A Deep Neural Network With Multiplex Interactions for Cold-Start Service Recommendation

Yutao Ma, *Member, IEEE*, Xiao Geng, and Jian Wang, *Member, IEEE*

***Abstract*—As service-oriented computing (SOC) technologies gradually mature, developing service-based systems (such as mashups) has become increasingly popular in recent years. Faced with the rapidly increasing number of Web services, recommending appropriate component services for developers on demand is a vital issue in the development of mashups. In particular, since a new mashup to develop contains no component services, it is a new "user" to a service recommender system. To address this new "user" cold-start problem, we propose a multiplex interaction-oriented service recommendation approach, named MISR, which incorporates three types of interactions between services and mashups into a deep neural network. In this article, we utilize the powerful representation learning abilities provided by deep learning to extract hidden structures and features from various types of interactions between mashups and services. Experiments conducted on a real-world dataset from ProgrammableWeb show that MISR outperforms several state-of-the-art approaches regarding commonly used evaluation metrics.**

***Index Terms*—Cold start, deep learning, mashup development, service recommendation, service-based system.**

I. Introduction

With the maturity of service-oriented computing (SOC), the development paradigm of software systems is shifting from component-based software development (CBSD) to service-oriented software development (SOSD). The SOSD paradigm can reduce the cost and effort of system development and increase the reusability and quality of software systems [1]. Nowadays, numerous Web services have been published on the Internet, and mashups (i.e., a new type of web application), which provide specific functionalities by integrating one or more Web services, become increasingly popular in this context [2]. The rapidly increasing number of Web services poses significant challenges for effective service management and reuse. Thus, promptly recommending appropriate component services for developers and easing their selection burden has become a vital issue in the development of mashups.

Service recommendation usually refers to recommending services according to users' explicit and implicit preferences (e.g., invocation and subscription) as well as historical data of service compositions [1]. The content information of user requests and service descriptions (including structured or unstructured descriptions in texts or Web services description language documents) can also be utilized if provided. In recent years, researchers from different research fields have proposed many service recommendation approaches. The keyword-based approach [2], [3], ontology-based approach [4], [5], and latent-semantics-based approach [6], [7] were studied to this end in the beginning. Considering the limited capability to achieve better performance by exploiting only the content information, other useful information of service usages, such as invocation history, co-invocation, and popularity, was also involved in improving the recommendation performance further [8]–[14].

The hybrid approach usually uses collaborative filtering (CF), natural language processing (NLP), and other techniques to deal with the content information and usage history. For example, Li *et al.* [8] combined functionality and usage history in a topic model to recommend services in mashup creation. Xiong *et al.* [13] also presented a hybrid recommendation approach by integrating CF and deep learning for NLP. Since most of these hybrid approaches leverage the interaction history between mashups and web application programming interfaces (APIs), they perform well in the normal recommendation process for services. However, if a developer wants to create a new mashup without any component service, the mashup does not have any interaction with the existing APIs. The lack of such interaction information will decrease the performance and generalizability of these hybrid approaches.

More specifically, a specific scenario investigated in this article is described as follows. A developer who plans to develop a new mashup inputs his or her functional requests into a service recommender system. Then, the developer wants to obtain a list of candidate services that can be used in the development of the mashup. From the perspective of the recommender system, the new mashup to be built does not contain any component service, which could be regarded as a new "user" to the recommender system. In such a scenario, the traditional CF-based approach does not work well because no usage history is available to the new mashup. Therefore, how to deal with the new "user" cold-start problem remains challenging for the mashup development.

Manuscript received July 28, 2019; revised October 12, 2019; accepted December 16, 2019. Date of publication January 15, 2020; date of current version November 13, 2020. This work was supported in part by the National Key Research and Development Program of China under Grant 2017YFB1400602, in part by the National Science Foundation of China under Grant 61972292, Grant 61832014, Grant 61702378, and Grant 61672387, in part by the Natural Science Foundation of Hubei Province of China under Grant 2018CFB511, and in part by the Science and Technology Project of Shenzhen City of China under Grant CKCY20180322093215776. Review of this manuscript as arranged by Department Editor P. Hung. *(Corresponding author: Jian Wang.)*

The authors are with the School of Computer Science, Wuhan University, Wuhan 430072, China (e-mail: ytma@whu.edu.cn; xiaogeng5515@whu.edu.cn; jianwang@whu.edu.cn).

