. The Softmax layer normalizes the confidence value.

**IV. EXPERIMENTS**

The dataset, experiment tasks, experiment setting, and evaluation metrics used in the experiment are described, followed by the main contrast methods. Finally, we compare our proposed DBGAT with other methods using three experimental tasks and interpret the results.

**A. DATASET**

1) PROMISE DATASET

To ensure that the experiments are comparable and persuasive, we used the PROMISE dataset, which is usually used in the field of non-functional requirements. Table 1 shows the main components of this dataset. PROMISE has 625 statements of requirements, including 255 FRs and 370 NFRs. The table shows an uneven distribution of categories, with 115 more NFRs than FRs. Four NFR subclasses have more than 50 (usability, security, operability, and performance). In addition, the dataset consists of 15 different items, providing a basis for testing generalization capabilities. Since the dataset includes the need for duplicates and mislabeling, Dalpiaz et al. [16] provided a re-tagged version of the dataset, consisting of only 612 of the original 625 requirements. We supplemented the deleted 13 data with replacements through data augmentation.

2) CONCORDIA RE DATASET

The Concordia RE corpus [43] was created by Rashwan et al. for their work [44]. It contains six types of requirement documents with a total of 3064 annotated sentences. In order to test the generalization ability of the

1 https://doi.org/10.5281/zenodo.268542
model in our experiments, we extracted four non-functional requirements, including usability, security, operability, and performance, from the Concordia RE corpus. Then, we re-labeled some of the data from the Concordia RE corpus. Table 2 shows the types and numbers of non-functional requirements extracted from the Concordia RE corpus.

### B. EXPERIMENTAL DESIGN

#### 1) EXPERIMENTAL TASKS

We established the following three tasks to evaluate the performance of the model on the PROMISE public dataset.

- **Task 1**: Perform FR/NFR classification on PROMISE.
- **Task 2**: Multi-class classification of the four most frequently occurring NFR subclasses in the PROMISE dataset, combined with the heterogeneous dataset Concordia RE for testing the generalization capability of the model.
- **Task 3**: Select all subclasses of NFR on the PROMISE dataset for multi-class classification.

#### 2) EVALUATION METHOD AND METRICS

We used two evaluation methods for the focus of the tasks. With **10-fold**, we describe tenfold cross-validation to check the classification ability of the model. This method divides the dataset into 10 equal parts, uses each one as the testing set and the others as the training set, and is then executed 10 times before the results are averaged. To further investigate the model generalization ability, we used the project-level cross-validation called **p-fold** proposed by Dalpiaz et al. [16], splitting the dataset with three items selected as testing sets and the remaining items as training sets.

For all tasks, we measure Precision (P), Recall (R), F1-score (F1), and Average F1-score (A), which are defined below:

\[
\begin{align*}
\text{Recall} &= \frac{TP}{TP + FN} \\
\text{Precision} &= \frac{TP}{TP + FP} \\
\text{F1} &= \frac{2 \times \text{Precision} \times \text{Recall}}{\text{Precision} + \text{Recall}} \\
\text{A} &= \frac{\sum_{i=1}^{m} F1_i}{m}
\end{align*}
\]

where, TP denotes True Positive (No. of correct positive classification), FP denotes False Positive (No. of incorrectly classified sentences), and FN represents False Negative (No. of sentences incorrectly not classified). In addition, in order to compare the classification performance of the model, we weighted average F1-score values of all categories in the classification task to get the average F1-score (A).

#### 3) OTHER GRAPH-CONSTRUCTION METHODS

Text data modeling methods such as the bag-of-words model and the word embedding model can only capture the shallow information of the word itself. They cannot do anything for deeper sentence structure information and semantic information. In order to capture more characteristic information of the requirement sentence, we express the requirements as a graph structure to enhance the representation of the sentence [15]. How to construct a text sequence into a graph structure is a difficult step. The following are the other graph construction methods we compared in the experiment of this paper:

- **(1) Constituency Graph**: This is one of the more popular graph-construction methods that capture phrase-based syntactic relationships in sentences. It follows the relationships of phrase structure syntax rather than dependencies and dependent syntax [45].
- **(2) Information-Extraction Graph**: This method obtains high-level information by extracting structural information between sentences. In many NLP tasks, these extracted relationships prove to be helpful [46].
- **(3) Dynamic Graph**: Dynamic learning performs collaborative learning on the graph-construction module and subsequent graph-representation modules end-to-end, which can better optimize downstream tasks [47].

