IEEE Access
Multidisciplinary | Rapid Review | Open Access Journal

Received February 8, 2021, accepted March 22, 2021, date of publication March 29, 2021, date of current version April 13, 2021.

Digital Object Identifier 10.1109/ACCESS.2021.3069509

Semantic Service Clustering With Lightweight BERT-Based Service Embedding Using Invocation Sequences

KUNGAN ZENG, (Member, IEEE), AND INCHEON PAIK, (Senior Member, IEEE)
School of Computer Science and Engineering, The University of Aizu, Aizuwakamatsu 965-8580, Japan
Corresponding author: Incheon Paik (paikic@u-aizu.ac.jp)

ABSTRACT Service clustering is an efficient method for facilitating service discovery and composition. Traditional approaches based on the self-description documents for services usually utilize service signatures. In Web service composition, service clustering can also be performed by the invocation relationship between services. Therefore, based on the successful development of several embedding techniques for words in several contexts, a novel deep learning-based service embedding using invocation sequences is devised for service clustering. Moreover, many microservices are being created because of the rapid development of the Internet of Things (IoT), and edge, and fog computing. Following these developments, Web service composition based on these environments has emerged in abundance. More efficient lightweight approaches to analyze large numbers of services are necessary for service clustering. Consequently, a lightweight deep learning-based approach for the semantic clustering of service composition is presented to address these requirements. In this paper, we first propose the concept of service embedding to capture semantic information from invocation sequences. Second, we suggest using state-of-the-art neural language sequence models for service embedding and develop a corresponding lightweight Bidirectional Encoder Representations of Transformers (BERT)-based model. Next, combined with K-means clustering, the semantic clustering of service composition is evaluated. Finally, the experimental results show that the clustering process can be effectively performed by the lightweight BERT-based model.

INDEX TERMS Semantic service clustering, service embedding, composition, lightweight BERT.

I. INTRODUCTION

Web services can implement interoperations between different software applications over the network. These implementation mainly rely on some standard technologies, such as Extensible Markup Language (XML), Web Service Description Language (WSDL), Simple Object Access Protocol (SOAP), and Universal Description, Discovery and Integration (UDDI). Web services are widely used in e-business and are becoming increasingly popular with application developers. With the dramatic increase in services, it has been a problem for consumers to obtain ideal services, thus limiting the development of Web services. To overcome this limitation, Web service discovery plays an important role.

Web service discovery aims to match the request of customers to corresponding services. Before the matching process, clustering relevant services according to their domains or features is an efficient way to boost service discovery or service composition [1], [2]. Fig. 1 shows the three main steps: *Requirement Analysis*, *Feature Extraction*, and *Matcher*. *Requirement Analysis* helps understand the requests of consumers and passes the expressions to *Matcher*. *Feature Extraction* can represent services with some formatted data that are understandable for computers. *Matcher* identifies target services based on the expression. If a single service cannot meet the complex requirements of consumers, service composition is introduced. In traditional service clustering, WSDL-based approaches, such as keywords, word embedding, Latent Dirichlet Allocation (LDA), and ontology are used to extract features from WSDL documents [3]–[5], and then relevant services are clustered by computing these features. These approaches usually involve service signatures in service representation, such as the names of operations and Inputs, Outputs, Preconditions, and Effects (IOPEs).

The associate editor coordinating the review of this manuscript and approving it for publication was Zhangbing Zhou.

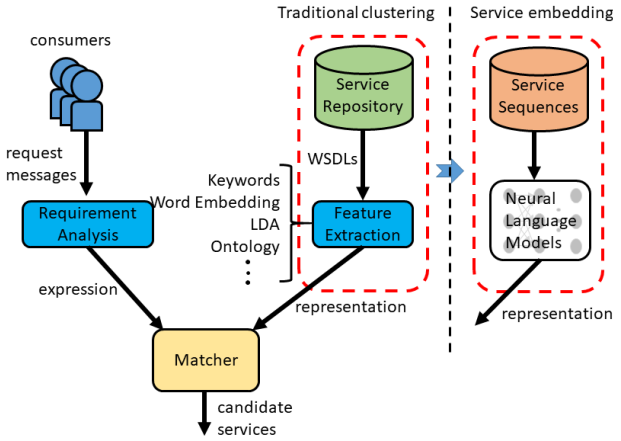


FIGURE 1. Service clustering with service embedding in Web service discovery.

As another approach, the invocation association between services can be considered for service clustering. This approach reflects the real invocation situation on service execution. Therefore, in this study, based on the successful development of several embedding techniques for words in several contexts, a novel service embedding method can be devised for realistic service clustering. In addition, because of the rapid development of IoT, and edge, and fog computing, many microservices in the environment are being created for the environment [6]–[8]. This leads to better quality service compositions, such as the acceleration of efficient mashup development [9], [10]. Moreover, service composition has been frequently implemented on clouding and edge computing environments. [11]–[15]. In these environments, however, there are not enough resources to support large-scale deep learning models. Optimizing and quantizing model weights can reduce resource costs [16]; thus, more efficient lightweight approaches to analyze large amounts of services are necessary for service clustering.

Recently, deep learning approaches in learning sequential data have greatly improved, such as long short-term memory (LSTM) and BERT. BERT shows the best state-of-the-art performance, but it is immature and too heavy. In this paper, a lightweight deep learning-based approach to perform service clustering is proposed. This approach performs semantic clustering of service composition with a lightweight BERT-based service embedding model that uses a novel transformer's encoder. First, we propose service embedding to build an informative cyclic framework in Web service composition that uses neural language networks to learn service composition sequences. The model can well understand the invocation relationship between services and extract semantic information from these sequences. Second, the pre-trained model can generate the representation vectors of all sequences. The general meaning can be illustrated in the right part of Fig. 1 and entirely uses deep learning methods to implement the representation of services. Then, we cluster these representation vectors to obtain different semantic clusters. The differences between the present and traditional approaches are shown in Table 1. Our contributions can be summarized as follows:

TABLE 1. Comparison of Approaches.

Approach	Source Data	Features Extraction	Construction
Traditional approaches	WSDL documents	Keywords, LDA, Ontology etc.	Integrated system
Deep learning-based	Service sequences	Service embedding	Unsupervised learning model

- We propose service embedding to construct an informative cyclic framework in service composition and suggest using neural language models (LMs) to perform service embedding.
- Considering the complexity of the existing model, we develop a lightweight deep neural LM in our approach. Compared with the base model, the lightweight model has similar performance, but is faster.
- A deep learning-based approach is proposed to perform semantic clustering of service composition based on service embedding. We then perform comprehensive experiments with a real-world dataset. The results show that our approach can implement the clustering effectively.

The remainder of this paper is organized as follows. Section II addresses related works. Section III introduces service embedding. Section IV illustrates neural LMs and proposes two service embedding models. Section V presents the whole semantic discovery of the Web service composition framework. Section VI describes the data preparation. Section VII demonstrates and discusses the experiments. Section VIII concludes the paper.

II. RELATED WORK

To the best of our knowledge, this is the first work to develop an entirely deep learning-based approach for service clustering. Thus, the related works are described based on several aspects.

A. WEB SERVICE CLUSTERING

Web service clustering considers related services as the same category based on the features. This approach commonly captures features from WSDL documents and computes the similarity of features between different services [17]–[21]. Wu *et al.* [1] suggested clustering Web services through both WSDL documents and tags, and Kumara *et al.* [5] presented computing the similarity of features with ontology learning. Shi *et al.* [3] presented a word embedding augmented LDA model for service clustering. Zou *et al.* [4] proposed clustering services via integrating service composability into deep semantic features. Differing from computing the similarity between services based on the service description documents, we propose using neural LMs to represent services with representation vectors and perform clustering.

