The Journal of Systems & Software 190 (2022) 111324

Contents lists available at ScienceDirect

The Journal of Systems & Software

journal homepage: www.elsevier.com/locate/jss

Graph4Web: A relation-aware graph attention network for web service classification ☆

Kunsong Zhao [a], Jin Liu [a,*], Zhou Xu [b,c,*], Xiao Liu [d], Lei Xue [e], Zhiwen Xie [a], Yuxuan Zhou [f], Xin Wang [a]

[a] School of Computer Science, Wuhan University, Wuhan, China
[b] Key Laboratory of Dependable Service Computing in Cyber Physical Society (Chongqing University), Ministry of Education, China
[c] School of Big Data and Software Engineering, Chongqing University, Chongqing, China
[d] School of Information Technology, Deakin University, Geelong, Australia
[e] Department of Computing, The Hong Kong Polytechnic University, Hong Kong, China
[f] College of Engineering and Computer Science, Syracuse University, Syracuse, USA

ARTICLE INFO

Article history:
Received 28 September 2021
Received in revised form 18 March 2022
Accepted 4 April 2022
Available online 9 April 2022

Keywords:
Web services
Graph neural network
Attention mechanism
Service discovery

ABSTRACT

Software reuse is a popular way to utilize existing software components to ensure the quality of newly developed software in service-oriented architecture. However, how to find a suitable web service from existing repositories to meet requirements is still an open issue. Among others, web service classification is one of the most essential and effective means for web service recommendation. Previous studies have concerned this problem, but a critical issue, i.e., the semantic and syntactic information for the web service, is often ignored. To address such an issue, in this work, we propose *Graph4Web*, which uses a relation-aware graph attention network for web service classification. Specifically, we first parse the web service description sequence into the dependency graph and initialize the embedding vector of each node in the graph by tuning the pre-trained BERT model. We further propose a *relation-aware graph attention* layer to learn and update the node embedding vector by aggregating the information of neighborhood nodes and the distinct types of relationships between nodes. In addition, we introduce the self-attention mechanism to acquire the high-level global representation for web service classification. Various experiments demonstrate that *Graph4Web* has better classification performance compared with seven baseline methods with three indicators.

1. Introduction

Software, as an Internet product or service, has brought unprecedented influence on all aspects of people's daily life. However, as the number, scale, and complexity of software increases, how to ensure the quality of newly developed software is critical to software development (Buckley and Poston, 1984; Zhang et al., 2016; Yu et al., 2019). Software reuse, as a means to develop new software products with similar functions by virtue of existing software components, has become a popular way during the software development process (Barros-Justo et al., 2018; Imoize et al., 2019).

As the service-oriented architecture became popular, web services turned into an indispensable part in modern software development. Web services provide a basic composition with high cohesion and loose coupling to support responses among heterogeneous software components (Wang et al., 2020b; Yang et al., 2020, 2018b), which have the possibility to lessen the development cost and promote the robustness of the software. Web services are also treated as valuable resources for software reuse (Yang et al., 2018a). The popular web service repositories, such as ProgrammableWeb[1] and Public APIs,[2] contain a mass of web services for beginners and developers to choose from. Nevertheless, the large number of web services also makes it difficult to select the suitable services. Thus, the key to reuse software components lies in how to find the appropriate web services from repositories to meet the requirements of developers in specific application scenarios, which is a hot research topic in service discovery (Elshater et al., 2015; Hajlaoui et al., 2017).

As an effective way, web services classification has shown the powerful ability for the service discovery task (Elgazzar et al., 2010). The aim of web service classification task is to determine which category one web service belongs to. When issuing a

☆ Editor: A. Bertolino.

* Corresponding authors.
E-mail addresses: jinliu@whu.edu.cn (J. Liu), zhouxullx@cqu.edu.cn (Z. Xu).

[1] http://www.programmableweb.com.
[2] https://github.com/public-apis/public-apis.

https://doi.org/10.1016/j.jss.2022.111324
0164-1212/

new web service on the repository, such as ProgrammableWeb, developers need to provide the description document explaining the functionality and select a category to which this web service belongs because of the platform specification. However, there are more than 500 categories in ProgrammableWeb site, which makes it difficult for developers to select. In addition, the well-categorized web services make the service discovery and service composition easier and it will improve the software reuse process because more suitable and high-quality software components (i.e., web services) will be found by developers (Atkinson et al., 2007; Wang et al., 2017; Yang et al., 2020).

Previous studies (Elshater et al., 2015; Hao et al., 2010; Liu and Wong, 2009) proposed to use traditional machine learning techniques for this purpose. With the rise of neural networks, deep learning techniques with strong abilities for representation learning has been widely applied by researchers to a variety of domain-specific tasks, such as natural language processing (Xie et al., 2020a,b) and software engineering (Wang et al., 2021c; Xu et al., 2021). Recently, Yang et al. (2020) proposed ServeNet-BERT that adopted the integration of **Bi**-directional **L**ong **S**hort-**T**erm **M**emory (**Bi-LSTM**) and **C**onvolutional **N**eural **N**etwork (**CNN**) to learn the high-level feature representation from the description and name information of web services and then combined them into unified representation for web service classification. The performance of their model was further enhanced by the **B**idirectional **E**ncoder **R**epresentation from **T**ransformers (**BERT**) model (Devlin et al., 2018). As web service documents are the unstructured textual sequences, modeling them directly cannot well reveal the implicit dependent relationship between words. As previous studies suggested (Guo et al., 2019; Tian et al., 2021; Vashishth et al., 2018; Yao et al., 2019), one way mining such implicit information is to parse the web service document into structural representation, such as the graph. Inspired by these studies, we seek to make the web service document structured. In addition, we also want to further learn the semantic and syntactic information hidden in this structural representation.

Recently, **G**raph **N**eural **N**etworks (**GNN**) with the powerful ability to learn feature representation from structured data has been applied to text-based representation learning and classification tasks (Huang et al., 2019; Wang et al., 2020c; Yao et al., 2019; Wang et al., 2021a,b; Yu et al., 2022). Some studies (Dozat and Manning, 2016; Wang et al., 2020c) parsed the text sequence into the dependency graph and then used the GNN model to update the feature representation. Following them, in this work, we propose the ***Graph4Web*** model that uses a relation-aware graph attention network for web service classification. This model adopts a GNN-based architecture because of its potential to learn and express implicit information hidden in structured data (Huang et al., 2019; Wang et al., 2020c). Specifically, we first employ the Biaffine Parser (Dozat and Manning, 2016) to encode the web service description sequence into the dependency graph that holds the abundant semantic and syntactic information. We further propose a **R**elation-**A**ware **G**raph **A**ttention (**RAGA**) layer to update the node representation in the dependency graph. Our RAGA layer simultaneously aggregates the neighbors information and the different types of relationship information between neighborhood nodes to learn and update the node representations. Then, we produce the global representation for the dependency graph by the virtue of the self-attention mechanism for web service classification.

We evaluate our proposed Graph4Web model on the real-world web service dataset collected from the ProgrammbleWeb site with three performance indicators. Our experimental results demonstrate that Graph4Web achieves average Precision, Recall, and F-measure values of 0.702, 0.690, and 0.693, respectively. Compared with the seven representative methods, Graph4Web obtains average improvements by 39.1%, 49.2%, and 50.3% in terms of the three indicators, respectively.

We summarize the main contributions of this paper as follows:

- To the best of our knowledge, this is the first work which builds the dependency graph and introduces the relation-aware graph attention network for web service classification.
- We propose a RAGA layer that considers the neighbors' information and the different types of relationship information between neighborhood nodes simultaneously to update the node representations.
- We conduct comprehensive experiments on real-world web services and evaluate our model with three performance indicators. The results show the superiority of our proposed model compared with seven representative methods.

The rest of this paper is organized as follows. Section 2 provides a motivating example. Section 3 introduces the related work. The proposed Graph4Web model is described in Section 4. Section 5 and Section 6 detail the experimental setup and results, respectively. Section 7 further discusses our proposed method. Section 8 discusses the threats to validity of our work. Finally, we conclude our work in Section 9.

2. Motivating example

Tom is a senior programmer whose daily work is to develop new software components or libraries to meet requirements proposed by users and other developers. One day, his project manager requests him to develop a new software component, i.e., the web service, and make it issue on public service repositories, such as the ProgrammableWeb. After coding finished, he describes the functionality of this web service in detail because of the development specification, i.e., the web service description document. When issuing, the platform requires the publisher select a specific category to which the web service belongs, aiming at making the management and discovery process easier for others. However, there are more than 500 categories can be chosen, which brings the trouble of time consumption and precision to developers because well-organized web services facilitate the web service discovery and composition (Elgazzar et al., 2010; Yang et al., 2019, 2020). With the help of our Graph4Web model, Tom can easily determine which category this newly developed web service belongs to by adopting the information at hand, i.e., the web service description document.