Color versions of one or more of the figures in this article are available online at http://ieeexplore.ieee.org.

Digital Object Identifier 10.1109/TEM.2019.2961376

In this article, we propose a **m**ultiplex **i**nteraction-oriented **s**ervice **r**ecommendation approach (referred to as MISR) to address the cold-start problem of developing new mashups. An interaction in the MISR represents an underlying relationship between a mashup and a service. Unlike the hybrid approaches mentioned previously, the objective of the MISR is to take advantage of the dominant representation learning ability of deep learning to learn hidden structures from various interactions between services and mashups. In the proposed approach, three types of interactions between services (or APIs) and mashups, including content interaction, implicit neighbor interaction, and explicit neighbor interaction, are identified and incorporated into a deep neural network (DNN), which can predict ratings of candidate services on a new mashup, i.e., the probabilities of candidate services to be invoked by a new mashup. Note that the content interaction indicates a relationship of functionality matching between a new mashup and a candidate service, and the two neighbor interactions represent the relationships between a new mashup's neighbor mashups that share similar functionalities and candidate services.

The main contributions of this article are threefold.

1) We make an in-depth analysis of the cold-start problem in service recommendation for new mashup development, which has not yet been sufficiently discussed before.
2) We propose a novel multiplex interaction-oriented service recommendation approach, called MISR, by integrating three types of interactions between services and mashups into a DNN.
3) Experiments conducted on a real-world dataset crawled from the website ProgrammableWeb[1] demonstrate that the proposed approach outperforms several state-of-the-art approaches regarding recommendation performance.

The rest of this article is organized as follows. Section II presents the related work of service recommendation. Section III defines the cold-start problem and introduces the details of the proposed approach. Section IV reports the experimental results and analysis. Section V concludes this article.

II. Related Work

The primary goal of recommender systems is to predict user ratings or preferences on an item. Along with great success in commercial applications, they have already been prevalent in the modern society. According to the type of information used in recommender systems, the existing recommendation algorithms mainly fall into three types: collaborative filtering algorithms that utilize usage history, content-based algorithms that utilize the content information, and hybrid algorithms that utilize two or more types of information [15]–[17].

In the past decade, recommendation algorithms have been widely used in the services computing field to address the "service overload" problem on the Internet. Generally speaking, service recommendation systems analyze developers' requests (or their preferences) and recommend appropriate candidate services for them. According to the aforementioned taxonomy of general recommendation algorithms, service recommendation methods can also be divided into the following three types: content-based approach, CF-based approach, and hybrid approach.

A. Content-Based Service Recommendation

The content-based approach recommends services according to the content similarities between candidate services and the target mashup. As earlier progress in this direction, the keyword-based approach matches services to mashup development requests in terms of keyword similarities, but it cannot recommend semantically relevant services [2], [3].

Semantics-aware service recommendation approaches were then proposed to overcome the limitation of the keyword-based approach. These approaches can be generally classified into two categories. First, the ontology-based approach annotates mashup requests and service descriptions with domain ontologies and calculates their semantic similarities based on logical reasoning [4], [5]. However, the lack of appropriate domain ontologies and the high cost of manual annotation make it difficult to apply such an approach to large-scale datasets [18]. Second, the latent-semantics-based approach usually extracts text features by using topic models and measures the content relevance of services to mashup requests in terms of their feature similarities [6], [7]. However, the bag-of-words model used in the approach ignores word orders, possibly leading to the loss of semantic information.

Due to the remarkable progress of deep learning in NLP, in this article, we will utilize a DNN to extract text features from the content information automatically.

B. CF-Based Service Recommendation

Collaborative filtering, which captures users' implicit requirements from their usage history, has been widely used in service recommendation. The CF-based approach predicts the quality of service (QoS) by leveraging historical QoS records of similar users or services, aiming to recommend and select high-quality services. For example, Zheng *et al.* [19] proposed a neighborhood integrated matrix factorization (MF) approach to predict QoS values. Chen *et al.* [20] presented a neighbor-based approach to predict QoS values of candidate services by utilizing the historical records of neighbors within the same region. Liu *et al.* [21] made use of the location information to find similar neighbors for users and services, and they predicted QoS values using a location-aware CF method.