B. SEMANTIC WEB SERVICE DISCOVERY

WSDL documents are a type of standard metadata that is very difficult for machine algorithms to understand from a semantic aspect. Therefore, semantic Web service discovery has been presented to address the problem. Several methods have been proposed to reconstruct service description for enriching Web services with machine-processable semantics, such as Web Ontology Language for Web service [22], Web service modeling ontology [23], and semantic annotations for the Web services description language [24], [25]. Ontology shows promising potential [26], [27]. Instead of extracting semantic knowledge from WSDL documents or constructing the ontology of services based on WSDL documents, we attempt to reveal semantic information from service composition sequences because the invocation relationship between services contains semantic information of services.

C. SOCIAL RELATIONSHIP FOR WEB SERVICE DISCOVERY

The social relationship between services corresponds to a type of association mechanism. It connects relevant services in a certain way, enabling service discovery or service composition. Such a connection is commonly constructed based on the relationships between services, such as functionality, quality of service, or sociability, and is considered a promising approach to discover target services [28]. Zakaria *et al.* [29] developed social networks for Web service discovery. Chen *et al.* [30] presented the Global Social Service Network to connect distributed services. Corbellini *et al.* [31] proposed mining social Web service repositories for social relationships to aid service discovery. Instead of constructing a social network, we adopt the neural network to learn service composition sequences and extract the invocation relationship for clustering services.

D. DEEP LEARNING FOR APPLICATION PROGRAMMING INTERFACE (API) LEARNING

To alleviate the burdens on developers, deep learning is applied to API learning. Gu *et al.* [32] presented a neural LM to learn the projection from a natural language query to an API usage sequence. Bhupatiraju *et al.* [33] proposed using the learning program with APIs in a novel neural synthesis algorithm. Wu *et al.* [34] proposed automatically finding answers for API-related natural language questions from tutorials and stack overflow. These cases demonstrate that the neural language network can understand not only natural language sequences, but also API usage sequences. These works inspire us to utilize neural LMs to learn service composition sequences and enable them to extract some important information for service clustering.

First, various traditional service clustering approaches are reviewed in this section. These approaches are usually based on WSDL documents. Then, some existing studies that perform service discovery based on the social relationship between services are reviewed. However, these studies did not consider the invocation relationship. Finally, some cases of deep learning for API learning are reviewed and indicate that deep learning models can understand API invocation sequences well. These studies inspire us to propose a new approach that performs service clustering with service embedding using invocation sequences.

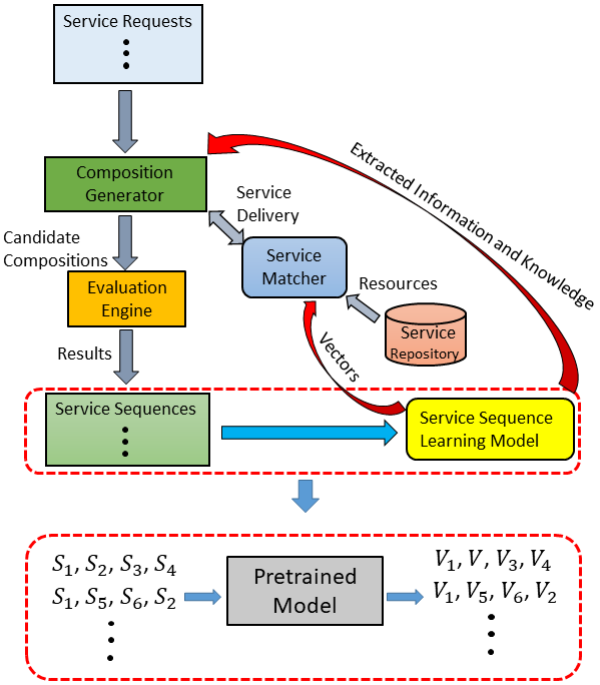


FIGURE 2. Service embedding in Web service composition.

III. SERVICE EMBEDDING

In this section, the concept of service embedding is proposed in Web service composition. Web service discovery can provide suitable services for consumers. However, when a single service cannot meet the complex requirements of consumers, the discovery task changes to service composition by combining several services and providing value-added services. Fig. 2 shows the Web service composition. The main components are *Service Matcher*, *Composition Generator* and *Evaluation Engine*. When *Composition Generator* receives service requests from customers, it needs to process the requests and gain relevant services from *Service Matcher* to composite these relevant services and then send candidate service compositions to *Evaluation Engine* for the test. The final tested service composition provides value-added services that can satisfy the complex functionality required by consumers. *Composition Generator* generates service compositions based on some rules or knowledge, meaning that these service sequences contain the invocation relationship. Determining exact information or knowledge is very helpful in service clustering. In other words, we can perform service clustering based on the invocation relationship. Thus, as shown in the bottom part of Fig. 2, we present service embedding in the framework to learn the service composition sequences by suitable models. Then, the sequences can be

projected into representative vectors by the pretrained models. We can determine related services by computing these representative vectors. The significance of service embedding can be concluded as follows:

- The representative vectors generated by the pretrained model can be used to find relevant services.
- The extracted information and knowledge can be used to contribute to the service composition procedure.
- The model is independent and open in the cyclic framework because the input and output are service composition sequences and representative vectors, respectively. Such kinds of data are easy to share and exploit.

IV. SERVICE EMBEDDING WITH DEEP NEURAL LANGUAGE NETWORKS

Transformer is a state-of-the-art model in neural machine translation. BERT is the stacked layers of Transformer's encoder. In this paper, BERT is used to service embedding. However, the base model is too heavy and immature. Therefore, a lightweight BERT-based mode is also developed for service embedding. This section gives a description of two models in detail.

A. TRANSFORMER AND BERT

LMs play a vital role in natural language processing (NLP), such as machine translation, questioning and answering, and sentiment analysis. LMs are required to represent a word sequence with the form that is understandable to the machine and estimate the probability distribution of words, phrases, and sentences. For a language sequence $(w_1, w_2, \ldots, w_n)$, LMs need to calculate the probability distribution of this sequence, namely, $P(w_1, w_2, \ldots, w_n)$. LMs can be divided into two categories: count-based LMs and continuous-space LMs. Count-based LMs usually refer to the traditional statistical models. As mentioned previously, when we try to compute the probability of a sequence such as $P(w_1, w_2, \ldots, w_n)$, we use the chain rule of probability to obtain (1).

$$P(w_1, w_2, \ldots, w_n) = P(w_1)P(w_2|w_1)\cdots P(w_n|w_1^{n-1}) \quad (1)$$

Neural LMs use neural networks to learn the probability, and there has recently been great improvement. Especially, transformer and its stacked layers construction BERT demonstrated excellent capacity for learning language sequences [35], [36]. As shown in Fig. 3, Transformer relies entirely on a self-attention mechanism and consists of the *Encoder* and *Decoder*. The main components are *Multi-head Attention*, *Feed Forward*, and *Add & Norm*. *Feed Forward* consists of two linear transformations with a Rectified Linear Unit activation function in between. *Add & Norm* is residual connection [37] and layer normalization [38]. *Multi-head Attention* is the crucial part that realizes a self-attention mechanism and is shown in Fig. 4. It consists of several attention layers running in parallel. h represents the number of heads or the parallel layers. The input vectors query (Q), keys (K), and values (V) are transferred to *Scaled Dot-Product Attention* through linear projections. In a

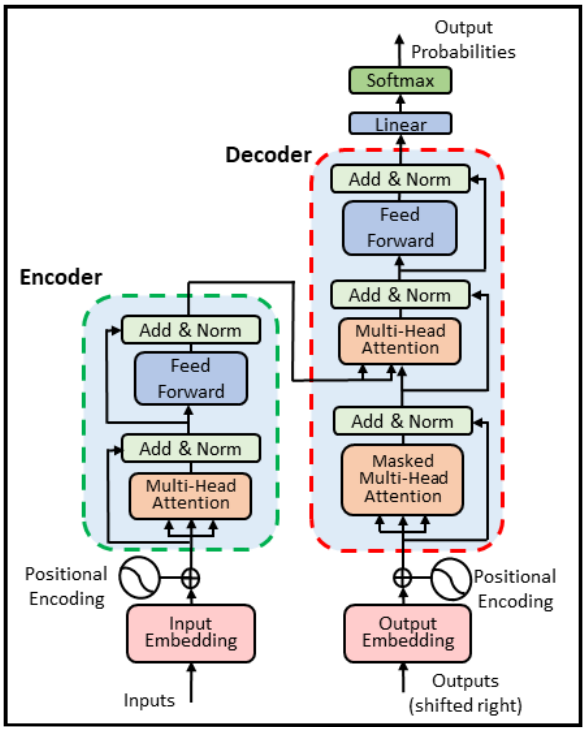


FIGURE 3. Transformer architecture.