### C. EXPERIMENTAL DESIGN

#### 1) DBGAT MODEL EXPERIMENT SETUP

The DBGAT model was implemented using the deep learning framework Pytorch and trained on a single GeForce RTX 3080 GPU. We use the Stanford NLP toolkit\(^2\) for the demand statements in the graph construction session. And we employ “bert-base-uncased” model to initialize the embedding of graph nodes, then the graph attention network for graph embedding,\(^3\) and finally input to the MLP classifier for classification. Since the hyperparameters of a deep learning model strongly impact its performance, we investigated different combinations of model settings for the DBGAT model. In Table 3, we set optional ranges for parameters such as word embedding strategy and graph attention network hyperparameters. The optimal parameters are determined through experiments. The optimal parameter

---

\(^2\)https://stanfordnlp.github.io/CoreNLP/

\(^3\)https://github.com/IBM/Text-to-LogicForm
TABLE 4. Comparison of F/NFR classification models. Underlining indicates the optimal value for that group in that column. The data in bold indicate the optimal value in the column. “-” indicates that data is missing.

| Group                  | Approach/Parameter          | F(255) | NFR(370) |       |
|------------------------|----------------------------|--------|----------|-------|
|                        |                            | P  | R  | F1 | P  | R  | F1 | A  |
| **Machine Learning**   |                            |    |    |    |    |    |    |    |
| DecisionTree           | .70 .75 .72                | .80 | .78 | .79 | .76 |
| RandomForest           | .82 .81 .81                | .83 | .88 | .85 | .83 |
| NaïveBayes             | .80 .93 .86                | .95 | .85 | .90 | .88 |
| SVC                    | .76 .80 .78                | .87 | .84 | .85 | .82 |
| **Pre-processing and NoRBERT** | K & M (word features) | .92 | .93 | .93 | .92 | .93 | .93 |
|                        | A. et al. (processed data) | .90 | .97 | .93 | .90 | .98 | .95 | .94 |
| D. & F (word2vec,50)   | -                          | -    | .93 | .92 | .92 | -    | -    |
| NoRBERT(large,OS)      | .92 .88 .90                | .92 | .95 | .93 | .92 |
| **GNN**                |                            |      |      |    |    |    |    |    |
| Dependency+GAT         | .91 .91 .91                | .92 | .91 | .91 | .91 |
| Constituency+GAT       | .87 .90 .88                | .91 | .88 | .89 | .89 |
| IE+GAT                 | .80 .89 .84                | .90 | .84 | .87 | .86 |
| Dynamic+GAT            | .91 .86 .88                | .91 | .93 | .92 | .90 |
| **GNN+BERT**           |                            |      |      |    |    |    |    |    |
| DBGAT_{Const}          | .91 .91 .91                | .92 | .94 | .93 | .92 |
| DBGAT_{IE}             | .93 .86 .89                | .89 | .92 | .90 | .90 |
| DBGAT_{Dy}             | .92 .90 .91                | .94 | .92 | .93 | .92 |
| DBGAT                  | .94 .90 .92                | .94 | .98 | .96 | .94 |

settings of the DBGAT model for the three experimental tasks in the above experimental design are shown in Table 3. Since there are many combinations of experimental parameters for the DBGAT model, we only partially show the results in the subsequent corresponding practical tasks according to the actual model effects.

D. EXPERIMENTAL RESULTS AND ANALYSIS

1) TASK 1: FR/NFR CLASSIFICATION

Task 1 is the F/NFR two-category task. We used the PROMISE dataset and folded all the NFR subclasses to represent the NFR. We divided the model into four groups according to its classification method and used the stratified 10-fold cross-validation setting to evaluate the capabilities of the classification model. The specific classification Precision, Recall, and F1-score are shown in Table 4. The last column indicates the weighted average F1-score(A).

a: MACHINE LEARNING METHODS

We used Machine Learning (ML) methods [6] in the Sklearn library to classify the requirements and used TF-IDF to represent the text features. As can be seen in Table 4, we found that Naïve Bayes achieved the best classification in the ML group with an F1-score of 86% for functional requirements and 90% for non-functional requirements. The weighted average F1-score of the Random Forest is 83%, which is 7% higher than that of the decision tree.