3. Related work

As our aim is to determine which category one web service belongs to, we first introduce some existing studies for web service classification and recommendation tasks. Liu and Wong (2009) proposed an approach that extracted four functionality characteristics from the web service description language (WSDL) by adopting textual mining techniques for web service clustering. Liu and Fulia (2015) introduced the probabilistic matrix to obtain the use-related and service-related features from WSDL and employed the probabilistic topic model to acquire topic-related features. Then, these latent features are used for web service recommendation. Aznag et al. (2014) incorporated the correlated topic model and formal concept analysis technique to mine features from WSDL and cluster the web services. Katakis et al. (2009) proposed to use both the textual description and semantic annotations of service-based web ontology language for feature representation and employ the classifier ensemble for web service classification automatically. Wang et al. (2010) used the functional features of web services and employed the support vector machine (SVM) algorithm for classifying web services. Fang et al. (2012) proposed an automatic method that

introduced the clustering techniques to tag web services by using the web services description language documents. Kapitsaki (2014) extracted the features for web services by using the Term Frequency–Inverse Document Frequency (TF–IDF) technique and adopted seven classifiers for web service segments classification. Nisa and Qamar (2015) first extracted features by parsing web services description language documents and took the maximum entropy as the criterion for web service classification. Elshater et al. (2015) developed the goDiscovery model that utilized the statistical model (i.e., TF–IDF) and the indexing technique (i.e., K-Dimensional tree index) for web service discovery. Liu et al. (2016a) proposed an active learning method, called LDA-SVM, which introduced the Latent Dirichlet Allocation algorithm and utilized the SVM for web service classification. Liu et al. (2016b) identified the web services using the ontology concept and introduced the Naive Bayes theory for web service classification. Shi et al. (2017) incorporated the active learning with the correlation-aware learning strategy for web service tag recommendation task.

Ye et al. (2019) proposed the Wide&Bi-LSTM model that used the wide learning model to combine all the discrete features in the web service descriptions for breadth learning and employed the Bi-LSTM model for depth learning. The results from the two aspects are integrated for web service classification. Cao et al. (2019) proposed the LAB-BiLSTM method that combined the local feature representation produced by Bi-LSTM model with the global topic representation via the attention mechanism for web service classification. Yang et al. (2019) proposed the ServeNet that used the Bi-LSTM model to encode the service descriptions and adopted the CNN model for obtaining the representation for web service classification. Moreover, Yang et al. (2020) improved ServeNet and proposed ServeNet-BERT that applied the BERT model to produce the embedding vectors and this model obtained the state-of-the-art performance for web service classification.

Traditional studies for web service discovery and classification always extracted related features based on feature engineering techniques which highly relies on expert efforts and the quality of features severely impact the classification performance. In addition, some studies directly treated the web service description as the textual sequence which to some extend lost the inherent dependent relationships between words. Different from the above studies, we parse the web service description into the structural dependency graph that reserves the implicit relation information. We further use the relation-aware graph attention network to update the node embedding vectors in the graph.

As the basic architecture of our model is the GNN, we also present the work related to it. Kipf and Welling (2016) was the first to develop a graph structure based semi-supervised learning model **G**raph **C**onvolutional **N**etwork (**GCN**). This model encoded the graph structure and node information leveraging the first-order approximation of spectral graph convolutions. GCN resorted the eigendecomposition of the Laplacian matrix to realize the graph convolution operation. However, it is difficult to decomposite eigen of the Laplacian matrix when faced with large-scale graphs. To solve this problem, Veličković et al. (2017) proposed the GAT network that leveraged the masked self-attention mechanism and the neighborhood nodes to update the node features. Busbridge et al. (2019) further improved the GAT and developed the RGAT model aggregating the relational information between nodes.

Different from the above studies, we apply the graph learning to the web service classification and develop the Graph4Web model considering both the neighborhood nodes' information and their relationship information simultaneously. Our model reserves both the syntactic and the semantic information in the web service description.

4. Proposed method

4.1. Overall framework

Fig. 1 elaborates an overview of our proposed relation-aware graph attention network. As our model adopts the graph neural network to update the feature embedding for each node, we start from parsing the description sequence into a dependency graph. Note that words in the description sequence are represented by nodes in the built graph. Each node embedding is initialized by the BERT model. To learn the high-quality node representation, we propose a RAGA layer that takes into account both the neighborhood information of the node and their relationship information simultaneously. After stacking multiple RAGA layers, our model is able to incorporate multi-hop neighborhood information for each node. To obtain the global representation, we introduce the self-attention mechanism that integrates all the node embedding into a high-level graph representation. Finally, the softmax function is used to determine which category the web service belongs to.

4.2. Dependency graph parsing

As shown in Fig. 2, a web service description sequence[3] provides its functional details. As the natural language description of web service inherently contains the dependence relationship between words, to start with, we pursue a parsing technique to find these dependence relationships that hold the semantic and syntactic information from the description. One way to cater to this idea is by means of the graph structure. The Biaffine Parser (Dozat and Manning, 2016) is a biaffine based neural dependency parsing technique, which uses the Bi-LSTM with a biaffine attention to substitute the traditional bi-linear attention mechanism. Then, it applied a biaffine dependency label classification model to decide the dependent head of each word. This parsing technique has the potential to model the prior probability of each word under any dependent relationships and the probability that satisfies a specific dependent relationship directly.

Take the sentence "*Junction Networks offers a variety of business VoIP services*" as an example, we apply the Biaffine Parser to obtain the dependence relationship among all words, and the parsing result is depicted in Fig. 3. The red oval means the root node of this sentence after parsing. The syntactic dependencies *DEP*, *PREP*, *POBJ*, and *AMOD* refer to the dependent, prepositional modifier, prepositional object, and adjective modifier, respectively.

A web service can be formalized as $s = \{Name, Des, c\}$ where *Name*, *Des*, and c denote the name, description sequence, and category, respectively. For the description sequence, we use the Biaffine Parser to parse it into a graph-based dependency. Based on the parsing results, we can construct the dependent relationship graph $G = \{V, E\}$ where V and E represent the corresponding node set and edge set, respectively. In our work, we take the words as the nodes in the graph and the edges as the dependent relationship between words. Note that, we only employ the Biaffine Parser on the descriptions because there exist the inherent relations in these description contexts. Instead, the service name just provides a high-level symbol that refers to this service but the dependency is scarce. We will analyze the impact of the service name in Section 6.3.

[3] http://www.programmableweb.com/api/junction-networks-web-services.

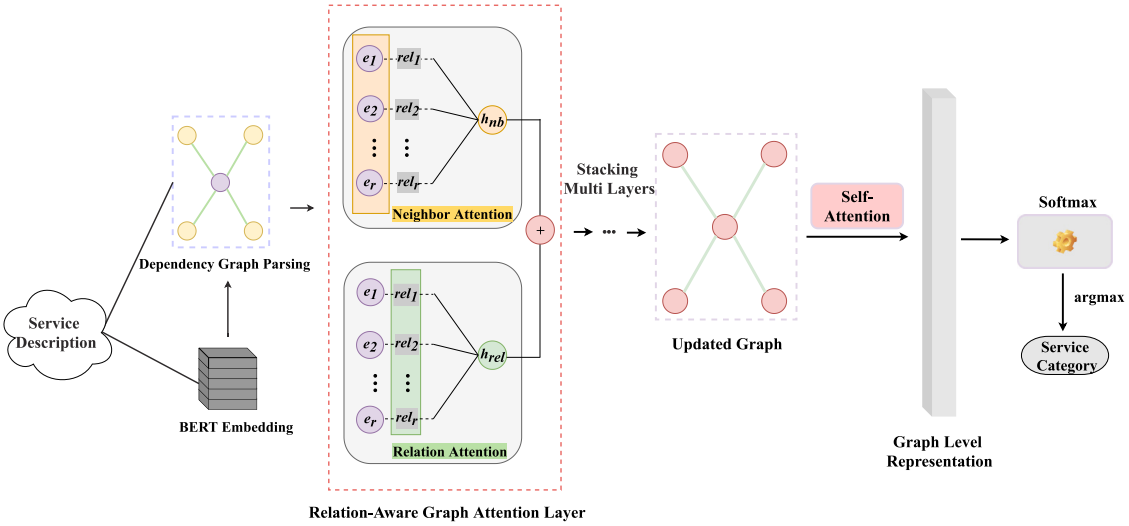

Fig. 1. Overall framework of our model.

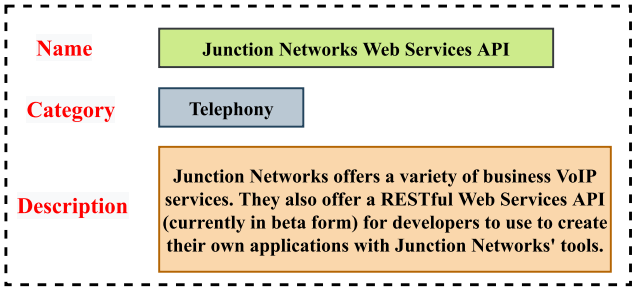

Fig. 2. An example of web service.

4.3. BERT embedding

As mentioned above, the web service contains the description consisting of the word sequence. To obtain the feature embeddings for the words, we employ the BERT model that has achieved promising performance in many text-based tasks (Devlin et al., 2018; Tang et al., 2020; Wang et al., 2020c; Yang et al., 2020). BERT is a deep auto-encoder model that excavates the semantic information in the word sequence by pre-training two tasks: the masked language model and next sentence prediction. It first randomly masks 15% of words in the input sequence and then restores these words by means of the deep Transformers (Vaswani et al., 2017). To catering the input format of BERT, two special signs [CLS] and [SEP] are added at the beginning and end of the sentence individually, in which [CLS] possesses the semantic information among the whole input sentence and [SEP] refers to the separator. After training, the BERT model can output the semantic representation of each word in the sentence, which is consistent with the order of the input sentence.