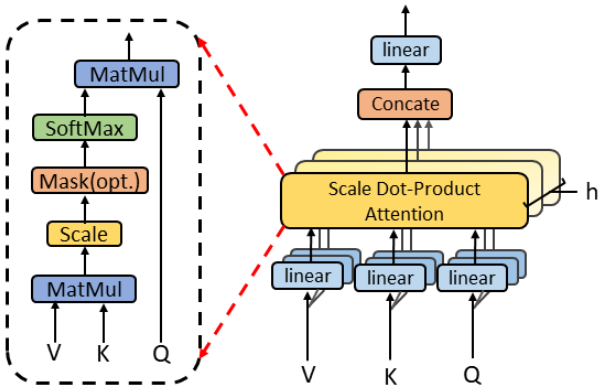


FIGURE 4. Multi-head attention.

self-attention layer, all queries, keys, and values come from the same place. The *Scaled Dot-Product Attention* can be formulated as:

$$Attention(Q, K, V) = softmax(\frac{QK^T}{\sqrt{d_k}})V. \quad (2)$$

The mask operation is used in the *Decoder* part to make the current position observe only its previous positions. All the attention weights are concatenated and transformed with a linear projection.

BERT consists of the stacked layers of the transformer encoder, as shown in Fig. 5. The proposal of BERT divides the NLP procedure into two phases: upstream representation and downstream tasks. BERT is used in the upstream representation. There are two unsupervised tasks to pretrain BERT:

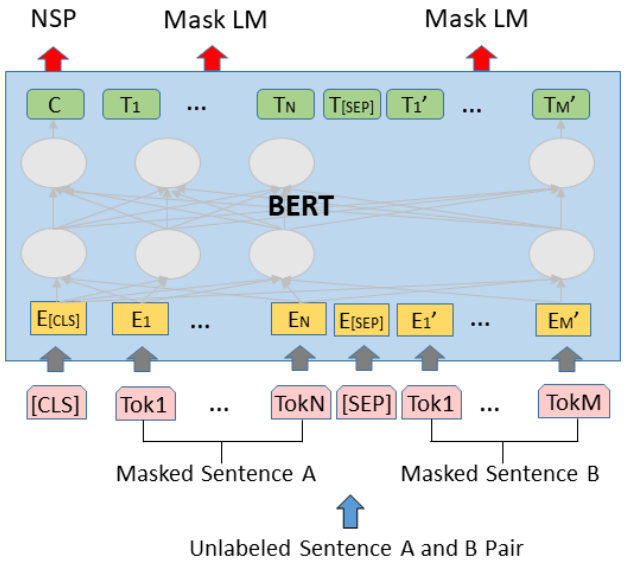


FIGURE 5. BERT architecture.

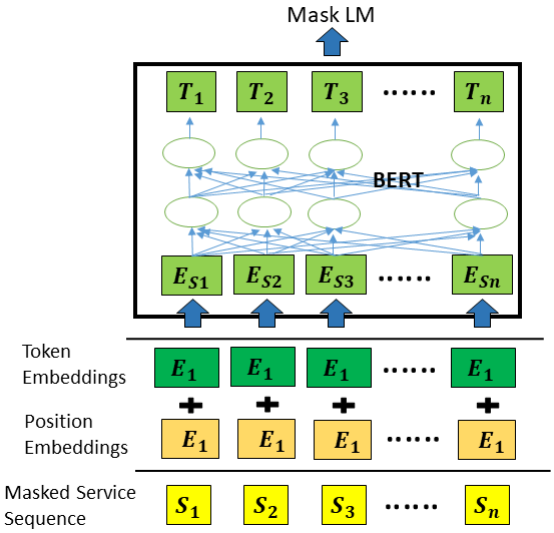


FIGURE 6. BERT-based service embedding model.

next sentence prediction (NSP) and masked LMs. As shown in Fig. 5, the input is the concatenation of two masked sentences, and the first position is [CLS]. NSP requires the model to predict whether the second sentence is the next sentence of the first sentence. The corresponding output position is probability. [SEP] is a special separator token of two sentences (e.g., separating questions/answers). Masked LMs predict the masked token in the input sentences. The pretrained BERT can be used in multiple downstream tasks. BERT has demonstrated good performance in machine translation, Q & A systems, and so on. The self-attention mechanism can learn an excellent representation of the input sequences through unsupervised learning. In this research, we propose using BERT to learn service composition sequences and capture the invocation relationship with its self-attention mechanism. In our case, some details have been accordingly adjusted in the model. The NSP task and the segment embedding are removed, as shown in Fig. 6. The input is a single masked service sequence, and the embedding layer consists of two processes: *Token Embeddings* and *Position Embeddings*. The model performs masked LMs. In the masked LMs, 15% of the masked token positions are randomly chosen for prediction. For an API invocation sequence "**getText toLowerCase replace split**," if the chosen position is the last one, the input and label are as follows:

- Input: **getText toLowerCase replace** [MASK]
- Label: [MASK] = split

The prediction of the model is the label "**split**." For the input sequence, the masked position is replaced with the [MASK] token 80% of the time, a random API method 10% of the time, or remains unchanged 10% of the time. We perform mask operation on all input service sequences; meanwhile, the labels of masked positions can be obtained. Then, the model will be trained with the samples and can learn these service sequences by predicting the masked positions.

B. A NOVEL LIGHTWEIGHT BERT ARCHITECTURE WITH CONVOLUTIONAL ATTENTION IN TRANSFORMER

As an LM, BERT is usually pretrained with a large dataset because to maintain a certain model complexity, BERT is comparatively large. In this study, we propose using BERT to learn service composition sequences. In comparison to natural language sequences, service composition sequences are more simple in creativity, and the dataset is small. Therefore, developing a comparative lightweight BERT is meaningful in our case. The following description shows that fully connected layers are heavily used in the architecture of the transformer encoder and result in a rapid increase in model size. Convolutional neural networks are used widely in computer vision, single processing, and NLP [39], [40]. Convolutional attention is also applied to several studies. For example, convolutional self-attention was presented for text classification [41] using Conv1D to extract features between neighboring words. In the transformer encoder, multihead attention allows the model to attend jointly to information from different representation subspaces at different positions, and the different Q, K, and V pairs are generated by linear projections. In our approach, we use two-dimensional convolution operations to perform this procedure instead of pure linear projections. If we assume the input of multihead attention as Q, K, and V, the new multihead attention can be illustrated in Fig. 7. The formulation can be given as:

$$\begin{aligned} \text{MultiHead(Q,K,V)} &= \text{Concat}(H_1, \ldots, H_h)W^o \\ \text{where} H_i &= \text{Attention}(Q_i, K_i, V_i), \\ Q_1, \ldots, Q_i &= \text{Conv2D}(QW_{linear} + b), \\ K_1, \ldots, K_i &= \text{Conv2D}(KW_{linear} + b), \\ V_1, \ldots, V_i &= \text{Conv2D}(VW_{linear} + b) \end{aligned} \tag{3}$$

i is equal to the number of heads h and the number of filters. W_{linear} is the weight matrix of a linear projection before the convolution operation because we keep one linear layer to

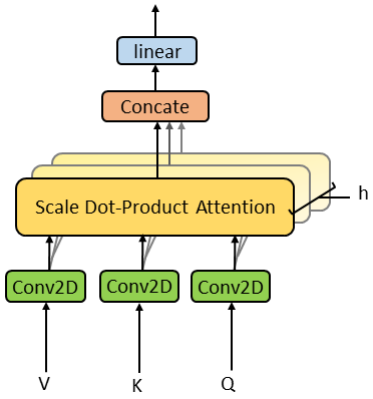


FIGURE 7. Convolutional multi-head attention.

enhance the linear capability of the model; the other components are the same as those in the original model.