b: PRE-PROCESSING METHODS AND NoRBERT

Abad et al. reached the highest weighted average F1-score of 94% by pre-processing the dataset with hand-supplied dictionaries and rules [8]. Kurtanović and Maalej reported higher F1-scores using models that used only word features without feature selection [9]. These two models, which rely on data pre-processing, fitted too well with this dataset and are not suitable for other unknown projects. NoRBERT [25] achieved comparable performance with an F1-score of 90% for functional requirements and 93% for non-functional requirements. The weighted average F1-score values of the models in this group were all higher than 91%.

c: GNN METHODS

We used four graph-construction methods combined with a GNN to execute Task 1. The precision, recall rate, and F1-score of the dependency-graph-construction method were higher than 90% for functional and non-functional requirements classification. Both dynamic and constituency graphs also maintained a high weighted average F1-score of 90% and 89%. However, the classification of the IE graph is poor, even lower than the Naïve Bayes algorithm in ML.

d: GNN+BERT

We added BERT as a node-embedding-initialization method based on the previous group. And compare the different graph construction strategies with the following three approaches: (1) Constituency Graph (DBGAT_{Const}), (2) Information Extraction (DBGAT_{IE}), and (3) Dynamic Graph (DBGAT_{Dy}). The DBGAT model achieved a weighted average F1 score of 94%, similar to that of the Ada et al. model, which uses manual pre-processing. One of the reasons for the good results of DBGAT classification is that our graph encoder jointly extracts these features in a unified model by propagating dependencies to the sentence structure graph, capturing more features to make the DBGAT model achieve the optimal classification results. The weighted average F1-score of both DBGAT_{Const} and DBGAT_{Dy} reached 92%. The Constituency Graph can capture phrase-based syntactic relations in one or more sentences and demonstrates a better classification effect. DBGAT_{IE} is the lowest in this group due to its unsuitability for short sentence classification, with a weighted average F1-score of only 90%.

Through the comparison of the third and fourth groups, we found that the weighted average F1-score of GNN models increased by 3%–4%. BERT can learn the nuances and features of each word which helps in word embedding, and the GNN model gets better word vectors in the word embedding stage. Adding BERT to a DBGAT can significantly improve the classification ability of the model. For the GNN classification method, we found that the graph-construction method itself has a strong classification ability, except for the IE graph. The IE graph had the worst
TABLE 5. The four NFRs with the largest number of samples in the PROMISE dataset were classified by availability, security, operability, and performance. Const and Dy indicate the Constituency and Dynamic Graph.

| Evaluate Method          | US       | SE       | O        | PE       | 10-fold | P-fold |
|--------------------------|----------|----------|----------|----------|---------|--------|
|                          | P | R  | F1 | P | R  | F1 | P | R  | F1 | P | R  | F1 | P | R  | F1 |
| DecisionTree             | 35 | 79 | 49 | 58 | 56 | 57 | 88 | 47 | 61 | 82 | 46 | 59 | 56 | 35 | 20 | 22 |
| RandomForest             | 58 | 69 | 63 | 78 | 76 | 77 | 71 | 71 | 71 | 92 | 73 | 81 | 73 | 75 | 62 | 68 |
| NaiveBayes               | 61 | 69 | 65 | 86 | 86 | 86 | 89 | 79 | 74 | 95 | 78 | 86 | 77 | 88 | 39 | 54 |
| SVC                      | 62 | 71 | 66 | 95 | 67 | 79 | 97 | 80 | 67 | 77 | 67 | 72 | 71 | 83 | 52 | 40 |
| K & M.(bin)              | 81 | 85 | 82 | 91 | 90 | 88 | 72 | 75 | 73 | 93 | 90 | 90 | 83 | -  | -  | -  |
| K & M.(multi)            | 65 | 82 | 70 | 81 | 77 | 75 | 81 | 86 | 82 | 86 | 81 | 80 | 76 | -  | -  | -  |
| NoRBERT(ep32)            | 78 | 85 | 81 | 78 | 92 | 84 | 83 | 84 | 83 | 94 | 87 | 90 | 85 | -  | -  | -  |
| NoRBERT(ep 50)           | 83 | 88 | 85 | 90 | 92 | 91 | 78 | 84 | 81 | 92 | 87 | 89 | 87 | 71 | 82 | 76 |
| Dependency+GAT           | 92 | 79 | 85 | 85 | 96 | 90 | 91 | 85 | 88 | 74 | 90 | 81 | 86 | 92 | 84 | 88 |
| Constituency+GAT         | 88 | 84 | 86 | 78 | 79 | 78 | 88 | 83 | 85 | 74 | 89 | 81 | 83 | 95 | 70 | 81 |
| Dynamic+GAT             | 90 | 80 | 85 | 77 | 80 | 78 | 87 | 91 | 89 | 83 | 92 | 87 | 85 | 80 | 89 | 84 |
| DBGAT\textsuperscript{Constr} | 96 | 85 | 90 | 92 | 91 | 91 | 92 | 89 | 90 | 89 | 96 | 92 | 91 | 71 | 83 | 77 |
| DBGAT\textsuperscript{Dyn}    | 97 | 90 | 93 | 90 | 87 | 88 | 93 | 84 | 88 | 87 | 95 | 91 | 90 | 89 | 63 | 74 |
| DBGAT                     | 93 | 92 | 92 | 94 | 87 | 90 | 91 | 87 | 89 | 91 | 97 | 94 | 91 | 94 | 72 | 82 |