In this work, we first tokenize the service description sequence and utilize the BERT model to embed it to obtain the initial embedding for each word. More specifically, for a service description sequence $Des = \{w_1, w_2, \ldots, w_n\}$ where n denotes the sequence length, we fine-tune the pre-trained BERT model to obtain the initial word embeddings, which are formulized as:

$$E = \text{BERT}(Des) \tag{1}$$

where $E = \{e_i, e_2, \ldots, e_n\}$ denotes the embedding vector and e_i corresponds to the word w_i. $e_i \in \mathbb{R}^{N_{emb}}$ and N_{emb} is the dimensionality of the embedding vector.

4.4. Relation-aware graph attention layer

After parsing the service description sequence by analyzing its grammatical information, we can obtain its graph structure representation that contains the dependent relationship between words. To further encode the graph node information, we propose the RAGA layer that incorporates both the neighbor information of each node and the relationship information between two adjacent nodes.

Graph Attention Network. As the neighbors' information contains the dependent relationship between nodes, we employ the **G**raph **At**tention network (**GAT**) (Veličković et al., 2017) to integrate the neighborhood information into the node. Concretely, for the node set $V = \{v_1, v_2, \ldots, v_n\}$ in a dependency graph G, we first initialize its node feature embedding following Eq. (1). Assume $\mathcal{N}_i$ refers to the neighbor nodes of ith node in the graph, we use the following formula to update the node embedding by absorbing its neighbors information:

$$h_i^{nb} = \sum_{k \in \mathcal{N}_i} \alpha_{ik} \mathbf{W} h_k \tag{2}$$

where h_i^{nb} represents the embedding after updating by the GAT layer, h_k represents the kth neighborhood node, and $\mathbf{W} \in \mathbb{R}^{N'_{emb} \times N_{emb}}$ is a shared weight matrix that can be learned during the training (Veličković et al., 2017). α_{ik} means the normalized attention weight for the kth neighborhood node, which is formulized as:

$$\alpha_{ik} = \frac{exp(score_{ik}^{nb})}{\sum_{k' \in \mathcal{N}_i} exp(score_{ik'}^{nb})} \tag{3}$$

where $exp(\cdot)$ means the exponential function. $score^{nb}$ is a scoring function that determines how important one node is to another, which can be computed by the following formula:

$$score_{ik}^{nb} = \text{LeakyRelu}(\mathbf{W}'[e_i \parallel e_k]) \tag{4}$$

where LeakyRelu (Maas et al., 2013) is an activation function, **W'** is a learnable vector, and $\parallel$ represents the concatenation operation between two embedding vectors.

The RAGA Layer. The GAT network updates the node embedding by incorporating its neighborhood node information, which ignores the relation information between nodes, i.e., the dependency produced during the description parsing phase. Intuitively, the neighbor nodes with different dependent relationships have

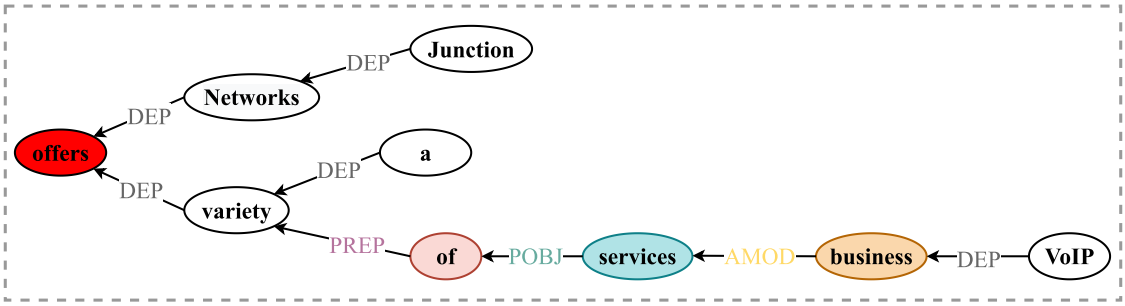

Fig. 3. An example of the parsing result by the Biaffine Parser.

distinct impacts on the node representation. To this end, we follow the previous study (Wang et al., 2020c) by taking into account the relation information during the update procedure of the node embedding. More specifically, we first initialize the embedding vector for the relation between nodes e_i and e_k by the following formula:

$$rel_{ik} = f_{emb}(e_i, e_k) \tag{5}$$

where $rel_{ik} \in \mathbb{R}^{r_{emb}}$ represents the relation embedding vector and r_{emb} is the dimensionality of the relation embedding vector. f_{emb} is a learnable function that is initialized with the uniform distribution and updated during the training process. Then, we compute the node embedding by the following formula:

$$h_i^{rel} = \sum_{k \in \mathcal{N}_i} \beta_{ik} h_k \tag{6}$$

where h_i^{rel} represents the embedding by considering the relation information. β_{ik} denotes the relation-aware attention weight for the kth neighborhood node, which is formulized as:

$$\beta_{ik} = \frac{exp(score_{ik}^{rel})}{\sum_{k' \in \mathcal{N}_i} exp(score_{ik'}^{rel})} \tag{7}$$

where $score^{rel}$ is a scoring function that decides the importance of each relationship, which is defined as the following formula:

$$score_{ik}^{rel} = \text{Relu}(W_1 rel_{ik} + b_1)W_2 + b_2 \tag{8}$$

where Relu is an activation function. W_1, b_1, W_2, and b_2 are the learnable parameters during the training process.

After obtaining the embedding vectors h_i^{nb} and h_i^{rel}, we can obtain the final node embedding h_i^{final} by introducing the neighbor information and relation information simultaneously, which is formulized as:

$$h_i^{final} = h_i^{nb} + h_i^{rel} \tag{9}$$

The above update operation only takes the first-order neighbors into consideration. By stacking multiple RAGA layers, we can aggregate the multi-hop neighborhood information and the corresponding relation information.

4.5. Graph level representation

After the above treatment, we obtain the embedding vector at the node level. To produce the global representation of the graph, we introduce the self-attention mechanism (Vaswani et al., 2017; Shaw et al., 2018). Concretely, for the description sequence embedding after updating by our RAGA layers, we can obtain the final embedding vectors $\{h_1^{final}, h_2^{final}, \ldots, h_n^{final}\}$ that correspond to the node set $\{v_1, v_2, \ldots, v_n\}$. Then, we use the following operation to fuse all the node embeddings into a high-level global representation that inside contains the whole information of the graph G, which is denoted as:

$$H = \sum_{i=1}^{n} \gamma_i h_i^{final} \tag{10}$$

where γ_i is a scoring function that is calculated as follows:

$$\gamma_i = \frac{exp(MLP(h_i^{final}))}{\sum_{j=1}^{n} exp(MLP(h_j^{final}))} \tag{11}$$

where MLP is the multi-layer perceptron.

4.6. Model training and classification

The training process is designed in an end-to-end manner to achieve high efficiency. Once obtained the final global representation vector that refers to the web service, we can acquire the probability distribution of every category using a softmax layer. Then, we treat the one that has the maximum probability value as the most probable category for this web service. We use the negative log likelihood loss function to optimize our model, which is defined as:

$$\mathcal{L}(\theta) = -\sum_{i=1}^{m} log(P(c_i|\hat{c}_i, \theta)) \tag{12}$$

where m is the total number of web service instances, c_i and $\hat{c}_i$ represent the true category and predicted category, respectively. θ is the parameter to be optimized.

After the model training is finished, we take the new web service instance tokenized by the BERT model as input, update the embedding, and learn the global representation. Finally, we output the category to which this web service belongs.

5. Experimental setup

5.1. Dataset

To evaluate the performance of our proposed model, we conduct experiments on a dataset collected from the ProgrammableWeb site which consists of 15,344 web services of 401 categories. Each web service contains the name, category, and description (as shown in Fig. 2). Following the previous study (Yang et al., 2020), the services with the empty name, description or category are removed. In addition, to make the dataset more balanced, after counting the scale of each category, some of them including one-shot, few-shot, and small-scale categories are discarded and the big-scale categories are retained as the previous study did (Yang et al., 2020). As a result, 10,943 web services from top 50 categories are reserved to form the final

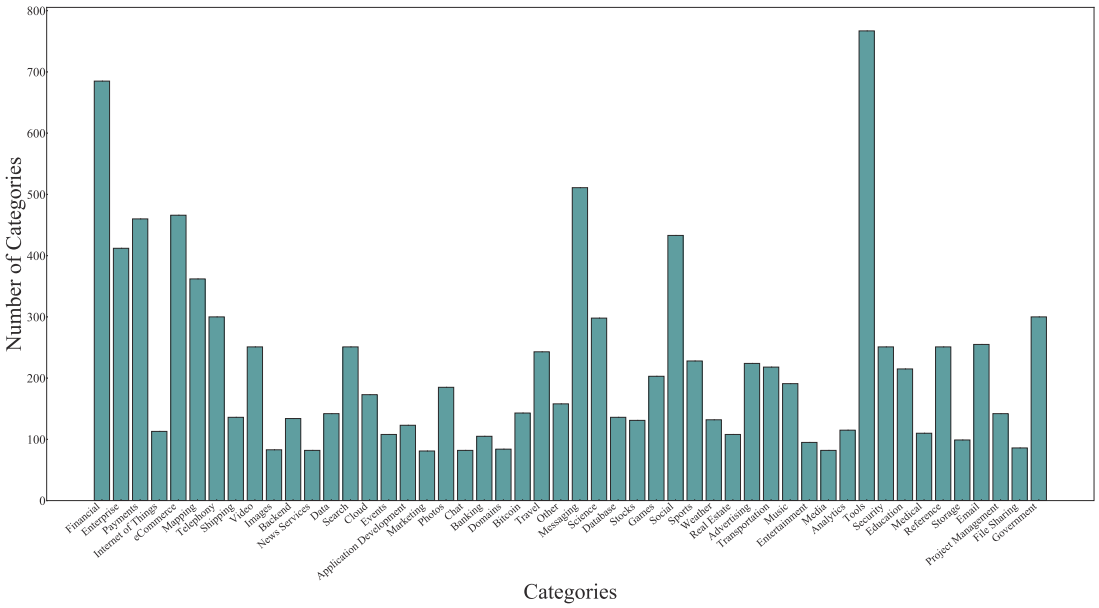

Fig. 4. The statistic information of each category.

dataset for our experiments. Fig. 4 gives the statistic information of each category. We randomly select 80% of the web services as the training set and the remainder as the test set following the same setting in recent studies (Yang et al., 2019, 2020).