Besides QoS prediction, CF was also applied in some service recommendation approaches to find similar users or services. For example, in [22], a hybrid random walk approach was adopted in computing the similarities between indirect users or services, and an improved CF model was designed for service recommendation. Zou *et al.* [23] integrated user-intensive and service-intensive CF in a reinforced CF approach and eliminated the interference of the services (or users) dissimilar with the target service (or the target user). In [24]–[26], the authors built a heterogeneous information network (HIN) using various types

[1] https://www.programmableweb.com

of information of mashups and services, measured an overall similarity score between mashups based on HIN, and finally, made a rating prediction using the user-based CF.

Since the service recommendation problem investigated in this article is for a cold-start scenario, new mashups do not have any usage history with the existing services, which hinders the CF-based approach from achieving ideal results. Inspired by the most "similar" strategy of the user-based CF, in this article, we will learn the interaction between a new mashup and a candidate service from the interactions between the mashup's neighbors (i.e., semantically similar mashups) and the service.

C. Hybrid Service Recommendation

Considering the performance limitation of a single prediction model in service recommendation, many hybrid approaches that integrate multiple models or various kinds of feature information have been proposed in recent years.

Some hybrid approaches usually integrate additional feature information into topic-model-based service recommendations. In [8], the invocation records between mashups and services were incorporated into a latent Dirichlet allocation (LDA) [27] model to discover topics from the content information, which enables the learned topics to model the connections among services, mashups, and words. Gao *et al.* [9] applied LDA to the data structure made up of services and their co-occurring services. Xia *et al.* [10] clustered services into categories based on their popularities and topic features extracted by LDA, and they then combined the services in the most relevant category to generate a set of candidate services.

Other hybrid approaches make use of the content information and usage history in service recommendation. In [11] and [12], the authors calculated the functional correlation scores between services and mashups based on topic models and neighbor interaction probabilities by using CF methods, and then, they multiplied the scores to generate a list of candidate services. However, these linear and multiplication-based approaches have a limited ability to capture complex interactions between mashups and services.

Deep-learning-based recommendation approaches, such as Wide & Deep [28] and neural collaborative filtering (NCF) [29], have been recently proposed. Wide & Deep memorizes interactions with data with a large number of features. However, the number of features identified in the interactions between mashups and services is usually very small, which makes this method difficult to apply to service recommendation scenarios. Instead, NCF, which combines the advantages of neural networks and CF, has begun to attract much attention in this research field. For example, Xiong *et al.* [13] integrated the invocation records between mashups and services as well as their content similarities into a DNN. Chen *et al.* [14] presented a preference-based neural CF recommender model, which leveraged feature vectors of users and items, including language preference and historical data, to recommend appropriate services in the normal recommendation process. However, the aforementioned two DNN models do not work well for developing new mashups without any component service before recommendation.

TABLE I
SYMBOLS USED IN THIS ARTICLE

Symbol	Meaning
m	A new mashup to be built
s	A candidate service to be rated
MS	A mashup-service invocation matrix
NM	The neighbor mashup set of m that shares similar functionalities
nm_i	A neighbor mashup of m, i.e., an element of NM
$\mathbf{v}_{\text{seq}_m}$, $\mathbf{v}_{\text{seq}_s}$	Feature vectors extracted from the content information of mashup m and service s, represented in the form of a word sequence
$\mathbf{v}_{\text{set}_m}$, $\mathbf{v}_{\text{set}_s}$	Feature vectors extracted from the content information of mashup m and service s, represented in the form of a separate word set
$\mathbf{e}_{t_i}$	An embedding vector of word t_i
$\mathbf{ci}_{ms}$	A vector of the content interaction between m and s
sim_{m,nm_i}	The similarity between m and nm_i
$\mathbf{r}_{nm_i}$	The latent representation of nm_i obtained by node2vec
$\mathbf{r}_s$	The latent representation of s obtained by node2vec
$\mathbf{ini}_{ms}$	A vector denoting the implicit neighbor interaction between m and s
$\mathbf{s}_{ms}$	A sparse binary vector constructed based on the invocation records of NM on s
$\mathbf{eni}_{ms}$	A vector denoting the explicit neighbor interaction between m and s
$\hat{r}_{ms}$	The possibility of m invoking s, i.e., the rating of s over m
Y^+, Y^-	All positive samples and all negative ones in the training set
MLP_{CI}	MLPs used in the content interaction (CI) component, the implicit neighbor interaction (INI) part, and the explicit neighbor interaction (ENI) part, respectively
MLP_{INI}	
MLP_{ENI}	
$\text{MLP}_{\text{fusion}}$	An MLP used to fuse multiple interaction vectors in the top layer of MISR

In brief, these existing hybrid approaches have limitations in capturing complex interactions between mashups and services, especially in the scenario of the new "user" cold-start problem. This article is, therefore, to address this problem.