Comparing the three different methods of constructing graphs, it can be found that 86% of the classification effect can be achieved using only the dependency graph and GAT while maintaining a solid generalization ability. This demonstrates that the graph construction approach still maintains the advantages of feature learning in non-functional demand multi-classification tasks. The weighted average F1-score rate of DBGAT reached 91%, which is 4% higher than that of the NoRBERT model. The weighted average F1-score of the DBGAT\textsuperscript{Const} and the DBGAT\textsuperscript{Dyn} graph models reached 91% and 90%, respectively. Using graph structure to represent the need to display more textual information combined with graph attention networks for feature learning yields the best results in multi-classification tasks with non-functional requirements.

We compared the weighted average F1-score results obtained by 10-fold and p-fold of the model in Task 2. Figure 4 ranks the generalization capabilities of the model by GA value. It can be seen from Figure 4 that the dependency and dynamic graphs have strong GA without adding the BERT pre-training model. Among them, the GA of the dynamic graph is the strongest, and that of the dependency graph is the second-strongest. However, although these two methods have strong GA, they do not achieve the best classification performance. After adding BERT, the weighted average F1-score of these two models increased by more than 5%. BERT plays an essential role in improving the performance of the GNN model of NFR multi-classification.

We added a new test to better show the generalization ability of the model. We use the four most numerous classes in the PROMISE dataset as the training set and the data extracted from the Concordia RE corpus as the test set. Here we use four machine learning algorithms to compare with the DBGAT model, and the experimental results are shown in Table 7.

As the experimental results presented in Table 7, as a whole, four traditional machine learning methods have poor

effect in the third and fourth groups, which may be because the IE graph is good at capturing the relationship between distant sentences [15] but the PROMISE dataset is mostly single sentences. And so, we eliminated the IE graph in the subsequent experimental task.

2) TASK 2: FOUR NFR SUBCLASSES CLASSIFICATION

Usability, safety, operability, and performance are the four frequent NFRs in the PROMISE. Therefore, we used these four NFR subclasses in Task 2 as a relatively balanced sample test classification model. First, we used a 10-fold cross-validation method to test the classification ability of the model. Then, to test the generalization performance of the models, we used the p-fold method. Table 5 shows the classification results obtained by the model using two test methods. The last column of Table 5 shows the change in the classification ability of the model on the unknown project, which we call generalization ability (GA), the calculation method of which used the 10-fold weighted average F1-score subtracting the p-fold weighted average F1-score.

The experimental results are shown in Table 5. First, the weighted average F1-score of the 10-fold experimental results of the four ML algorithms are all lower than 80%. The worst generalization ability of the SVC model reduces the weighted average F1-score by 19%. Only the Random Forest model has a certain generalization ability among the four ML methods, but the classification weighted average F1-score is low. The unimproved ML method is not competent for the multi-classification task of NFR. Kurtanović-Maleej [9] proposed model reached 83% weighted average F1-score with manual data pre-processing (due to the lack of code for the model, the model cannot be p-fold-tested). The BERT-based NoRBERT [25] model achieved 87% weighted average F1-score and has strong generalization capabilities. We can see that BERT can improve the classification ability of the model on unknown demand classification items.
3. TASK 3: ALL NFR SUBCLASS CLASSIFICATIONS

In Task 3, we performed a multi-class classification of the PROMISE dataset for all NFR subclasses to check the DBGAT performance. We filtered out the portability of only one piece of data because it cannot be in the training and testing sets simultaneously, so it failed to predict. The experimental results obtained by 10-fold are shown in Table 6.