5.2. Performance indicators

As our goal is to determine which category one web service belongs to, it can be treated as a multi-classification task. In this work, we employ three confusion matrix based indicators, including Precision, Recall, and F-measure, to evaluate the performance of our model, which are calculated as follows:

$$Precision = \frac{TP}{TP + FP} \tag{13}$$

$$Recall = \frac{TP}{TP + FN} \tag{14}$$

$$F\text{-}measure = \frac{2 \times Precision \times Recall}{Precision + Recall} \tag{15}$$

where TP indicates the number of web services that are correctly predicted as the category, FP indicates the number of services with other categories but are predicted as the category, and FN indicates the number of web services with the category but are predicted as other categories.

All the used three indicators are range from 0 to 1. The larger value indicates that the model achieves better classification performance. These indicators are commonly used in previous studies (Ferenc et al., 2020; Gu et al., 2019; Pascarella et al., 2019; Ren et al., 2019; Tang et al., 2021; Wang et al., 2020a; Xu et al., 2019a,b; Zhao et al., 2021c,b,a).

5.3. Parameter configuration

We use the BERT model to embed every node in the graph by parsing the service description into a 768-dimensional embedding vector (Devlin et al., 2018), and the max length of each description sequence m is set as 128. We use two layers of our proposed RAGA layer with the hidden dimension as 768 (Devlin et al., 2018). We train the model 200 epochs with the batch size as 32. To optimize the parameters, the AdamW algorithm (Devlin et al., 2018) is applied with the learning rate as 1e−5.

5.4. Methods for comparison

Seven representative methods including two state-of-the-art models (i.e., ServeNet and ServeNet-BERT) for web service classification and five mainstream models for text classification are adopted for comparison.

- **CNN** (Wang et al., 2018): **C**onvolutional **N**eural **N**etwork (**CNN**) first proposed to extract the feature from the image. For the text classification task, CNN assumes that each word is related to its several neighborhood words. After extracting the feature by the convolutional kernel, the pooling operation is used to obtain the high-level sequence representation.
- **RCNN** (Lai et al., 2015): **R**ecurrent **C**onvolutional **N**eural **N**etwork (**RCNN**) integrates a recurrent structure and the convolutional layer to gather the contextual representation information and relieve the bias issue. Then, a max-pooling operation is used to produce the feature representation at the sequence level.
- **LSTM** (Johnson and Zhang, 2016): This method employs the **L**ong **S**hort-**T**erm **M**emory (**LSTM**) to learn the word representation by incorporating the previous word representations.
- **Bi-LSTM** (Zhang et al., 2015): This method proposed to use a **Bi**-directional **L**ong **S**hort-**T**erm **M**emory (**Bi-LSTM**) architecture that gathers the long distance relationship started from two directions, aiming at learning the high-level sentence representation for the given sequence.
- **C-LSTM** (Zhou et al., 2015): This method combines the CNN and LSTM into a new unified model that has the potential to capture the local and global information simultaneously.
- **ServeNet** (Yang et al., 2019): This method comprises one embedding layer, feature extraction layers, and task layers for web service classification. The embedding layer uses the **Glo**bal **Ve**ctors for word representation (**Glove**) (Pennington et al., 2014) to initialize the service description and then the feature extraction layers combine the CNN and Bi-LSTM models to learn the high-level feature representation. Finally, the task layers utilize a fully connected layer followed by a softmax layer for classification.

- **ServeNet-BERT** (Yang et al., 2020): This method takes the service name and service description information into account for web service classification. For the service description, it first applies the BERT model to obtaining the initialized feature embedding for each word and then uses the CNN and Bi-LSTM models to learn the high-level representation for the whole description. Also, the service name is initialized and pooled by BERT to obtain the feature embedding. Finally, the feature representations between description and name are concatenated, and a fully connected layer followed by a softmax layer is employed for classification. ServeNet-BERT has obtained the state-of-the-art performance on web service classification task.

In all the baseline models, the first five mainstream models, including CNN, RCNN, LSTM, Bi-LSMT, and C-LSTM, only take the web service description as inputs. Then, the feature representation can be leant by these models to determine the probability of each category for web service classification task. In addition, the two state-of-the-art models, i.e., ServeNet and ServeNet-BERT take both the service name and the service description sequence into account. Then, they incorporate the representations of the service name and description to identify which category one web service belongs to.

5.5. Research questions

In this work, we empirically design the following three research questions (RQs) to measure our proposed model.

RQ1: Is our proposed Graph4Web model superior to the state-of-the-art methods?

Motivation: Deep learning techniques have obtained satisfactory performance in the classification task (Kowsari et al., 2017; Minaee et al., 2021). In particular, some previous studies (Yang et al., 2019, 2020) proposed to apply the deep learning techniques for web service classification task and obtained the state-of-the-art performance. This research question is designed to explore whether our proposed Graph4Web model can perform better compared with the existing methods.

RQ2: How effective is our Graph4Web model compared with its variants?

Motivation: Our Graph4Web model first employs BERT to initialize the node embedding vector for the built dependency graph and then adopts the RAGA layers that incorporates both the neighborhood nodes information and the relationship information between them to update the node representation. This question is designed to investigate whether each part used in our Graph4Web model is effective to improve the performance of the web service classification task.

RQ3: Can service name improve the performance of our Graph4Web model?

Motivation: In this work, we only use the service description information for model building, but the web services also contain the service name that refers to it. This question is designed to explore whether aggregating the name information can further improve the classification performance.

6. Experimental results

6.1. RQ1: Is our proposed Graph4Web model superior to the state-of-the-art methods?

Methods: To answer this question, we choose seven representative methods consisting of five commonly used deep learning techniques and two state-of-the-art methods for web service classification (as shown in Section 5.4).

Table 1
The average results for Graph4Web and seven representative methods.

Method	Precision	Recall	F-measure
CNN	0.287	0.237	0.238
RCNN	0.601	0.551	0.561
LSTM	0.486	0.446	0.441
Bi-LSTM	0.586	0.568	0.562
C-LSTM	0.556	0.550	0.536
ServeNet	0.611	0.578	0.581
ServeNet-BERT	0.677	0.658	0.662
Graph4Web	**0.702**	**0.690**	**0.693**

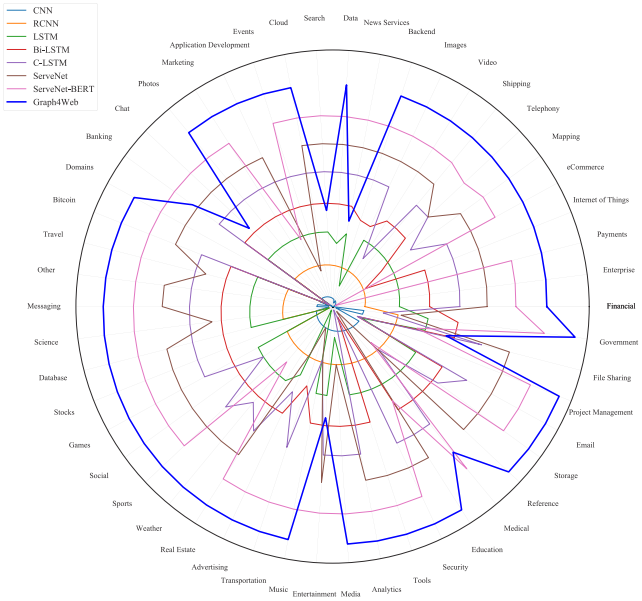

Fig. 5. The radar map for F-measure among all categories.

Results: Table 1 demonstrates the average results of our Graph4Web model and the seven comparative methods in terms of Precision, Recall, and F-measure, respectively. We can see from this table that, in terms of Precision, the average value by Graph4Web achieves improvements by 144.6%, 16.9%, 44.6%, 19.9%, 26.2%, 14.9%, and 3.7% compared with CNN, RCNN, LSTM, Bi-LSTM, C-LSTM, ServeNet, and ServeNet-BERT, individually. Our Graph4Web model obtains the best average Precision value of 0.702 and achieves an average improvement by 38.7%. In terms of Recall, the average value by Graph4Web achieves improvements by 191.8%, 25.2%, 54.8%, 21.5%, 25.5%, 19.4%, and 4.9% compared with the seven representative methods, individually. Our Graph4Web model obtains the best average Recall value of 0.690 and achieves an average improvement by 49.0%. In terms of F-measure, the average value by Graph4Web achieves improvements by 191.7%, 23.5%, 57.0%, 23.4%, 29.4%, 19.2%, and 4.7% compared with the seven representative methods, individually. Our Graph4Web model obtains the best average F-measure value of 0.693 and achieves an average improvement by 49.8%.