C. COMPARISON OF MODEL COMPLEXITY

In this section, we compare the computational complexities of the lightweight model and the base model. For neural network models, the computational complexity can be considered from two aspects: the time complexity T and the number of parameters P. The time complexity generally represents multiply-accumulate operations [42], [43]. For the base BERT model, the computational complexity is given as follows:

$$\begin{aligned} T &\sim O((3Ld_m^2 + Ld_m d_{ff})N + LDd_m), \\ P &\sim O((3d_m^2 + d_m d_{ff})N + Dd_m). \end{aligned} \tag{4}$$

L is the maximum sequence length, D is the vocabulary size, d_m is the embedding dimension, d_{ff} is the hidden dimension, and N is the number of layers. Regarding the new model, the computational complexity is given as follows:

$$\begin{aligned} T &\sim O((Ld_m^2 + 3M^2 Ld_m + Ld_m d_{ff})N + LDd_m), \\ P &\sim O((d_m^2 + 3M^2 h + d_m d_{ff})N + Dd_m). \end{aligned} \tag{5}$$

M represents the filter size and h represents the number of heads.

V. SEMANTIC SERVICE CLUSTERING BASED ON SERVICE EMBEDDING

In NLP, contextual knowledge is very important for semantic segmentation [44]. BERT can understand the semantics of words based on the context and be used to address lexical ambiguity. In the same way, the same service match with different services can obtain different service compositions with different functions. If these composition sequences are used to pretrain a BERT-based service embedding model, the model can capture the semantic services and generate the representation vectors. Thus, we can find similar semantic services and retrieve similar semantic compositions. The entire procedure can be illustrated in Fig. 8. The semantic clustering of service composition can be divided into two stages:

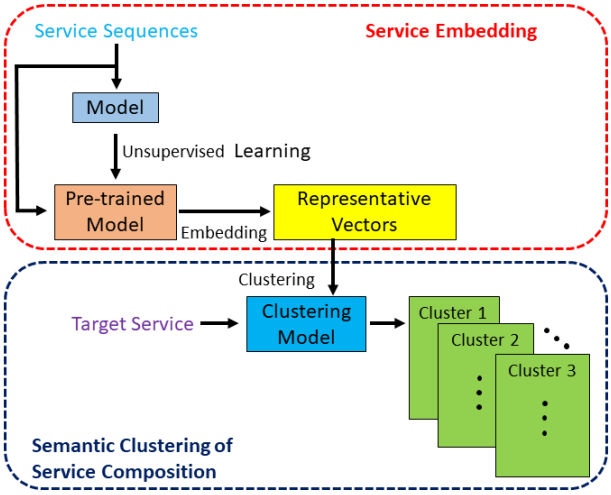


FIGURE 8. Semantic clustering of service composition with service embedding.

The first stage is the service embedding. We use service sequences to pretrain a neural LM and generate the representation vectors through the pretrained model. In our case, BERT-based service embedding models are utilized because the self-attention mechanism can capture the invocation relationship between services. Such knowledge contains the semantic information of services and is already represented by the embedding process. The second stage is to perform clustering. In this paper, we use K-means clustering, which is an unsupervised method for clustering representation vectors. Then, the semantic clustering model can be obtained. When a target service is entered into the clustering model, the different semantic clusters are returned.

VI. DATA PREPARATION

We choose the invocation sequences of Web APIs as the experimental dataset. We crawled Java source codes from GitHub, and these codes were developed for implementing the Twitter APIs. The data preparation is shown in Fig. 9. First, we parse the source code into abstract syntax trees to identify all the methods in each calling method or class. Because the research target is the Twitter API, we need to determine the Twitter API methods and filter some unrelated methods. Finally, we can obtain the Twitter API invocation sequences in a certain definition scope.

In the experiments, we use about 3000 API invocation sequences as training data, and the number of methods is about 800. Compared with other NLP datasets, ours is comparatively small. The reasons for this are twofold: First, the complexity of the model is low. The base BERT in NLP has 110M parameters, but our models have only $1.6M \sim 2.5M$ parameters. The number of parameters is far fewer than the base BERT, so a large dataset is not required. In addition, our model is not a full BERT model, as it does not learn sentence pairs; instead, we just make it predict the masked position to embed the sequences, simplifying the task. Second, the type of dataset is different. Instead of natural

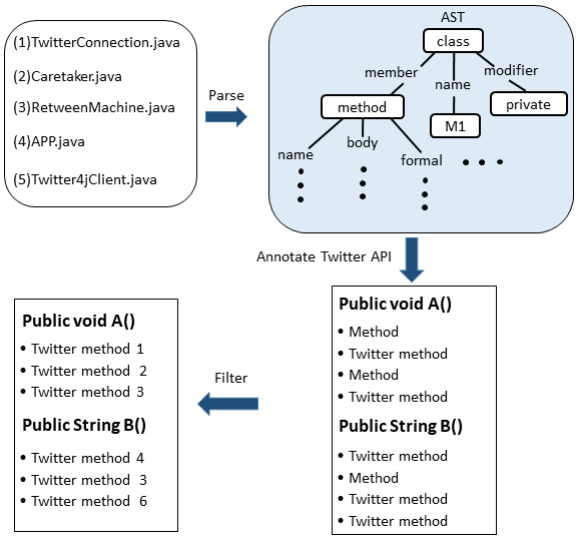


FIGURE 9. Processing of the dataset.

language, our dataset consists of API invocation sequences, which are not as complex or creative. Moreover, nearly all Twitter APIs are almost contained.

VII. EXPERIMENT AND DISCUSSION

In this section, we observe the following two issues: service embedding with the lightweight BERT architecture, and semantic service clustering. For the former, the computational complexity and reduction in model size are evaluated. The experimental results of service embedding are also introduced. For the latter, semantic service clustering with lightweight BERT-based service embedding by invocation sequence is discussed through clustering performance.

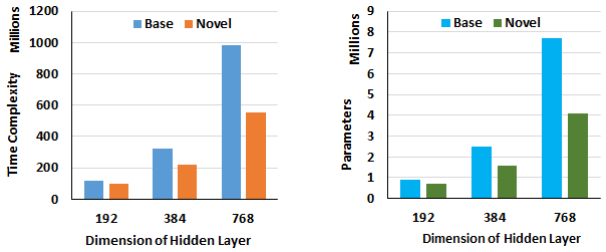
TABLE 2. Hyperparameters of Models.

Model	N	d_{model}	d_{ff}	h	Filter Size
Base	3	384	768	6	—
lightweight	3	384	768	6	3*9

A. SERVICE EMBEDDING WITH LIGHTWEIGHT BERT-BASED MODELS

1) CALCULATION OF COMPUTATIONAL COMPLEXITY

The purpose of this experiment is to compare the performance of the base model of BERT and the lightweight model with the proposed architecture. For the two models, the hyperparameters are set as Table 2. The batch size is 12, the maximum sequence length is 128, the vocabulary size is 800, and the other configurations reference the original literature [36]. Referring to section IV-C, the computational complexity of the two models can be calculated. As shown in Fig. 10, as d_m changes, the time complexity and the number of parameters all increased dramatically in the two models. However, compared with the base model, the lightweight model has reduced time complexity and a smaller number of parameters. When the embedding dimension d_m is set as 384, the time



(a) The time complexity (b) The number of parameters

FIGURE 10. Computational complexity of the models.