III. MULTIPLEX INTERACTION-ORIENTED SERVICE RECOMMENDATION

First of all, in Section III-A, we state the problem studied in this article and analyze our solution with a real-life case. Next, we detail two main components of the MISR, namely the content interaction component and neighbor interaction component, in Sections III-B and III-C, respectively. Section III-D then shows how the two components are combined to make rating prediction. Finally, we describe the offline training and online prediction phases in Sections III-E and III-F, respectively. In Table I, we list those frequently-used symbols in this section and their respective meanings.

A. Overall Framework

1) Problem Statement: Above all, the problem to be addressed in this article can be described as follows. Given a request (in text form) to develop a new mashup, how to recommend appropriate component services for developers to facilitate the development process? In this article, we focus on recommending possible component services for new mashups to be built.

Suppose a developer plans to develop a new mashup that tracks the price of products on Amazon and provides price drop alerts for users. First, the developer inputs the requirements into

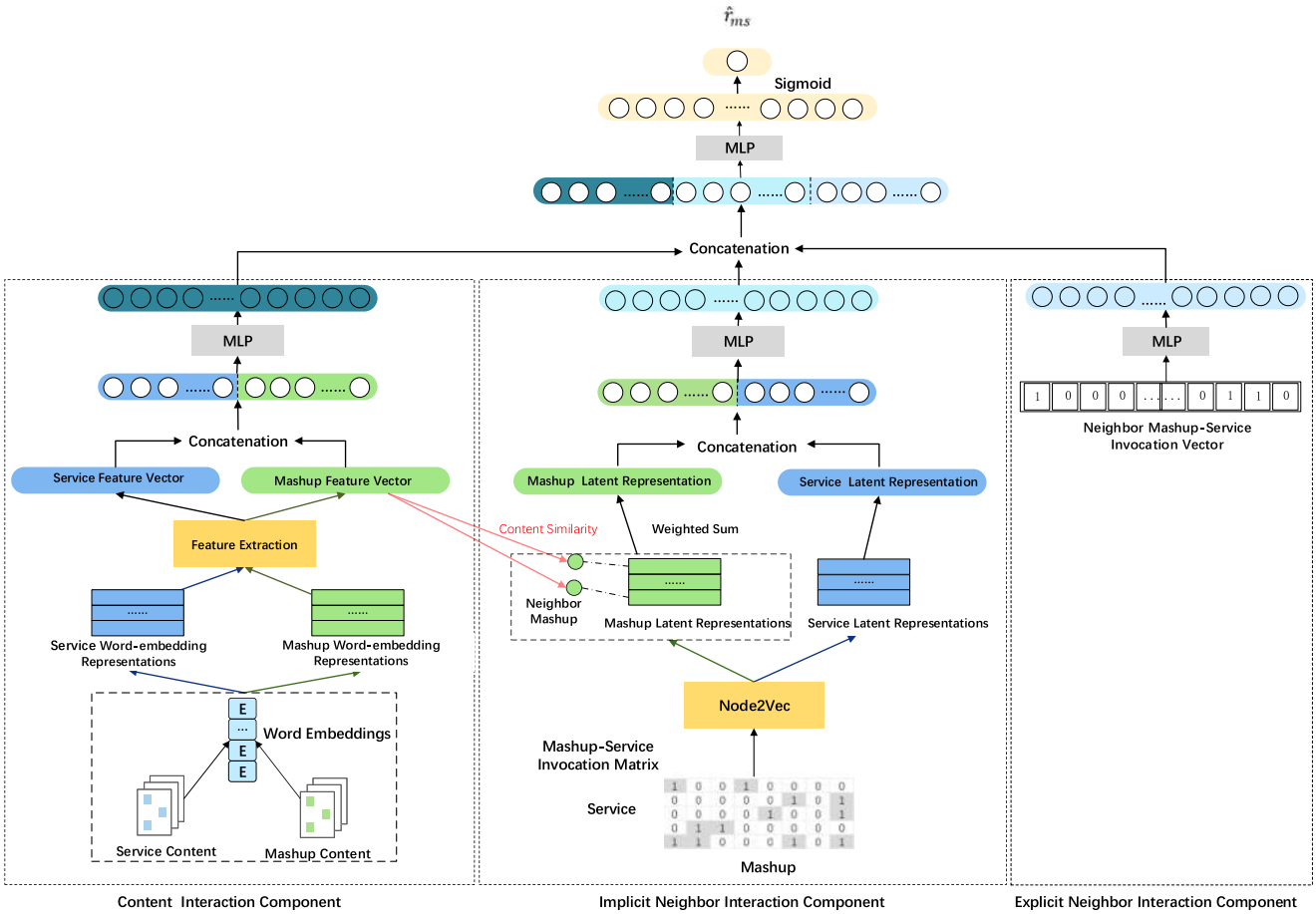


Fig. 1. Architecture of MISR.

a service recommendation system. The system aims to predict the rating of each candidate service (denoted as s) over the new mashup (denoted as m). We take "Amazon Product Advertising API" as an example service, whose description is "Through this API, developers can retrieve product information. The API exposes Amazon's product data and e-commerce functionality."

We then analyze our solution from two aspects. On the one hand, requests can be used to match with services that provide the desired functionalities. Since developers are more likely to select services that are able to meet their functional requests when developing mashups, a content interaction component can be designed to learn the functional interactions based on the content information of m and s, as well as to analyze the possibility of a developer selecting s for m from the functional perspective. Here, the content information means user requests and functionality descriptions of candidate services, and the content interaction indicates the relationship of functionality matching between m and s, i.e., whether the functionality of s (e.g., retrieving product price) can meet the request of m.

On the other hand, the invocation history between mashups and services can also be used to improve the recommendation performance. Unfortunately, a new mashup has no invocation history. If two mashups have similar functional descriptions, they have a substantial probability of invoking the same service; in other words, they may have similar interactions with the same service. Although new mashup m has no interaction with service s, the interactions between s and the neighbor mashups (referred to as NM) of m that share similar functionalities can be leveraged. The most intuitive way is to predict unknown interactions between m and s from all the historical interactions between NM and s (i.e., neighbor interaction). For example, the content information of an existing mashup, PriceZombie, is similar to the request of m, and s is a component service of PriceZombie. Therefore, there is a substantial possibility that m interacts with s.