To illustrate the results in a more intuitive way, we elaborate a radar map that comprehensively analyzes our Graph4Web model and seven representative methods among all categories via drawing a closed polygonal for F-measure as shown in Fig. 5. From this figure, we can find that, our proposed Graph4Web model (the blue line) appears on the most outside of the radar map and forms the largest closed area, which indicates that Graph4Web always obtains the best performance on most of categories for the web service classification task.

In addition, we give the detailed classification results among all categories in terms of F-measure in Table 2. From this table,

Table 2
The detailed results for F-measure among all categories.

Category	CNN	RCNN	LSTM	Bi-LSTM	C-LSTM	ServeNet	ServeNet-BERT	Graph4Web
Financial	0.450	0.639	0.523	0.578	0.574	0.650	**0.815**	0.796
Enterprise	0.093	0.549	0.390	0.561	0.377	0.571	**0.774**	0.607
Payments	0.268	0.686	0.651	0.689	0.747	0.696	**0.941**	0.787
Internet of Things	0.441	0.828	0.815	0.920	0.820	**0.911**	0.563	0.578
eCommerce	0.563	0.774	0.776	0.750	0.737	**0.785**	0.739	0.761
Mapping	0.000	0.600	0.240	0.400	0.516	0.533	**0.926**	0.811
Telephony	0.169	0.263	0.116	0.269	0.296	0.296	0.700	**0.729**
Shipping	0.512	0.750	0.833	0.880	0.816	**0.898**	0.357	0.836
Video	0.187	0.661	0.469	0.619	0.700	0.660	0.727	**0.852**
Images	0.545	0.690	0.417	0.500	0.560	0.588	**0.780**	0.485
Backend	0.000	0.279	0.167	0.227	0.195	0.255	0.609	**0.630**
News Services	0.423	0.738	0.615	0.767	0.744	**0.795**	0.306	0.667
Data	0.404	**0.796**	0.667	0.791	0.743	0.716	0.476	0.531
Search	0.125	0.682	0.392	0.714	0.619	**0.698**	0.551	0.516
Cloud	0.310	0.604	0.485	0.590	0.539	**0.606**	**0.606**	0.576
Events	0.408	0.733	0.837	0.766	0.765	0.750	0.800	**0.818**
Application Development	0.164	0.575	0.383	**0.587**	0.521	0.508	0.615	0.383
Marketing	0.222	0.376	0.374	0.395	0.447	0.443	0.538	0.625
Photos	0.313	0.852	0.717	**0.840**	0.826	0.917	0.806	0.686
Chat	0.471	0.729	0.690	0.739	0.732	0.752	0.409	**0.757**
Banking	0.276	0.756	0.698	0.703	0.676	0.742	0.559	**0.769**
Domains	0.093	0.000	0.000	0.000	0.000	0.059	0.872	**0.875**
Bitcoin	0.065	0.378	0.140	0.333	0.143	0.391	0.556	**0.793**
Travel	0.000	0.177	0.094	0.107	0.333	0.296	0.661	**0.857**
Other	0.462	0.750	0.111	0.759	0.821	**0.848**	0.450	0.237
Messaging	0.304	0.585	0.409	0.595	0.464	0.594	0.070	**0.789**
Science	0.000	0.345	0.250	0.400	0.214	0.465	0.389	**0.746**
Database	0.303	0.710	0.742	0.753	0.629	**0.779**	0.773	0.473
Stocks	0.056	0.000	0.000	0.186	0.115	0.071	0.764	**0.943**
Games	0.294	0.683	0.486	0.757	0.732	0.737	0.794	**0.824**
Social	0.253	0.826	0.761	0.690	0.761	0.769	**0.783**	0.646
Sports	0.452	0.693	0.722	0.738	0.733	0.735	0.871	**0.925**
Weather	0.143	0.622	0.667	0.713	0.725	0.711	0.833	**0.847**
Real Estate	0.233	0.727	0.473	0.645	0.720	0.581	0.729	**0.837**
Advertising	0.167	0.750	0.485	0.722	0.647	0.683	**0.782**	0.639
Transportation	0.167	0.316	0.000	0.250	0.100	0.167	0.832	**0.841**
Music	0.438	0.700	0.500	0.694	0.727	0.683	0.791	**0.895**
Entertainment	0.211	0.516	0.432	0.489	0.407	0.605	**0.653**	0.595
Media	0.130	0.244	0.150	0.229	0.105	0.244	0.408	**0.564**
Analytics	0.260	0.469	0.406	**0.513**	0.479	0.503	0.444	0.488
Tools	0.067	0.171	0.227	0.465	0.000	0.278	**0.706**	0.568
Security	0.095	0.400	0.143	0.222	0.455	0.308	**0.821**	0.679
Education	0.152	0.429	0.327	0.427	0.481	0.506	0.595	**0.756**
Medical	0.091	0.519	0.385	0.643	0.514	0.545	**0.866**	0.609
Reference	0.061	0.595	0.587	0.693	0.667	0.692	**0.826**	0.362
Storage	0.259	0.667	0.647	0.725	0.737	0.700	**0.913**	0.718
Email	0.557	0.782	0.780	0.786	0.814	0.800	0.480	**0.871**
Project Management	0.000	0.267	0.080	0.095	0.095	0.174	0.629	**0.644**
File Sharing	0.150	0.453	0.300	0.493	0.509	0.554	**0.667**	0.645
Government	0.080	0.741	0.516	0.667	0.710	**0.815**	0.551	0.790
Average	0.238	0.561	0.441	0.562	0.536	0.581	0.662	**0.693**

we can find that CNN produces a strongly negative impact on the performance of web service classification task. By contrast, our Graph4Web model achieves the promising performance improvements among all categories and obtains the best F-measure value on 22 out of 50 categories, especially on some categories like 'Bitcoin' and 'Stocks'. However, we can also see from Table 2 that our Graph4Web method obtains worse performance on some categories than the state-of-the-art model ServeNet-BERT, especially on the category 'Reference'. Here, we try to analyze this phenomenon in terms of the actual meaning of the data and the model itself. From the aspect of categories, such as 'Reference' and 'Bitcoin', the category 'Reference' is a more general expression which means that various web service descriptions may not focus on a smaller application area, whereas another category 'Bitcoin' is more specific and web service descriptions in this category always contain the domain specific words, such as 'currency', 'price', and even 'Bitcoin' itself. Besides, from the aspect of the model architecture, ServeNet-BERT adopting the CNN for feature extraction is benefit to extract the local information and may learn the changes of description details due to the translation and scale invariance of CNN (Yu et al., 2020). In contrast, our Graph4Web based on the GNN model learns the neighborhood relationships and may more suitable for obtaining domain specific information because similar web service descriptions in the same domain may hold similar neighborhood information in their corresponding dependency graphs.

Answer to RQ1

Our proposed Graph4Web model obtains the best performance compared with seven representative methods in terms of three indicators for web service classification task.

6.2. RQ2: How effective is our Graph4Web model compared with its variants?

Methods: To answer this question, we explore the effectiveness of our Graph4Web model from different aspects. More

Table 3
The average results for Graph4Web and its variants.

Method	Precision	Recall	F-measure
Graph4Web$_{Glove}$	0.462	0.404	0.419
BERT$_{only}$	0.681	0.671	0.670
Graph4Web$_{GCN}$	0.691	0.668	0.673
Graph4Web$_{GAT}$	0.682	0.673	0.674
Graph4Web	**0.702**	**0.690**	**0.693**

specifically, we generate the following four variants for comparison by removing and substituting some components used in our model.

- **Graph4Web**$_{Glove}$: This variant utilizes the statical Glove technique to replace the BERT for initializing the node embedding vector, aiming at exploring how BERT impacts the performance of our model.
- **BERT**$_{only}$: This variant only uses the BERT model for web service classification, aiming at investigating the effectiveness of our proposed RAGA layer.
- **Graph4Web**$_{GCN}$: This variant only employs the original graph convolutional neural network (Kipf and Welling, 2016) to update the node embedding and ignores the information originated from the neighborhood nodes and their relationships, aiming at exploring whether this information impacts our model.
- **Graph4Web**$_{GAT}$: This variant removes the different types of relationship information between nodes and only takes into account the neighbors information for updating node embedding, aiming at investigating the influence of relationships.

Results: Table 3 shows the average value of our Graph4Web model and its four variants with three indicators. Fig. 6 depicts the box plot of the five methods among all categories in terms of Precision, Recall, and F-measure, respectively. From the table and figure, we can find that, in terms of Precision, the average value by Graph4Web achieves improvements by 51.9%, 3.2%, 1.7%, and 2.9% compared with Graph4Web$_{Glove}$, BERT$_{only}$, Graph4Web$_{GCN}$, and Graph4Web$_{GAT}$, individually. In terms of Recall, the average value by Graph4Web achieves improvements by 70.9%, 2.8%, 3.4%, and 2.6% compared with its four variants, individually. In terms of F-measure, the average value by Graph4Web achieves improvements by 65.2%, 3.4%, 3.0%, and 2.8% compared with its four variants, individually. Overall, our Graph4Web model achieves average improvements by 14.9%, 19.9%, and 18.6% in terms of Precision, Recall, and F-measure, respectively.

BERT yields performance improvement compared with the statical word embedding technique Glove to a large extent, whereas our proposed RAGA layer taking into account both the neighborhood nodes and relationship information further promotes the classification performance. Compared with Graph4Web$_{GCN}$ and Graph4Web$_{GAT}$, only considering the neighbors information does not seem to affect the model performance, which indicates that the distinct kinds of relationships between nodes possess abundant semantic and syntactic information.