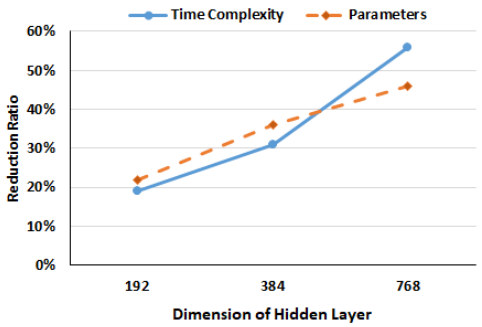
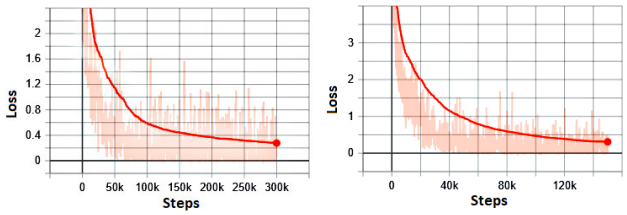


FIGURE 11. Reduction ratio of time complexity and parameters.

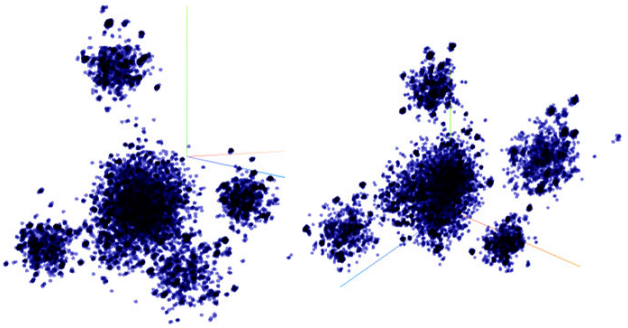


(a) Base model (b) Lightweight model

FIGURE 12. Loss of models in training process.

complexity of the base model is about 322M and the number of parameters is about 2.5M. For the lightweight model, the time complexity is about 221M and the number of parameters is about 1.6M. The reduction ratio is also computed and shown in Fig. 11. In the lightweight model, the time complexity can be reduced by 19% $\sim$ 56%, and the number of parameters can be reduced by 22% $\sim$ 46%. Theoretically, it is more lightweight and faster than the base model. In the service of deep learning-based applications, the response time that includes transmission delay, scheduling time, and inference time is a crucial problem. The inference time is the dominant impact factor [16], [45] because the process of deep learning inference is computation intensive. Therefore, when performing the same inference task on edge computing, the inference time of the lightweight model can be reduced by 19% $\sim$ 56% compared with the base model.

The two models were trained on a GTX 1080 Ti. The base model took about 10 hours, while the lightweight model took about 6 hours. The training procedures are shown as Fig. 12. The results show that the loss of the base model becomes stable at about 300K steps, while the lightweight model completes the training at about 150k steps. Consequently, the lightweight model is trained faster than the base model,



(a) Base model (b) Lightweight model

FIGURE 13. Visualization of the representation vectors.

Nearest points in the original space:	
(1032)setMedia	0.002
(2106)setMedia	0.002
(2657)setMedia	0.003
(2079)setMedia	0.003
(2697)setMedia	0.003
(1875)setMedia	0.003
(2937)setMedia	0.004
(2596)setMedia	0.004
(2060)setMedia	0.004

(a) Base model

Nearest points in the original space:	
(2657)setMedia	0.005
(2106)setMedia	0.006
(1032)setMedia	0.006
(2697)setMedia	0.006
(2079)setMedia	0.008
(1875)setMedia	0.009
(2596)setMedia	0.010
(2937)setMedia	0.010
(2060)setMedia	0.010

(b) Lightweight model

FIGURE 14. Nearest points of (185)setMedia.

and this is consistent with the previous comparison of the computational complexity.

2) VISUALIZATION OF SERVICE EMBEDDING

After pretraining, we can obtain the representation vectors of all sequences through the pretrained models. Principal component analysis is used to perform dimension reduction and visualize these representation vectors, and the results are shown in Fig. 13. The results show that the distribution is very similar. Overall, the results show that all points have been divided into several large groups. However, we believe that this has no clear significance. In addition, there are many small clusters, indicating that the model shows promising capability in service embedding.

We compare the results by computing the nearest points of some target method. For example, when ''(185)setMedia'' is chosen as a target, ''185'' represents the number of sequences in the dataset. ''setMedia'' is the name of the API method. Its nearest points in the space can be determined by computing the cosine distance, as shown in Fig. 14. The results show that while the order indicates a few differences, the points are the same. Several target API methods have been used to compare the difference between the nearest points. The result is the same as the example. Thus, through measuring the visualization result, the performances of the two models are close.

B. SEMANTIC SERVICE CLUSTERING

Our approach aims to realize semantic service clustering. When consumers input a target service, the clustering model

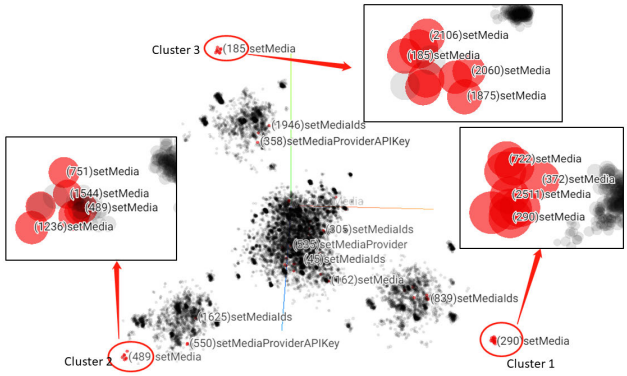


FIGURE 15. Example of semantic clustering for ''setMedia''.

can return different semantic clusters that contain the target service. In this experiment, the K-means clustering algorithm is used to construct a clustering model. K-means clustering is a kind of unsupervised learning algorithm and is widely used. For K-means algorithms, the number of clusters K needs to be determined in advance. Therefore, several values are used in the experiments with the purpose of comparing the performance of clustering models in different k values.

For the evaluation of cluster quality [5], [20], purity and entropy are used. In addition, we accordingly adjust entropy in our case. Purity shows the purity of a cluster and is defined as follows:

$$Purity = \frac{1}{n} max_c\{n_j^i\}. \tag{6}$$

n is the total number of services in the cluster. $max_c\{n_j^i\}$ represents the number of semantic class i in cluster j, and only the maximum value is considered. Entropy demonstrates how the various semantic classes are distributed within each cluster. A smaller entropy value corresponds to better clustering quality. The entropy of a single cluster is defined as follows:

$$E_j = -\sum_{i=1}^{q} \frac{n_j^i}{n_j} \log \frac{n_j^i}{n_j}. \tag{7}$$

n_j^i is the number of services of a semantic class i in cluster j, and n_j is the number of services in a cluster j. The final entropy value is given as:

$$Entropy = \log G \sum_{j=1}^{G} E_j. \tag{8}$$

G is the number of clusters.

Semantic clustering of service composition provides different semantic categories of compositions when consumers enter a target service. The semantic category means the clustered service composition sequences have similar functional descriptions. Here, we show an example with the API method ''setMedia.'' In the previous experiment, the three-dimensional visualization of service embedding was demonstrated, and some clusters already appeared, as shown in Fig. 15. In Fig. 15, three clusters—cluster 1, cluster 2,

TABLE 3. Sequences in Cluster 1.