In this article, we attempt to utilize neighbor interactions to alleviate the absence of the direct interaction between m and s. As a result, we propose a service recommendation approach to learn the interactions between a new mashup and candidate services based on their content information (also known as requests in text form or functional descriptions) and the invocation history of the new mashup's neighbor mashups on the services.

2) Model Framework: As shown in Fig. 1, the proposed MISR approach consists of two primary components: a content interaction (CI) component, and a neighbor interaction (NI) component, which is made up of an implicit neighbor interaction (INI) part and an explicit neighbor interaction (ENI) part. The CI component is a prerequisite for the NI component, and the two parts in NI work in parallel. All the components are complementary to each other. Different types of interactions between

candidate services and mashups can be learned from the three components. These interaction vectors are then concatenated and fed into a multiple layer perceptron (MLP), which predicts the ratings of candidate services over the target mashup.

The CI component first represents the content information of m and s as their word-embedding forms, and then, extracts their respective feature vectors. Finally, an MLP is designed to process the concatenated features and learn their content interactions.

The NI component aims to learn interactions between m and s based on the usage history of the neighbor mashups of m to s. A prerequisite of the component is to get neighbor mashups that are most similar to m in terms of content similarity, which will be detailed in Section III-C. In the INI part, neighbor mashups and s are mapped into a deep feature space by applying node2vec [30], a graph embedding technology in deep learning, to the invocation matrix. Then, we can obtain the weighted representation of m in the space. After concatenating the representations of m and s, an MLP is used to learn their interactions in this feature space. The ENI part learns more direct interactions from the original invocations about the neighbor mashups of m to s.

B. Content Interaction Component

The functionality descriptions of mashups and services, namely their content information, generally have two forms: word sequence and separate word set. We use descriptions and tags as the representatives of the two types of information and extract their features in different ways. Finally, the content interaction between mashups and services can be learned based on the extracted features.