Answer to RQ2

Both the BERT initialization technique and our proposed RAGA layer bring a positive influence on web service classification.

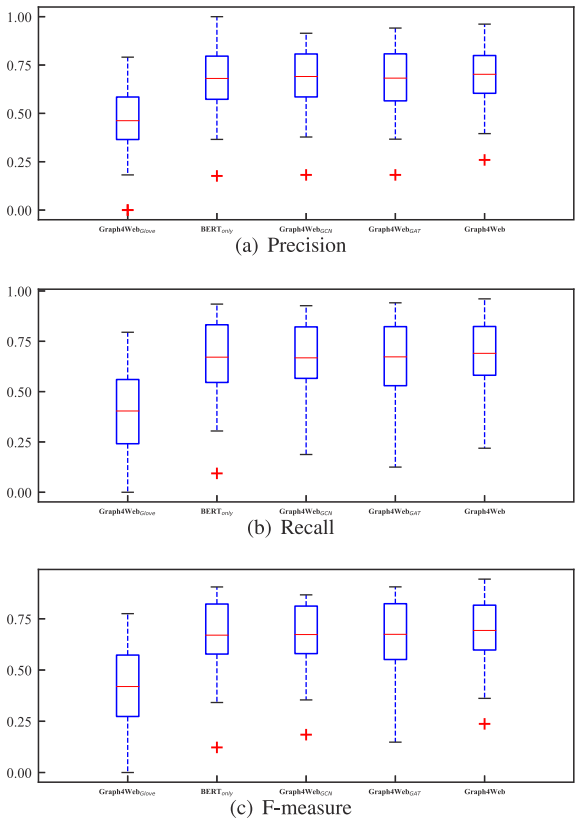

Fig. 6. The box plot of the average value for Graph4Web and its variants among all categories in terms of three indicators.

6.3. RQ3: Can service name improve the performance of our Graph4Web model?

Methods: To answer this question, we use the BERT to tokenize and initialize the name sequence and aggregate the name embedding with the description embedding after updating by the RAGA layers, short for Graph4Web$_{both}$. In addition, we treat the model that only uses the name embedding vectors for classification, short for Graph4Web$_{name}$ as the basic method for comparison.

Results: Table 4 demonstrates the average value of our Graph4Web model with different service information in terms of three indicators, and Fig. 7 illustrates the box plot of these methods among all categories in terms of Precision, Recall, and F-measure, respectively. From the table and figure, we can find that, only using the service name for classification results in poor performance among the three indicators because the name sequence merely specifies this service but the significant semantic information of the web services is always included in its description sequence. By virtue of the service description information, our Graph4Web model obtains average improvements by 65.5%, 70.2%, and 70.1% compared with Graph4Web$_{name}$ in terms of Precision, Recall, and F-measure, respectively. In addition, the model combining both the name and description information (i.e., Graph4Web$_{both}$) does not observe the significant performance improvement but even exhibits slight performance deterioration in terms of Precision and F-measure indicators. As a consequence, due to that encoding the name sequence into embedding vectors will produce extra resource consumption, the service description with sufficient semantic information are adequate to deal with web service classification task.

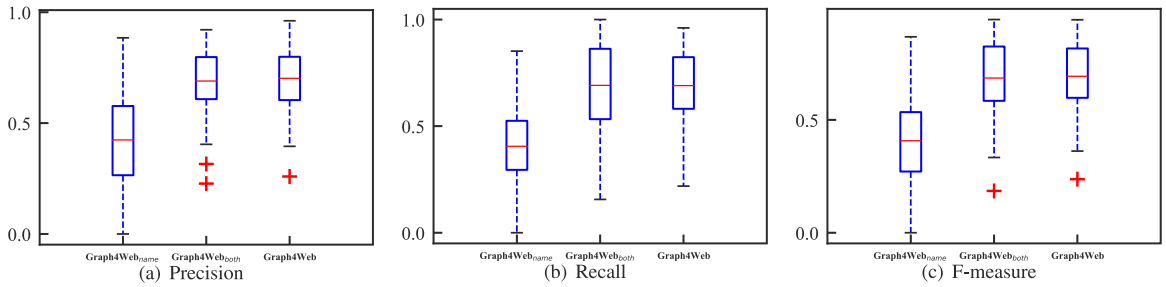

Fig. 7. The box plot of the average value for Graph4Web with different service information among all categories in terms of three indicators.

Table 4
The average results for Graph4Web with different service information.

Method	Precision	Recall	F-measure
Graph4Web$_{name}$	0.424	0.406	0.408
Graph4Web$_{both}$	0.690	**0.691**	0.685
Graph4Web	**0.702**	0.690	**0.693**

Answer to RQ3

Service names are not always required, and modeling informative service descriptions can yield promising performance for web service classification.

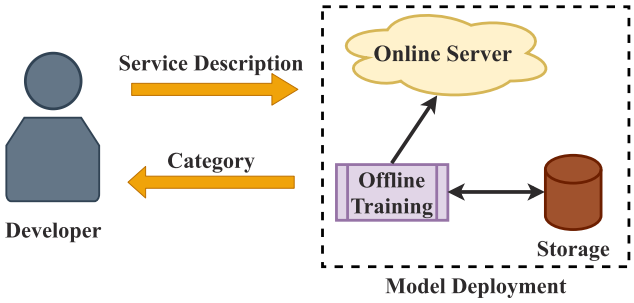

Fig. 8. The application process of our model.

7. Discussion

As an effective way for service discovery, web service classification has attracted much more attention in recent studies (Elgazzar et al., 2010; Elshater et al., 2015; Hao et al., 2010; Liu and Wong, 2009; Yang et al., 2019, 2020). As we mentioned in the motivating example, web service classification can determine which category one web service belongs to, aiming at promoting the maintenance and management process of service brokers in development communities. In addition, the well organized web services make it easier for beginners or developers to retrieve and pick up the suitable one to satisfy their development need. In this work, we propose a novel model, called Graph4Web, to facilitate this process. We briefly explain how can developers use our model. Giving an example as shown in Fig. 8, we can first train our Graph4Web based on the storage data of web services collected from service repositories such as ProgrammableWeb in an offline manner. Once the model training finished, we can upload and deploy our model online and provide an interface to make users and developers call easily. For a developer who wants to issue a new web service, she can invoke the interface of our model deployed on a server by passing through the web service description. Then, the server will return the suitable category to her to accelerate the web service issuing process. Note that, the storage data of web services can be enriched and updated when new web services appearing in the service repositories. By training and deploying the Graph4Web model periodically, developers will obtain more appropriate and precise category prediction results. We follow previous studies only selecting top 50 categories as candidate data to build our classification model because these categories are more common. However, for some emerging categories, our model may predict them as the category 'Other' rather than an exact and specific category. This is a common solution in previous studies (Yang et al., 2019, 2020) because the number of new categories is relatively small. We will explore this issue and find better solutions in the future.

8. Threats to validity

8.1. Threats to internal validity

This kind of threats lies in the potential coding faults during the implementation of our experiments. To eliminate the threats, we implement our Graph4Web model based on PyTorch and the off-the-shelf third-part libraries, such as AllenNLP[4]. For the comparative methods, we carefully modify the source code provided by the previous study (Yang et al., 2020) to satisfy our requirements. In addition, the hyper-parameters tuning is also a threat to internal validity. To reduce the threats, we fine-tune the batch size from {16, 32, 64} and the learning rate from {1e−3, 1e−4, 1e−5}, and select the best parameter settings, i.e., batch size as 32 and learning rate as 1e−5, for our experiments. We stack two RAGA layers and keep them the same during the fine-tuning. Other choices of the number of RAGA layers possibly bring better performance and we leave that for the future work.

8.2. Threats to external validity

This kind of threats focuses on the generalizability of our model. We follow the previous studies (Yang et al., 2019, 2020) and conduct experiments on a web service dataset collected from the ProgrammableWeb site. This is a publicly available dataset, and it can help future researchers replicate our results. We are conscious that it would be better to verify our model with the web services from other repositories (such as the GitHub repository), we leave the exploration as the future work. Besides, we choose two state-of-the-art models for web service classification and five deep learning based methods that have achieved satisfactory performance in previous studies as baseline methods.

4 https://github.com/allenai/allennlp.

More advanced classification models for other similar tasks are also needed to be explored to investigate the superiority of our model.

8.3. Threats to construct validity

This kind of threats fastens on the suitability of the used performance indicators. As the web service classification task can be treated as the multi-classification task, in this work, we employ three indicators, i.e., Precision, Recall, and F-measure, as the evaluation metrics. Another kind of threats to construct validity relates to the practicability of our model. The goal of our work is to determine which category on web service belongs to, aiming at finding the suitable web services from repositories to meet requirements of developers. The experiment results show the superiority of our model, which means that our model is feasible in such task.

9. Conclusion

In this work, we propose a novel model, called Graph4Web, to determine which category one web service belongs to. Graph4Web first encodes the dependent relationship from the web service description sequence into the dependency graph to represent the intrinsic semantic and syntactic information, and then the BERT model is adopted to initialize the embedding vector for each node. We further propose a RAGA layer to learn and update the node embedding in the dependency graph, in which each node is updated by aggregating both its neighborhood nodes and the corresponding different types of relationship information simultaneously. Then we employ the self-attention mechanism to obtain the high-level global representation for web service classification. We have conducted various experiments on the real-world dataset with three performance indicators and the results have shown that our Graph4Web successfully outperformed seven baseline methods.