No.	Invocation sequences in cluster 1	Functional description
1	setInReplyToStatusId setLocation **setMedia** setPossiblySensitive getStatus contains updateStatus	Twitter client for Android
2	setOAuthConsumer setOAuthAccessToken **setMedia** updateStatus	Twitter account logs in and updates
3	setInReplyToStatusId setLocation **setMedia** setPossiblySensitive updateStatus getUserMentionEntities getId	Twitter client for Android
4	getErrorMessage printStackTrace **setMedia**	Media diagnosis in background
5	setInReplyToStatusId setLocation **setMedia** getStatus contains updateStatus	Twitter client for Android
6	printStackTrace toString **setMedia**	Media diagnosis in background
7	setLocation setInReplyToStatusId **setMedia** setPossiblySensitive printStackTrace updateStatus	Twitter client for Android
8	getText getScreenName **setMedia**	Handle input
9	getMessage setInReplyToStatusId **setMedia**	Follow users

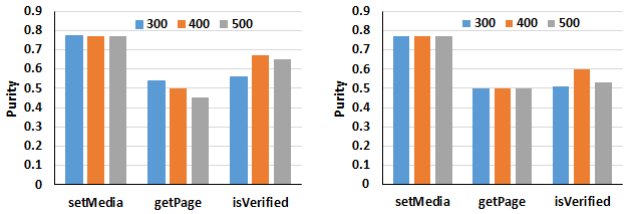
TABLE 4. Sequences in Cluster 2.

No.	Invocation sequences in cluster 2	Functional description
10	getLocalizedMessage **setMedia** getText	Send tweets via Twitter
11	getMessage **setMedia**	Post a tweet on Twitter
12	getErrorMessage **setMedia** updateStatus	Media diagnosis in background
13	inReplyToStatusId **setMedia** updateStatus getText getId getId	Post tweets
14	getMessage **setMedia** getMediaEntities	Tweet with image
15	printStackTrace **setMedia** printStackTrace	Media diagnosis
16	setInReplyToStatusId **setMedia**	Set on click listener

and cluster 3—are used to make an explanation. First, we determine the sequences that contain "setMedia," as listed in Tables 3 to 5. From Table 3, sequences 1, 3, 5, and 7 have the same functional description: "Twitter client for Android." In Table 4, sequences 10, 11, 13, and 14 have the similar functional description: "Send/Post a Tweet." In Table 5, all sequences except for 18 and 24 have the same functional description: "update status." In each cluster, these sequences with similar functional descriptions are regarded as the same semantic category. Based on the presented analysis, the same semantic category is mostly distributed in the same cluster. In general, service embedding can effectively capture the semantic information from the invocation sequences. The

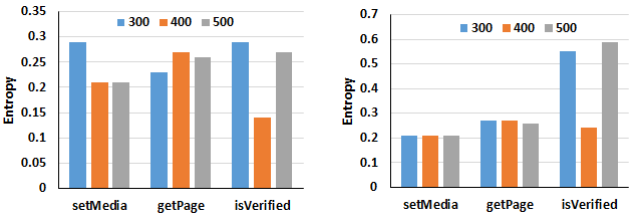
TABLE 5. Sequences in Cluster 3.

No.	Invocation sequences in cluster 3	Functional description
17	**setMedia** getStatus	Update status
18	**setMedia** setLocation	On Start command
19	**setMedia** setInReplyToStatusId tweets updateStatus	Twitter mention channel update
20	**setMedia** UpdateStatus inReplyToStatusId updateStatus	Update status
21	**setMedia** printStackTrace updateStatus	Update status
22	**setMedia** updateStatus	Update status
23	**setMedia** updateStatus getText getId getId	Update status
24	**setMedia** updateStatus getStatus getPlaceId	Send task to twitter



(a) Base model (b) Lightweight model

FIGURE 16. Purity of the models.

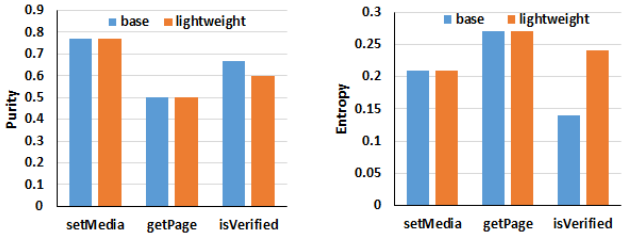


(a) Base model (b) Lightweight model

FIGURE 17. Entropy of the models.

presented example is just the simple visualization process. Then, we apply the K-means clustering algorithm to perform semantic clustering and inspect the clustering quality with previous evaluation metrics. If we inspect all the clusters with the presented metrics, the workload is very heavy because it needs to annotate the functional description of all API sequences. Thus, about 100 sequences that all contain three Twitter methods—"setMedia," "getPage," and "isVerified"—have been chosen as evaluation samples.

The results are shown in Fig. 16-18. Fig. 16 is the purity of two models with different numbers of clusters. The results show that the value of purity is from 50% to 77%, and the purity values are very similar when the number of clusters changes from 300 to 500. However, better performance is always obtained by $K = 400$ for these two models. Fig. 17 is the entropy of two models with different numbers of clusters, and the low value corresponds to better clustering quality.



(a) Purity (b) Entropy

FIGURE 18. Comparison of clustering quality ($K = 400$).

The results show that the best performance is also obtained by $K = 400$ in these two models. Therefore, we compare the performances of the two models at $K = 400$. As shown in Fig. 18, the purities of the two models are very similar, but the entropy of the base model is slightly better than that of the lightweight model. Generally, the clustering quality of the two models is approximately the same in the experiment, and this is consistent with the previous experiment.

C. DISCUSSION

In this section, we discuss our approach from two aspects: clustering category and specific performance when using different neural LMs. First, regarding the different clustering categories, as mentioned previously, traditional approaches usually involve service signature in service representation, such as names of operations and IOPEs, and cluster similar service domains together. As shown in Fig. 19, similar service domains are clustered by generating the ontologies of service names [5] such as medical, quantity, and academia. Differing from traditional approaches, our approach performs service clustering based on service invocation sequences. As shown in Fig. 20, the invocation sequences with similar functional descriptions were clustered together. For example, sequences 1954, 1129, 290, 722, and 1698 were clustered together with the same functional description "Twitter client for Android." Second, through changing different neural LMs, we developed three service embedding models: a RNN-based model, a base BERT-based model, and a lightweight BERT-based model. The RNN-based model is a common RNN encoder-decoder model [46] and the target sentence only shifts the right position of the input sequence. In addition, the output vector of the encoder is the representation vector of the corresponding input token. Referring to existing studies [5], we use precision, recall, and F-measure to evaluate our examples and compare the performance of these three models.

$$Precision(i,j) = \frac{NM_{ij}}{NM_j} \tag{9}$$

$$Recall(i,j) = \frac{NM_{ij}}{NM_i} \tag{10}$$

$$F(i,j) = \frac{2 \times precision(i,j) \times recall(i,j)}{precision(i,j) + recall(i,j)} \tag{11}$$

NM_{ij} is the number of class i in cluster j, NM_j is the number of cluster j, and NM_i is the total number of class i. In our case,

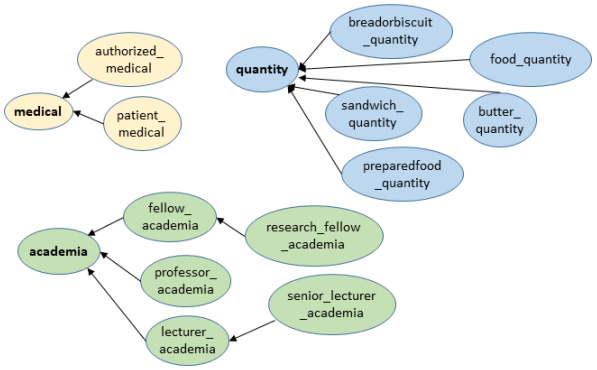


FIGURE 19. Service clustering based on domains with ontology learning.