From Table 6, we can find that four machine learning models failed on this task, and their weighted average F1-score for the NFR multi-classification task did not exceed 60%. The NoRBERT model proposed by Hey et al. exhibited strong classification performance with a weighted average F1-score of 82%. DBGAT$_{Dy}$ also achieved better generalization ability, and the classification effect decreases very significantly. The DBGAT model achieves the best generalization ability with an average F1 value of 83% for heterogeneous datasets, but the classification effect decreases by 5% compared to homogeneous datasets. Since the PROMISE dataset does not cover all demand item types, this impacts the DBGAT model feature learning, which in turn weakens the generalization ability of the model. It is found that machine learning models were less effective in classifying the requirements for unknown projects.

Among the three graph-construction methods, constituency graphs have the worst generalization effect. They rely on sentence phrase grammar for classification, which may not be suitable for this type of task. The Random Forest algorithm ranked fourth has high generalization performance due to its unique voting mechanism but low overall classification performance. In summary, we can find the GNN itself has some generalization ability, and adding BERT can improve the model’s classification performance. The DBGAT proposed herein achieved the current optimal classification effect on the four subclasses of NFR.
classification results in the task. However, the DBGAT\textsubscript{Const}
underperformed only 74% in the NFR multi-classification task. Here, the weighted average F1-score of the DBGAT
model reached 83%, which is the best result. We attribute this to the dependency graph approach that captures sentence structure and syntactic information and the ability of BERT
to grasp linguistic subtleties better.

E. THREATS TO VALIDITY

1) To reduce the potential risk of experimental construct validity, we used a widely used experimental design and
metric to fine-tune hyperparameters by experimenting with different parameter configurations.
2) The items in the dataset may not represent all types of items. The PROMISE dataset written by students is not representative of industry standards. In addition, some of the data in this dataset are mislabeled, resulting in missing requirements.
3) Unbalanced datasets may threaten the validity of statistical findings. The numbers corresponding to different categories in the dataset we used vary widely. There are currently not enough open requirements datasets in the field, which is a well-known problem in requirements engineering.

V. CONCLUSION

In this paper, a new requirements classification model is proposed that integrates a graph model into the requirements representation. The proposed method considers the sentence structure and syntactic information of the requirements and introduces BERT for node initialization and embedding to enhance the generalization ability of the model. We compared our results with other state-of-the-art methods. The performance of DBGAT on four NFR subclasses classification and all NFR subclasses classification is better than all methods that do not require manual pre-processing, and the performance on the FR/NFR classification task is similar. When applied to unknown projects, the model maintains high classification performance and can be involved in practice requirements classification tasks.

Overall, graph neural network-based models perform well in demand classification. Our current research focuses on graph neural networks with static graph structures. In contrast, dynamic graph structures have some potential in non-functional requirement classification tasks, and we will work on dynamic graph research in the future. In addition, the available public data in requirements engineering is not comprehensive enough. We will subsequently expand the data volume through data annotation by our team to improve the model capability and better adapt to the actual project requirements.

ACKNOWLEDGMENT

The authors would like to thank LetPub (www.letpub.com) for its linguistic assistance during the preparation of this manuscript.