In the future, we plan to collect more real-world web services to verify the generalization ability of our model. In addition, we will consider the data sparsity issue into the model construction.

CRediT authorship contribution statement

Kunsong Zhao: Writing – original draft, Methodology, Data curation, Software. **Jin Liu:** Supervision, Project administration. **Zhou Xu:** Methodology, Visualization. **Xiao Liu:** Conceptualization, Writing – review & editing. **Lei Xue:** Formal analysis. **Zhiwen Xie:** Software. **Yuxuan Zhou:** Writing – review & editing. **Xin Wang:** Writing – review & editing.

Declaration of competing interest

The authors declare that they have no known competing financial interests or personal relationships that could have appeared to influence the work reported in this paper.

Acknowledgments

This work was supported by the National Natural Science Foundation of China under Grants (No. 61972290), the National Natural Science Foundation of China (No. 62102054), the Natural Science Foundation of Chongqing in China (No. cstc2020jcyj-bshX0114), and the Key Laboratory of Dependable Service Computing in Cyber-Physical-Society (Ministry of Education), Chongqing University (CPSDSC202004).

References

Atkinson, C., Bostan, P., Hummel, O., Stoll, D., 2007. A practical approach to web service discovery and retrieval. In: IEEE International Conference on Web Services. ICWS, IEEE, pp. 241–248.

Aznag, M., Quafafou, M., Jarir, Z., 2014. Leveraging formal concept analysis with topic correlation for service clustering and discovery. In: 2014 IEEE International Conference on Web Services. ICWS, IEEE, pp. 153–160.

Barros-Justo, J.L., Pinciroli, F., Matalonga, S., Martínez-Araujo, N., 2018. What software reuse benefits have been transferred to the industry? A systematic mapping study. Inf. Softw. Technol. (IST) 103, 1–21.

Buckley, F.J., Poston, R., 1984. Software quality assurance. IEEE Trans. Softw. Eng. (TSE) (1), 36–41.

Busbridge, D., Sherburn, D., Cavallo, P., Hammerla, N.Y., 2019. Relational graph attention networks. arXiv preprint arXiv:1904.05811.

Cao, Y., Liu, J., Cao, B., Shi, M., Wen, Y., Peng, Z., 2019. Web services classification with topical attention based bi-lstm. In: International Conference on Collaborative Computing: Networking, Applications and Worksharing. Springer, pp. 394–407.

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

Dozat, T., Manning, C.D., 2016. Deep biaffine attention for neural dependency parsing. arXiv preprint arXiv:1611.01734.

Elgazzar, K., Hassan, A.E., Martin, P., 2010. Clustering wsdl documents to bootstrap the discovery of web services. In: 2010 IEEE International Conference on Web Services. ICWS, IEEE, pp. 147–154.

Elshater, Y., Elgazzar, K., Martin, P., 2015. Godiscovery: Web service discovery made efficient. In: 2015 IEEE International Conference on Web Services. ICWS, IEEE, pp. 711–716.

Fang, L., Wang, L., Li, M., Zhao, J., Zou, Y., Shao, L., 2012. Towards automatic tagging for web services. In: 2012 IEEE 19th International Conference on Web Services. ICWS, IEEE, pp. 528–535.

Ferenc, R., Gyimesi, P., Gyimesi, G., Tóth, Z., Gyimóthy, T., 2020. An automatically created novel bug dataset and its validation in bug prediction. J. Syst. Softw. (JSS) 169, 110691.

Gu, Y., Xuan, J., Zhang, H., Zhang, L., Fan, Q., Xie, X., Qian, T., 2019. Does the fault reside in a stack trace? assisting crash localization by predicting crashing fault residence. J. Syst. Softw. (JSS) 148, 88–104.

Guo, Z., Zhang, Y., Lu, W., 2019. Attention guided graph convolutional networks for relation extraction. arXiv preprint arXiv:1906.07510.

Hajlaoui, J.E., Omri, M.N., Benslimane, D., Barhamgi, M., 2017. Qos based framework for configurable IaaS cloud services discovery. In: 2017 IEEE International Conference on Web Services. ICWS, IEEE, pp. 460–467.

Hao, Y., Zhang, Y., Cao, J., 2010. Web services discovery and rank: An information retrieval approach. Future Gener. Comput. Syst. 26 (8), 1053–1062.

Huang, L., Ma, D., Li, S., Zhang, X., Wang, H., 2019. Text level graph neural network for text classification. arXiv preprint arXiv:1910.02356.

Imoize, A.L., Idowu, D., Bolaji, T., 2019. A brief overview of software reuse and metrics in software engineering. World Sci. News 122, 56–70.

Johnson, R., Zhang, T., 2016. Supervised and semi-supervised text categorization using LSTM for region embeddings. In: International Conference on Machine Learning. PMLR, pp. 526–534.

Kapitsaki, G.M., 2014. Annotating web service sections with combined classification. In: 2014 IEEE International Conference on Web Services. ICWS, IEEE, pp. 622–629.

Katakis, I., Meditskos, G., Tsoumakas, G., Bassiliades, N., et al., 2009. On the combination of textual and semantic descriptions for automated semantic web service classification. In: IFIP International Conference on Artificial Intelligence Applications and Innovations. Springer, pp. 95–104.

Kipf, T.N., Welling, M., 2016. Semi-supervised classification with graph convolutional networks. arXiv preprint arXiv:1609.02907.

Kowsari, K., Brown, D.E., Heidarysafa, M., Meimandi, K.J., Gerber, M.S., Barnes, L.E., 2017. Hdltex: Hierarchical deep learning for text classification. In: 2017 16th IEEE International Conference on Machine Learning and Applications. ICMLA, IEEE, pp. 364–371.

Lai, S., Xu, L., Liu, K., Zhao, J., 2015. Recurrent convolutional neural networks for text classification. In: Twenty-Ninth AAAI Conference on Artificial Intelligence.

Liu, X., Agarwal, S., Ding, C., Yu, Q., 2016a. An LDA-svm active learning framework for web service classification. In: 2016 IEEE International Conference on Web Services. ICWS, IEEE, pp. 49–56.

Liu, X., Fulia, I., 2015. Incorporating user, topic, and service related latent factors into web service recommendation. In: 2015 IEEE International Conference on Web Services. ICWS, IEEE, pp. 185–192.

Liu, J., Tian, Z., Liu, P., Jiang, J., Li, Z., 2016b. An approach of semantic web service classification based on naive Bayes. In: 2016 IEEE International Conference on Services Computing. SCC, IEEE, pp. 356–362.

Liu, W., Wong, W., 2009. Web service clustering using text mining techniques. Int. J. Agent-Orient. Softw. Eng. 3 (1), 6–26.