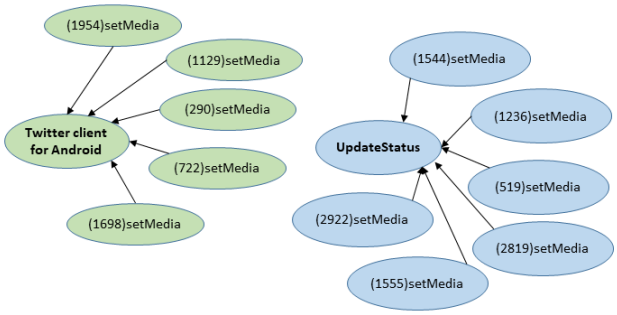


FIGURE 20. Service clustering based on a functional description with service embedding.

TABLE 6. Comparison of different neural LMs.

Model	Target	Precision (%)	Recall (%)	F-Measure (%)
RNN-based service embedding	setMedia	60	34	41
	getPage	33	20	22
	isVerified	22	13	16
Base BERT-based service embedding	setMedia	74.1	61.1	66.9
	getPage	63.3	33.3	43.6
	isVerified	76.6	63.4	68
Light weight BERT-based service embedding	setMedia	74.1	61.1	66.9
	getPage	63.3	33.3	43.6
	isVerified	72.2	63.4	66

for each target, the three largest clusters are considered, and the average value is the final value.

The results are shown in Table 6. The two BERT-based models show far better performance than the RNN-based model. The lightweight BERT-based model has a similar result as the base BERT-based model. For the target "isVerified", the base BERT-based model outperforms the lightweight BERT-based model. This is consistent with previous experiments. For the two BERT-based models, the recall of the "getPage" is only 33.3%. Based on our analysis, this effect is because of two reasons: First, the computation process of recall in this paper is different from the existing literature. Instead of computing one cluster, we compute three clusters and use their average

value. Consequently, the final value is low when one of the three clusters gains low recall. Second, the frequency of "getPage" is low. When the number of some classes is comparatively small, there is a large probability one of the three clusters obtains low recall.

VIII. CONCLUSION

In this paper, to perform semantic service clustering, we proposed a novel deep learning-based approach called lightweight BERT-based service embedding. Moreover, another novel aspect of our proposal is the much lighter and faster model for service embedding compared with the base BERT-based model. The experiment results show that our approach can effectively perform semantic clustering of service composition based on the invocation relationship. Furthermore, the lightweight BERT-based model could obtain a high clustering quality as good as the base BERT-based model but with fewer parameters, smaller time complexity, and faster training speed. Consequently, the invocation relationship between services is vital information in service clustering. At this stage, one limitation of our approach is the lack of high clustering precision when some services have low frequency in the dataset. In future work, we will try to improve the performance by improving the architecture of the neural network model, and by combining some traditional approaches.

REFERENCES

[1] J. Wu, L. Chen, Z. Zheng, M. R. Lyu, and Z. Wu, "Clustering Web services to facilitate service discovery," *Knowl. Inf. Syst.*, vol. 38, no. 1, pp. 207–229, Jan. 2014.

[2] A. K. Tripathy, M. R. Patra, M. A. Khan, H. Fatima, and P. Swain, "Dynamic Web service composition with QoS clustering," in *Proc. IEEE Int. Conf. Web Services*, Jun. 2014, pp. 678–679.

[3] M. Shi, J. Liu, D. Zhou, M. Tang, and B. Cao, "WE-LDA: A word embeddings augmented LDA model for Web services clustering," in *Proc. IEEE Int. Conf. Web Services (ICWS)*, Jun. 2017, pp. 9–16.

[4] G. Zou, Z. Qin, Q. He, P. Wang, B. Zhang, and Y. Gan, "DeepWSC: Clustering Web services via integrating service composability into deep semantic features," *IEEE Trans. Services Comput.*, early access, Sep. 23, 2020, doi: 10.1109/TSC.2020.3026188.

[5] B. T. G. S. Kumara, I. Paik, W. Chen, and K. H. Ryu, "Web service clustering using a hybrid term-similarity measure with ontology learning," *Int. J. Web Services Res.*, vol. 11, no. 2, pp. 24–45, Apr. 2014.

[6] J. Han, S. Park, and J. Kim, "Dynamic OverCloud: Realizing microservices-based IoT-cloud service composition over multiple clouds," *Electronics*, vol. 9, no. 6, p. 969, Jun. 2020.

[7] H.-L. Truong and P. Klein, "DevOps contract for assuring execution of IoT microservices in the edge," *Internet Things*, vol. 9, Mar. 2020, Art. no. 100150.

[8] C. Jian, M. Li, and X. Kuang, "Edge cloud computing service composition based on modified bird swarm optimization in the Internet of Things," *Cluster Comput.*, vol. 22, no. S4, pp. 8079–8087, Jul. 2019.

[9] B. Xia, Y. Fan, W. Tan, K. Huang, J. Zhang, and C. Wu, "Category-aware API clustering and distributed recommendation for automatic mashup creation," *IEEE Trans. Services Comput.*, vol. 8, no. 5, pp. 674–687, Sep. 2015.

[10] B. Cao, X. Liu, B. Li, J. Liu, M. Tang, T. Zhang, and M. Shi, "Mashup service clustering based on an integration of service content and network via exploiting a two-level topic model," in *Proc. IEEE Int. Conf. Web Services (ICWS)*, Jun. 2016, pp. 212–219.

[11] V. Hayyolalam and A. A. P. Kazem, "A systematic literature review on QoS-aware service composition and selection in cloud environment," *J. Netw. Comput. Appl.*, vol. 110, pp. 52–74, May 2018.

[12] A. Vakili and N. J. Navimipour, "Comprehensive and systematic review of the service composition mechanisms in the cloud environments," *J. Netw. Comput. Appl.*, vol. 81, pp. 24–36, Mar. 2017.

[13] I. Al Ridhawi, M. Aloqaily, Y. Kotb, Y. Al Ridhawi, and Y. Jararweh, "A collaborative mobile edge computing and user solution for service composition in 5G systems," *Trans. Emerg. Telecommun. Technol.*, vol. 29, no. 11, Nov. 2018, Art. no. e3446.

[14] B. Pang, F. Hao, D.-S. Park, and C. D. Maio, "A multi-criteria multi-cloud service composition in mobile edge computing," *Sustainability*, vol. 12, no. 18, p. 7661, Sep. 2020.

[15] H. Wu, S. Deng, W. Li, M. Fu, J. Yin, and A. Y. Zomaya, "Service selection for composition in mobile edge computing systems," in *Proc. IEEE Int. Conf. Web Services (ICWS)*, Jul. 2018, pp. 355–358.

[16] X. Wang, Y. Han, V. C. M. Leung, D. Niyato, X. Yan, and X. Chen, "Convergence of edge computing and deep learning: A comprehensive survey," *IEEE Commun. Surveys Tuts.*, vol. 22, no. 2, pp. 869–904, 2nd Quart., 2020.

[17] K. Elgazzar, A. E. Hassan, and P. Martin, "Clustering WSDL documents to bootstrap the discovery of Web services," in *Proc. IEEE Int. Conf. Web Services*, Jul. 2010, pp. 147–154.

[18] W. Liu and W. Wong, "Web service clustering using text mining techniques," *Int. J. Agent-Oriented Softw. Eng.*, vol. 3, no. 1, pp. 6–26, Feb. 2009.

[19] J.-X. Liu, K.-Q. He, J. Wang, and D. Ning, "A clustering method for Web service discovery," in *Proc. IEEE Int. Conf. Services Comput.*, Jul. 2011, pp. 729–730.