REFERENCES

[1] V. Bajpai and R. P. Gorthi, “On non-functional requirements: A survey,” in Proc. IEEE Students Conf. Electr., Electron. Comput. Sci., Mar. 2012, pp. 1–4, doi: 10.1109/SCECS.2012.6184810.
[2] M. Glinz, “On non-functional requirements,” in Proc. 15th IEEE Int. Requirements Eng. Conf. (RE), Oct. 2007, pp. 21–26, doi: 10.1109/RE.2007.45.
[3] F.-L. Li, J. Horkoff, J. Mylopoulos, R. S. S. Guizzardi, G. Guizzardi, A. Borgida, and L. Liu, “Non-functional requirements as qualities, with a spicce of ontology,” in Proc. IEEE 22nd Int. Requirements Eng. Conf. (RE), Aug. 2014, pp. 293–302, doi: 10.1109/RE.2014.6912271.
[4] E. Knauss, S. Huymb, K. Schneider, S. Islam, and J. Jürjens, “Supporting requirements engineers in recognising security issues,” in Proc. Int. Work. Conf. Requirements Eng., Found. Softw. Qual., 2011, pp. 4–18, doi: 10.1007/978-3-642-19588-2_2.
[5] J. Cleland-Huang, R. Settimi, X. Zou, and P. Sole, “Automated classification of non-functional requirements,” Requir. Eng., vol. 12, no. 2, pp. 103–120, Apr. 2007, doi: 10.1007/s00766-007-0045-1.
[6] M. Ezzati-Karami and N. H. Madhavji, “Automatically classifying non-functional requirements with feature extraction and supervised machine learning techniques: A research preview,” in Proc. Int. Work. Conf. Requirements Eng., Found. Softw. Qual., 2021, pp. 71–78, doi: 10.1007/978-3-030-73128-1_5.
[7] I. Hussain, L. Kosseim, and O. Ormandjieva, “Using linguistic knowledge to classify non-functional requirements in SRS documents,” in Proc. Int. Conf. Appl. Natural Lang. Inf. Syst., 2008, pp. 287–298, doi: 10.1007/978-3-540-69858-6_28.
[8] Z. S. H. Abad, O. Karras, P. Ghazi, M. Glinz, G. Ruhe, and K. Schneider, “What works better? A study of classifying requirements,” in Proc. IEEE 25th Int. Requirements Eng. Conf. (RE), Sep. 2017, pp. 496–501, doi: 10.1109/RE.2017.36.
[9] Z. Kurtanovic and W. Maalej, “Automatically classifying functional and non-functional requirements using supervised machine learning,” in Proc. IEEE 25th Int. Requirements Eng. Conf. (RE), Sep. 2017, pp. 490–495, doi: 10.1109/RE.2017.32.
[10] I. Winkler and A. Vogelsang, “Automatic classification of requirements based on convolutional neural networks,” in Proc. IEEE 24th Int. Requirements Eng. Conf. Workshops (REW), Sep. 2016, pp. 39–45, doi: 10.1109/REW.2016.021.
[11] T. Mikolov, M. Karafiát, L. Burget, J. Černocký, and S. Khudanpur, “Recurrent neural network based language model,” in Proc. INTERSPEECH, 2010, vol. 2, no. 3, pp. 1045–1048. [Online]. Available: https://www.isca-speech.org/archive_v0/interspeech_2010/i10_1045.html
[12] A. Dekhtyar and V. Fong, “RE data challenge: Requirements identification with Word2Vec and TensorFlow,” in Proc. IEEE 25th Int. Requirements Eng. Conf. (RE), Sep. 2017, pp. 484–489, doi: 10.1109/RE.2017.26.
[13] T. Mikolov, I. Sutskever, K. Chen, G. S. Corrado, and J. Dean, “Distributed representations of words and phrases and their compositionality,” in Proc. Adv. Neural Inf. Process. Syst., 2013, pp. 3111–3119. [Online]. Available: http://papers.nips.cc/paper/5021-distributed-representations-of-words-and-phrases-and-their-com.pdf
[14] A. Rahimi, F. Eassa, and L. Elrefai, “One-and-two phase software requirement classification using ensemble deep learning,” Entropy, vol. 23, no. 10, p. 1264, 2021, doi: 10.3390/e23101264.
[15] L. Wu, Y. Chen, K. Shen, X. Guo, H. Gao, S. Li, J. Pei, and B. Long, “Graph neural networks for natural language processing: A survey,” 2021, arXiv:2106.06090.
[16] F. Dalpiaz, D. Dell’Anna, F. B. Aydemir, and S. Çevikol, “Requirements classification with interpretable machine learning and dependency parsing,” in Proc. IEEE 27th Int. Requirements Eng. Conf. (RE), Sep. 2019, pp. 142–152, doi: 10.1109/RE.2019.00025.
[17] L. Zhao, W. Alhosan, A. Ferrari, K. J. Letsholo, M. A. Ajagbe, E.-V. Chioasca, and R. T. Batista-Navarro, “Natural language processing for requirements engineering: A systematic mapping study,” ACM Comput. Surv., vol. 54, no. 3, pp. 1–41, Jun. 2021, doi: 10.1145/3444689.
[18] J. Devlin, M.-W. Chang, K. Lee, and K. Toutanova, “BERT: Pre-training of deep bidirectional transformers for language understanding,” 2018, arXiv:1810.04805.
[19] F. Scarselli, M. Gori, A. C. Tsoi, M. Hagenbuchner, and G. Monfardini, “The graph neural network model,” IEEE Trans. Neural Netw., vol. 20, no. 1, pp. 61–80, Jan. 2009, doi: 10.1109/TNN.2008.2005605.
20] C. Malaviya, C. Bhagavatula, A. Bosselut, and Y. Choi, “Commonsense knowledge base completion with structural and semantic context,” in Proc. AAAI Conf. Artif. Intell., Apr. 2020, vol. 34, no. 3, pp. 2925–2933, doi: 10.1609/aaai.v34i3.5684.