Maas, A.L., Hannun, A.Y., Ng, A.Y., et al., 2013. Rectifier nonlinearities improve neural network acoustic models. In: Proc. Icml, Vol. 30, no. 1. Citeseer, p. 3.
Minaee, S., Kalchbrenner, N., Cambria, E., Nikzad, N., Chenaghlu, M., Gao, J., 2021. Deep learning–based text classification: A comprehensive review. ACM Comput. Surv. 54 (3), 1–40.
Nisa, R., Qamar, U., 2015. A text mining based approach for web service classification. Inf. Syst. E-Bus. Manag. 13 (4), 751–768.
Pascarella, L., Palomba, F., Bacchelli, A., 2019. Fine-grained just-in-time defect prediction. J. Syst. Softw. (JSS) 150, 22–36.
Pennington, J., Socher, R., Manning, C.D., 2014. Glove: Global vectors for word representation. In: Proceedings of the 2014 Conference on Empirical Methods in Natural Language Processing. EMNLP, pp. 1532–1543.
Ren, X., Xing, Z., Xia, X., Lo, D., Wang, X., Grundy, J., 2019. Neural network-based detection of self-admitted technical debt: From performance to explainability. ACM Trans. Softw. Eng. Methodol. (TOSEM) 28 (3), 1–45.
Shaw, P., Uszkoreit, J., Vaswani, A., 2018. Self-attention with relative position representations. arXiv preprint arXiv:1803.02155.
Shi, W., Liu, X., Yu, Q., 2017. Correlation-aware multi-label active learning for web service tag recommendation. In: 2017 IEEE International Conference on Web Services. ICWS, IEEE, pp. 229–236.
Tang, B., Yan, M., Zhang, N., Xu, L., Zhang, X., Ren, H., 2021. Co-attentive representation learning for web services classification. Expert Syst. Appl. 180, 115070.
Tang, X., Zhang, J., Chen, B., Yang, Y., Chen, H., Li, C., 2020. BERT-INT: A BERT-based interaction model for knowledge graph alignment. In: IJCAI. pp. 3174–3180.
Tian, Y., Chen, G., Song, Y., Wan, X., 2021. Dependency-driven relation extraction with attentive graph convolutional networks. 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). pp. 4458–4471.
Vashishth, S., Bhandari, M., Yadav, P., Rai, P., Bhattacharyya, C., Talukdar, P., 2018. Incorporating syntactic and semantic information in word embeddings using graph convolutional networks. arXiv preprint arXiv:1809.04283.
Vaswani, A., Shazeer, N., Parmar, N., Uszkoreit, J., Jones, L., Gomez, A.N., Kaiser, Ł., Polosukhin, I., 2017. Attention is all you need. In: Advances in Neural Information Processing Systems. pp. 5998–6008.
Veličković, P., Cucurull, G., Casanova, A., Romero, A., Lio, P., Bengio, Y., 2017. Graph attention networks. arXiv preprint arXiv:1710.10903.
Wang, H., Chen, X., Wu, Q., Yu, Q., Hu, X., Zheng, Z., Bouguettaya, A., 2017. Integrating reinforcement learning with multi-agent techniques for adaptive service composition. ACM Trans. Auton. Adapt. Syst. (TAAS) 12 (2), 1–42.
Wang, S., Huang, M., Deng, Z., et al., 2018. Densely connected CNN with multi-scale feature attention for text classification.. In: IJCAI. pp. 4468–4474.
Wang, X., Liu, J., Li, L., Chen, X., Liu, X., Wu, H., 2020a. Detecting and explaining self-admitted technical debts with attention-based neural networks. In: Proceedings of the 35th IEEE/ACM International Conference on Automated Software Engineering. ASE, pp. 871–882.
Wang, X., Liu, X., Liu, J., Chen, X., Wu, H., 2021a. A novel knowledge graph embedding based api recommendation method for mashup development. World Wide Web 24 (3), 869–894.
Wang, X., Liu, J., Liu, X., Cui, X., Wu, H., 2020b. A novel dual-graph convolutional network based web service classification framework. In: 2020 IEEE International Conference on Web Services. ICWS, IEEE, pp. 281–288.
Wang, X., Liu, X., Liu, J., Wu, H., 2021b. Relational graph neural network with neighbor interactions for bundle recommendation service. In: 2021 IEEE International Conference on Web Services (ICWS). IEEE, pp. 167–172.
Wang, K., Shen, W., Yang, Y., Quan, X., Wang, R., 2020c. Relational graph attention network for aspect-based sentiment analysis. arXiv preprint arXiv:2004.12362.
Wang, H., Shi, Y., Zhou, X., Zhou, Q., Shao, S., Bouguettaya, A., 2010. Web service classification using support vector machine. In: 2010 22nd IEEE International Conference on Tools with Artificial Intelligence, Vol. 1. IEEE, pp. 3–6.
Wang, H., Xia, X., Lo, D., He, Q., Wang, X., Grundy, J., 2021c. Context-aware retrieval-based deep commit message generation. ACM Trans. Softw. Eng. Methodol. (TOSEM) 30 (4), 1–30.
Xie, Z., Zhou, G., Liu, J., Huang, X., 2020a. Reinceptione: Relation-aware inception network with joint local-global structural information for knowledge graph embedding. In: Proceedings of the 58th Annual Meeting of the Association for Computational Linguistics. ACL, pp. 5929–5939.
Xie, Z., Zhu, R., Zhao, K., Liu, J., Zhou, G., Huang, X., 2020b. A contextual alignment enhanced cross graph attention network for cross-lingual entity alignment. In: Proceedings of the 28th International Conference on Computational Linguistics. COLING, pp. 5918–5928.
Xu, Z., Li, S., Luo, X., Liu, J., Zhang, T., Tang, Y., Xu, J., Yuan, P., Keung, J., 2019a. TSTSS: A two-stage training subset selection framework for cross version defect prediction. J. Syst. Softw. (JSS) 154, 59–78.
Xu, Z., Li, S., Xu, J., Liu, J., Luo, X., Zhang, Y., Zhang, T., Keung, J., Tang, Y., 2019b. LDFR: Learning deep feature representation for software defect prediction. J. Syst. Softw. (JSS) 158, 110402.
Xu, Z., Zhao, K., Zhang, T., Fu, C., Yan, M., Xie, Z., Zhang, X., Catolino, G., 2021. Effort-aware just-in-time bug prediction for mobile apps via cross-triplet deep feature embedding. IEEE Trans. Reliab..
Yang, Y., Ke, W., Wang, W., Zhao, Y., 2019. Deep learning for web services classification. In: 2019 IEEE International Conference on Web Services. ICWS, IEEE, pp. 440–442.
Yang, Y., Li, X., Qamar, N., Liu, P., Ke, W., Shen, B., Liu, Z., 2018a. Medshare: a novel hybrid cloud for medical resource sharing among autonomous healthcare providers. IEEE Access 6, 46949–46961.
Yang, Y., Qamar, N., Liu, P., Grolinger, K., Wang, W., Li, Z., Liao, Z., 2020. ServeNet: A deep neural network for web services classification. In: 2020 IEEE International Conference on Web Services. ICWS, IEEE, pp. 168–175.
Yang, Y., Zu, Q., Liu, P., Ouyang, D., Li, X., 2018b. MicroShare: Privacy-preserved medical resource sharing through microservice architecture. Int. J. Biol. Sci. 14 (8), 907.
Yao, L., Mao, C., Luo, Y., 2019. Graph convolutional networks for text classification. In: Proceedings of the AAAI Conference on Artificial Intelligence, Vol. 33, no. 01. pp. 7370–7377.
Ye, H., Cao, B., Peng, Z., Chen, T., Wen, Y., Liu, J., 2019. Web services classification based on wide & Bi-LSTM model. IEEE Access 7, 43697–43706.
Yu, Z., Cao, R., Tang, Q., Nie, S., Huang, J., Wu, S., 2020. Order matters: Semantic-aware neural networks for binary code similarity detection. In: Proceedings of the AAAI Conference on Artificial Intelligence, Vol. 34, no. 01. pp. 1145–1152.
Yu, X., Liu, J., Keung, J.W., Li, Q., Bennin, K.E., Xu, Z., Wang, J., Cui, X., 2019. Improving ranking-oriented defect prediction using a cost-sensitive ranking svm. IEEE Transactions on Reliability 69 (1), 139–153.
Yu, J., Zhao, K., Liu, J., Liu, X., Xu, Z., Wang, X., 2022. Exploiting gated graph neural network for detecting and explaining self-admitted technical debts. Journal of Systems and Software 111219.
Zhang, T., Jiang, H., Luo, X., Chan, A.T., 2016. A literature review of research in bug resolution: Tasks, challenges and future directions. Comput. J. 59 (5), 741–773.
Zhang, S., Zheng, D., Hu, X., Yang, M., 2015. Bidirectional long short-term memory networks for relation classification. In: Proceedings of the 29th Pacific Asia Conference on Language, Information and Computation. pp. 73–78.
Zhao, K., Liu, J., Xu, Z., Li, L., Yan, M., Yu, J., Zhou, Y., 2021a. Predicting crash fault residence via simplified deep forest based on a reduced feature set. In: 2021 IEEE/ACM 29th International Conference on Program Comprehension (ICPC). IEEE, pp. 242–252.
Zhao, K., Xu, Z., Yan, M., Zhang, T., Yang, D., Li, W., 2021b. A comprehensive investigation of the impact of feature selection techniques on crashing fault residence prediction models. Information and Software Technology 139, 106652.
Zhao, K., Xu, Z., Zhang, T., Tang, Y., Yan, M., 2021c. Simplified deep forest model based just-in-time defect prediction for android mobile apps. IEEE Transactions on Reliability 70 (2), 848–859.
Zhou, C., Sun, C., Liu, Z., Lau, F., 2015. A C-LSTM neural network for text classification. arXiv preprint arXiv:1511.08630.

Kunsong Zhao is currently a master student at the School of Computer Science, Wuhan University, China. He received the B.S. degree in software engineering from School of Computer Science and Information Engineering, Hubei University in 2019. His current research interests include software engineering and natural language processing.

Jin Liu received the Ph.D. degree in computer science from the State Key Lab of Software Engineering, Wuhan University, China, in 2005. He is currently a professor at School of Computer Science, Wuhan University. He has authored or coauthored a number of research papers in international journals. His research interests include mining software repositories and intelligent information processing.

Zhou Xu is an assistant professor in the School of Big Data and Software Engineering at Chongqing University, China. He received two Ph.D. degrees from Wuhan University (Wuhan, China) and The Hong Kong Polytechnic University (Hong Kong, China) in 2019 and 2021, respectively. His research interests include software defect prediction, empirical software engineering, feature engineering, and data mining.

Xiao Liu received his Ph.D. degree in computer science and software engineering from the Faculty of Information and Communication Technologies, Swinburne University of Technology, Melbourne, Australia, in 2011. He was an associate professor at the Software Engineering Institute, East China Normal University, Shanghai, China during 2013 to 2015. He is currently an associate professor with the School of Information Technology, Deakin University, Melbourne. His

research interests include workflow systems, cloud and edge computing, big data analytics, and human-centric software engineering.

Lei Xue is a Research Assistant Professor with the Department of Computing, The Hong Kong Polytechnic University. He also received the Ph.D. degree in Computer Science from The Hong Kong Polytechnic University. His current research focuses on mobile security and privacy, program analysis, and automotive security.

Zhiwen Xie received the B.S. degree and master's degree from Central China Normal University, Wuhan, China. He is currently pursuing Ph.D. degree at the School of Computer Science, Wuhan University. His research interests include natural language processing, knowledge graph embedding, and text mining.

Yuxuan Zhou is currently a Ph.D. candidate in Computer and Information Science and Engineering (CISE) at Syracuse University. He received his master's degree in computer science from Syracuse University in 2020 and a BS degree in Software Engineering from Hubei University in 2019. His research interests lie in security evaluation, data analysis, vulnerability detection and mitigation, with a current focus on Blockchain.

Xin Wang received the M.S degree in computer technology from Yunnan University in 2019. He is currently a Ph.D. candidate in computer science at Wuhan University. He has co-authored more than 10 papers, and published in top venues such as ASE, ICWS, WWWJ. His current research interests mainly include service computing, software engineering and recommender systems.