[20] A. Bukhari and X. Liu, "A Web service search engine for large-scale Web service discovery based on the probabilistic topic modeling and clustering," *Service Oriented Comput. Appl.*, vol. 12, no. 2, pp. 169–182, Jun. 2018.

[21] F. Chen, M. Li, H. Wu, and L. Xie, "Web service discovery among large service pools utilising semantic similarity and clustering," *Enterprise Inf. Syst.*, vol. 11, no. 3, pp. 452–469, Mar. 2017.

[22] D. Martin, M. Paolucci, S. McIlraith, M. Burstein, D. McDermott, D. McGuinness, B. Parsia, T. Payne, M. Sabou, M. Solanki, and N. Srinivasan, "Bringing semantics to Web services: The OWL-S approach," in *Proc. Int. Workshop Semantic Web Services Web Process Composition*. Berlin, Germany: Springer, 2004, pp. 26–42.

[23] J. De Bruijn, C. Bussler, J. Domingue, D. Fensel, M. Hepp, M. Kifer, B. König-Ries, J. Kopecky, R. Lara, E. Oren, and A. Polleres, "Web service modeling ontology (WSMO)," *Interface*, vol. 5, no. 1, p. 50, 2005.

[24] D. Roman, J. Kopecký, T. Vitvar, J. Domingue, and D. Fensel, "WSMO-lite and hRESTS: Lightweight semantic annotations for Web services and RESTful APIs," *J. Web Semantics*, vol. 31, pp. 39–58, Mar. 2015.

[25] J. Kopecký, T. Vitvar, C. Bournez, and J. Farrell, "SAWSDL: Semantic annotations for WSDL and XML schema," *IEEE Internet Comput.*, vol. 11, no. 6, pp. 60–67, Nov. 2007.

[26] J. Pathak, N. Koul, D. Caragea, and V. G. Honavar, "A framework for semantic Web services discovery," in *Proc. 7th Annu. ACM Int. Workshop Web Inf. Data Manage.*, 2005, pp. 45–50.

[27] M. Bellouki, "Review of ontology based approaches for Web service discovery," in *Proc. Smart Data Comput. Intell., Int. Conf. Adv. Inf. Technol., Services Syst. (AIT2S) Held*, vol. 66, Mohammedia, Morocco: Springer, Oct. 2019, p. 78.

[28] Z. Maamar, H. Hacid, and M. N. Huhns, "Why Web services need social networks," *IEEE Internet Comput.*, vol. 15, no. 2, pp. 90–94, Mar. 2011.

[29] Z. Maamar, N. Faci, L. Wives, Y. Badr, P. Santos, and J. P. M. de Oliveira, "Using social networks for Web services discovery," *IEEE Internet Comput.*, vol. 15, no. 4, pp. 48–54, Jul. 2011.

[30] W. Chen and I. Paik, "Toward better quality of service composition based on a global social service network," *IEEE Trans. Parallel Distrib. Syst.*, vol. 26, no. 5, pp. 1466–1476, May 2015.

[31] A. Corbellini, D. Godoy, C. Mateos, A. Zunino, and I. Lizarralde, "Mining social Web service repositories for social relationships to aid service discovery," in *Proc. IEEE/ACM 14th Int. Conf. Mining Softw. Repositories (MSR)*, May 2017, pp. 75–79.

[32] X. Gu, H. Zhang, D. Zhang, and S. Kim, "Deep API learning," in *Proc. 24th ACM SIGSOFT Int. Symp. Found. Softw. Eng.*, Nov. 2016, pp. 631–642.

[33] S. Bhupatiraju, R. Singh, A.-R. Mohamed, and P. Kohli, "Deep API programmer: Learning to program with APIs," 2017, *arXiv:1704.04327*. [Online]. Available: http://arxiv.org/abs/1704.04327

[34] D. Wu, X.-Y. Jing, H. Chen, X. Zhu, H. Zhang, M. Zuo, L. Zi, and C. Zhu, "Poster: Automatically answering API-related questions," in *Proc. IEEE/ACM 40th Int. Conf. Softw. Eng., Companion (ICSE-Companion)*, May 2018, pp. 270–271.
[35] A. Vaswani, N. Shazeer, N. Parmar, J. Uszkoreit, L. Jones, A. N. Gomez, L. Kaiser, and I. Polosukhin, "Attention is all you need," in *Proc. Adv. Neural Inf. Process. Syst.*, 2017, pp. 5998–6008.
[36] J. Devlin, M.-W. Chang, K. Lee, and K. Toutanova, "BERT: Pre-training of deep bidirectional transformers for language understanding," 2018, *arXiv:1810.04805*. [Online]. Available: http://arxiv.org/abs/1810.04805
[37] K. He, X. Zhang, S. Ren, and J. Sun, "Deep residual learning for image recognition," in *Proc. IEEE Conf. Comput. Vis. Pattern Recognit. (CVPR)*, Jun. 2016, pp. 770–778.
[38] J. Lei Ba, J. R. Kiros, and G. E. Hinton, "Layer normalization," 2016, *arXiv:1607.06450*. [Online]. Available: http://arxiv.org/abs/1607.06450
[39] H. Zhang, Z. Xiao, J. Wang, F. Li, and E. Szczerbicki, "A novel IoT-perceptive human activity recognition (HAR) approach using multihead convolutional attention," *IEEE Internet Things J.*, vol. 7, no. 2, pp. 1072–1080, Feb. 2020.
[40] S. O. Arik, H. Jun, and G. Diamos, "Fast spectrogram inversion using multi-head convolutional neural networks," *IEEE Signal Process. Lett.*, vol. 26, no. 1, pp. 94–98, Jan. 2019.
[41] S. Gao, A. Ramanathan, and G. Tourassi, "Hierarchical convolutional attention networks for text classification," in *Proc. 3rd Workshop Represent. Learn. (NLP)*, 2018, pp. 11–23.
[42] K. He and J. Sun, "Convolutional neural networks at constrained time cost," in *Proc. IEEE Conf. Comput. Vis. Pattern Recognit. (CVPR)*, Jun. 2015, pp. 5353–5360.
[43] M. A. Nahmias, T. F. de Lima, A. N. Tait, H.-T. Peng, B. J. Shastri, and P. R. Prucnal, "Photonic multiply-accumulate operations for neural networks," *IEEE J. Sel. Topics Quantum Electron.*, vol. 26, no. 1, pp. 1–18, Jan. 2020.
[44] E. Cambria and B. White, "Jumping NLP curves: A review of natural language processing research [review article]," *IEEE Comput. Intell. Mag.*, vol. 9, no. 2, pp. 48–57, May 2014.
[45] Q. Yang, X. Luo, P. Li, T. Miyazaki, and X. Wang, "Computation offloading for fast CNN inference in edge computing," in *Proc. Conf. Res. Adapt. Convergent Syst.*, Sep. 2019, pp. 101–106.
[46] I. Sutskever, O. Vinyals, and Q. Le, "Sequence to sequence learning with neural networks," in *Proc. Adv. NIPS*, 2014, pp. 1–9.

KUNGAN ZENG (Member, IEEE) received the bachelor's degree from the Department of Civil Engineering, Hunan University of Technology, China, in 2014, and the master's degree from the School of Civil Engineering, Guangzhou University, China, in 2016. He is currently pursuing the Ph.D. degree with the School of Computer Science and Engineering, The University of Aizu, Japan. His research interests include deep learning, web services, and natural language processing.

INCHEON PAIK (Senior Member, IEEE) received the M.E. and Ph.D. degrees in electronic engineering from Korea University, in 1987 and 1992, respectively. He is currently a Professor with The University of Aizu, Japan. His research interests include semantic web, web services and their composition, data mining, deep learning, and big data science infrastructure. He is a member of ACM, IEICE, and IPSJ.