21] M. Qu, T. Gao, L. P. Xionneux, and J. Tang, “Few-shot relation extraction via Bayesian meta-learning on relation graphs,” in Proc. Int. Conf. Mach. Learn., 2020, pp. 7867–7876.

22] Y. Liu, S. Feng, D. Wang, K. Song, F. Ren, and Y. Zhang, “A graph reasoning network for multi-turn response selection via customized pre-training,” 2020, arXiv:2012.11099. [Online]. Available: https://www.arxiv.org/AAAI21Papers/AAAI-5296-LiuY.pdf.

23] P. Kapanipathi, V. Thost, S. S. Patel, S. Whitehead, I. Abdelaziz, A. Balakrishnan, and A. Fokoue, “Infusing knowledge into the textual entailment task using graph convolutional networks,” in Proc. AAAI Conf. Artif. Intell., 2020, vol. 34, no. 5, pp. 8074–8081, doi: 10.1609/aaai.v34i05.6318.

24] P. Veličković, G. Cucurull, A. Casanova, A. Romero, P. Liò, and Y. Bengio, “Graph attention networks,” 2017, arXiv:1710.10903.

25] T. Hey, J. Keim, A. Koziolek, and W. F. Tichy, “NoRBERT: Transfer learning for requirements classification,” in Proc. IEEE 28th Int. Requirements Eng. Conf. (RE), Aug. 2020, pp. 169–179, doi: 10.1109/RE48521.2020.00028.

26] D. Kici, G. Malik, M. Cevik, D. Parikh, and A. Başar, “A BERT-based transfer learning approach to text classification on software requirements specifications,” in Proc. 34th Int. Conf. Softw. Eng., 2021, pp. 1–13. [Online]. Available: https://assets.pubpub.org/rrc10a/ja/31621568150182.pdf.

27] J. Mylopoulos, L. Chung, and B. Nixon, “Representing and using non-functional requirements: A process-oriented approach,” IEEE Trans. Softw. Eng., vol. 18, no. 6, pp. 483–497, Jun. 1992, doi: 10.1109/32.142871.

28] H. Alrumaih, A. Mirza, and H. Alsalama, “Domain ontology for requirements classification in requirements engineering context,” IEEE Access, vol. 8, pp. 89899–89908, 2020, doi: 10.1109/ACCESS.2020.2993838.

29] L. Chung, B. A. Nixon, E. Yu, and J. Mylopoulos, Non-Functional Requirements in Software Engineering, New York, NY, USA: Springer, 2000, pp. 89–142, doi: 10.1007/978-1-4615-5269-7.

30] M. Broy, “Rethinking nonfunctional software requirements,” Computer, vol. 48, no. 5, pp. 96–99, May 2015, doi: 10/1109/MC.2015.139.

31] J. Eckhardt, A. Vogelsang, and D. M. Fernández, “Are ‘non-functional’ requirements really non-functional?: Ans investigation of non-functional requirements in practice,” in Proc. 38th Int. Conf. Softw. Eng., May 2016, pp. 832–842, doi: 10.1145/2884781.2884788.

32] A. Mahmoud, “An information theoretic approach for extracting and tracing non-functional requirements,” in Proc. IEEE 23rd Int. Requirements Eng. Conf. (RE), Aug. 2015, pp. 36–45, doi: 10.1109/RE.2015.7320406.

33] R. Navarro-Almanza, R. Juarez-Ramirez, and G. Licea, “Towards supporting software engineering using deep learning: A case of software requirements classification,” in Proc. 5th Int. Conf. Softw. Eng. Res. Innov. (CONISOFT), Oct. 2017, pp. 116–120, doi: 10.1109/CONISOFT.2017.00021.