Before extracting features by deep learning techniques, we need to use a dense vector to represent each term that appears in the content information of existing mashups and services. To this end, we first transform these terms into sparse binary vectors with one-hot encoding, e.g., [0, 0, …, 1, …, 0]. Next, we feed the vectors into an embedding layer and map each term to a dense vector or an embedding. More specifically, the embedding layer can be viewed as a lookup table, and the embedding of a term is indeed its corresponding weights in the embedding layer.

1) Feature Extraction From Word Sequences: After the preprocessing of truncation or padding, the word sequence information of m and s are transformed into their respective word-embedding forms, E_m and E_s, which are two matrices with a fixed size. The process can be described as

$$E = [\mathbf{e}_{t_1}, \mathbf{e}_{t_2}, \ldots, \mathbf{e}_{t_i}, \ldots, \mathbf{e}_{t_L}]^T \tag{1}$$

where E denotes the word-embedding representation of a word sequence, L is the length of the processed sequence, t_i is the ith term in the sequence, and $\mathbf{e}_{t_i}$ is the D-dimensional word embedding for t_i.

We then extract feature vectors from E_m and E_s. As a popular class of DNNs, convolutional neural networks (CNNs) have been successfully applied in many NLP tasks [31]. Here, we design a new network (named *text_inception*) based on inceptionV2 [32] to extract feature vectors. Fig. 2 shows the structure of the *text_inception* network.

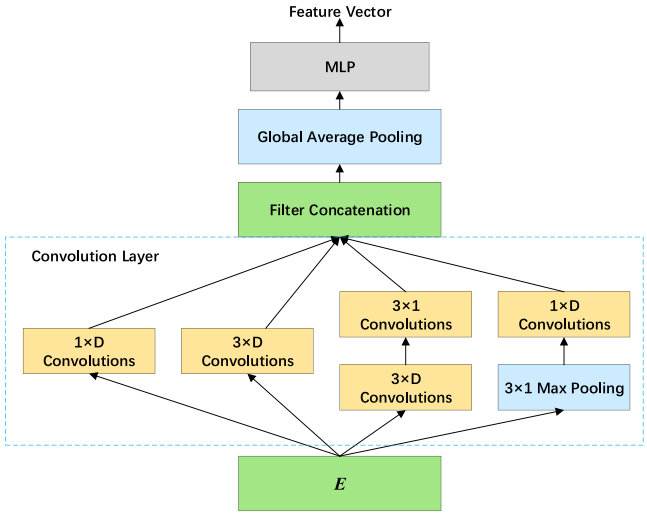


Fig. 2. Structure of the *text_inception* network.

The convolution layer in Fig. 2 captures feature maps of different scales from E by using parallel convolution kernels of different sizes. We first introduce how the convolution operation works on sequential data. Inspired by Kim's work [31], we use convolution kernel j with shared weight $W^j \in \mathbb{R}^{ws \times D}$ to extract a local feature c_i^j from ws terms adjacent to t_i. The process is described as

$$c_i^j = f\left(W^j * E_{(i:(i+ws-1),:)} + b^j\right) \tag{2}$$

where $E_{(i:(i+ws-1),:)}$ is a $ws \times D$ submatrix of E formed by rows from i to $i+ws-1$, $*$ denotes a convolution operation, b^j is a bias term corresponding to W^j, and f is a rectified linear unit (ReLU) function that can avoid the gradient disappearance problem and encourage sparse activations [29]. We apply the convolution kernel j to different locations in the sequence, and then, get a feature map $\mathbf{c}^j \in \mathbb{R}^L$.

$$\mathbf{c}^j = \left[c_1^j, c_2^j, \ldots, c_i^j, \ldots, c_L^j\right]. \tag{3}$$

In the convolution layer, the first two branches utilize $1 \times D$ and $3 \times D$ convolution kernels, respectively. The third branch first executes a convolution on E with $3 \times D$ kernels, and then, executes another convolution on this result with 3×1 kernels, increasing the nonlinearity of the network and improving feature abstraction. Moreover, it can reduce the number of model parameters as well as the risk of overfitting. In the fourth branch, a max-pooling operation with size 3×1 is first carried out, followed by a convolution operation with $1 \times D$ kernels. The max-pooling helps the convolution operation obtain higher level features. After the last convolution operations in all the four branches end, E is compressed into a feature map $\mathbf{c}^j \in \mathbb{R}^L$. These feature maps are then concatenated in the concatenation layer for subsequent operations.

$$C = \left[\mathbf{c}^1, \mathbf{c}^2, \ldots, \mathbf{c}^j, \ldots, \mathbf{c}^F\right]^T \tag{4}$$

where F is the sum of all the convolution kernels in the last convolution layer in the four branches.