34] C. Baker, L. Deng, S. Chakraborty, and J. Dehlinger, “Automatic multi-class non-functional software requirements classification using neural networks,” in Proc. IEEE 43rd Annu. Comput. Softw. Appl. Conf. (COMPSAC), Jul. 2019, pp. 610–615, doi: 10.1109/COMPSAC.2019.10273.

35] H. Tang, Y. Mi, F. Xue, and Y. Cao, “An integration model based on graph convolutional network for text classification,” IEEE Access, vol. 8, pp. 148865–148876, 2020, doi: 10.1109/ACCESS.2020.3015770.

36] T. N. Kipf and M. Welling, “Semi-supervised classification with graph convolutional networks,” 2016, arXiv:1609.02907.

37] M. Henaff, J. Bruna, and Y. LeCun, “Deep convolutional networks on graph-structured data,” 2015, arXiv:1506.05162.

38] H. Peng, J. Li, Y. He, Y. Liu, M. Bao, L. Wang, Y. Song, and Q. Yang, “Large-scale hierarchical text classification with recursively regularized deep graph-CNN,” in Proc. World Wide Web Conf. (WWW), 2018, pp. 1063–1072, doi: 10.1145/3178876.3186005.

39] M. Defferrard, X. Bresson, and P. Vandergheynst, “Convolutional neural networks on graphs with fast localized spectral filtering,” in Proc. Adv. Neural Inf. Process. Syst., vol. 29, 2016, pp. 3844–3852.

40] L. Yao, C. Mao, and Y. Luo, “Graph convolutional networks for text classification,” in Proc. AAI Conf. Artif. Intell., 2019, vol. 33, no. 1, pp. 7370–7377, doi: 10.1609/aaai.v33i01.33017370.

41] L. Cai, Y. Song, T. Liu, and K. Zhang, “A hybrid BERT model that incorporates label semantics via attentive adjustment for multi-label text classification,” IEEE Access, vol. 8, pp. 152183–152192, 2020, doi: 10.1109/ACCESS.2020.3017382.

42] K. Xu, L. Wu, Z. Wang, Y. Feng, M. Witbrock, and V. Shピンin, “Graph2Seq: Graph to sequence learning with attention-based neural networks,” 2018, arXiv:1804.00823.

43] (2013). Concordia RE Corpus. [Online]. Available: http://semanticsoftware.info/re-corpus

44] 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 Proc. IEEE 37th Annu. Comput. Softw. Appl. Conf., Jul. 2013, pp. 381–386, doi: 10.1109/COMPSAC.2013.64.

45] S. Li, L. Wu, S. Feng, F. Xu, F. Xu, and S. Zhong, “Graph-to-tree neural networks for learning structured input-output translation with applications to semantic parsing and math word problem,” 2020, arXiv:2004.13781.

46] S. Vashishth, R. Joshi, S. S. Prayaga, C. Bhattacharyya, and P. Talukdar, “RESIDE: Improving distantly-supervised neural relation extraction using side information,” 2018, arXiv:1812.04361.

47] Y. Chen, L. Wu, and M. J. Zaki, “Iterative deep graph learning for graph neural networks: Better and robust node embeddings,” 2020, arXiv:2006.13009.

GANG LI received the master’s degree in management from Shandong University and the Ph.D. degree in management science and engineering from the Harbin Institute of Technology. He was a Professor in software engineering with the Qilu University of Technology (Shandong Academic of Sciences). His research interests include big data analytics applications, data governance, digital economy, and digital government.

CHENPENG ZHENG received the bachelor’s degree in software engineering from the Qilu University of Technology (Shandong Academic of Sciences). His research interests include software requirements engineering, natural language processing, machine learning, and deep learning.

MIN LI received the master’s degree in communications engineering from Tianjin University. She was a Professor in software engineering with the Qilu University of Technology (Shandong Academic of Sciences). Her research interests include information technology standardization, economic and information development, big data analysis and application, data governance and data openness, and digital government planning and evaluation.

HAOSEN WANG is currently pursuing the master’s degree with the School of Information Management and Artificial Intelligence, Zhejiang University of Finance and Economics, China. His major is management science and engineering. He has published in ACM Transactions on Information Systems and Expert Systems with Applications. His research interests include recommender systems, graph representation learning, graph neural networks, and machine learning.