A global average pooling (GAP) layer, followed by the inceptionV2 [32], is applied to the pooling of C. We then utilize an MLP to process the pooling results and obtain a dense feature vector. The process can be expressed as

$$l_1 = f\left(W_1^T\left[c_{\mathrm{GAP}}^1, c_{\mathrm{GAP}}^2, \ldots, c_{\mathrm{GAP}}^i, \ldots, c_{\mathrm{GAP}}^F\right]^T + b_1\right) \tag{5}$$

$$l_i = f\left(W_i^T l_{i-1} + b_i\right) \quad (i = 2,\ 3, \ldots, n) \tag{6}$$

where c_{GAP}^i denotes the GAP result of a feature map $\mathbf{c}^i$; W_i^T and b_i denote the weights and bias parameter of the ith layer, respectively; and l_i denotes the output of the ith hidden layer in the MLP.

For convenience, we use MLP to represent all the operations contained in the MLPs in this article. Equations (5) and (6) can be simplified as

$$\mathbf{v}_{\mathrm{seq}} = \mathrm{MLP}\left[c_{\mathrm{GAP}}^1, c_{\mathrm{GAP}}^2, \ldots, c_{\mathrm{GAP}}^i, \ldots, c_{\mathrm{GAP}}^F\right]^T. \tag{7}$$

Particularly, $\mathbf{v}_{\mathrm{seq}_m}$ and $\mathbf{v}_{\mathrm{seq}_s}$ represent the features extracted from the word sequence information of m and s, respectively.

2) Feature Extraction From Separate Word Sets: Unlike natural language sequences, the tags of mashups or services are represented in the form of a separate word set. We cannot apply the *text_inception* network and other deep-learning-based techniques designed for word sequences to the feature extraction of tags. For word set T, we retrieve and average the embeddings of all words to obtain its feature vector of fixed size, $\mathbf{v}_{\mathrm{set}}$.

$$\mathbf{v}_{\mathrm{set}} = \mathrm{average}\left[\mathbf{e}_{T_1} \ldots \mathbf{e}_{T_i} \ldots \mathbf{e}_{T_x}\right] \tag{8}$$

where $\mathbf{e}_{T_i}$ is the embedding of the ith term in the set and x is the size of the set.

3) Content Interaction Learning: After the feature vectors of mashup m and service s, denoted by $\mathbf{v}_{\mathrm{seq}_m}$, $\mathbf{v}_{\mathrm{seq}_s}$, $\mathbf{v}_{\mathrm{set}_m}$, and $\mathbf{v}_{\mathrm{set}_s}$, are extracted by the *text_inception* network and average pooling, respectively, they are concatenated together and fed into an MLP to learn their functional (or content) interactions. Finally, a low-dimensional content interaction vector, $\mathbf{ci}_{ms}$, can be obtained as

$$\mathbf{ci}_{ms} = \mathrm{MLP}_{\mathrm{CI}}\left(\mathbf{v}_{\mathrm{seq}_m} \oplus \mathbf{v}_{\mathrm{seq}_s} \oplus \mathbf{v}_{\mathrm{set}_m} \oplus \mathbf{v}_{\mathrm{set}_s}\right) \tag{9}$$

where $\oplus$ denotes the concatenation operation.

C. Neighbor Interaction Component

The NI component aims to learn the interactions between m and s based on the usage history of the neighbor mashups of m to s. An essential work of the NI component is to find neighbor mashups NM for new mashup m based on their content similarities. When calculating the content similarity between a neighbor mashup nm_i and m, we compute the similarity between word sequences and the similarity between separate word sets, respectively, and then, integrate the two similarities. Since the word sequence features of m and nm_i are real-valued vectors, we calculate their similarity, $\mathrm{sim}_{\mathbf{v}_{\mathrm{seq}_m}, \mathbf{v}_{\mathrm{seq}_{nm_i}}}$, using the commonly used Cosine similarity.

$$\mathrm{sim}_{\mathbf{v}_{\mathrm{seq}_m}, \mathbf{v}_{\mathrm{seq}_{nm_i}}} = \frac{\mathbf{v}_{\mathrm{seq}_m} \cdot \mathbf{v}_{\mathrm{seq}_{nm_i}}}{\|\mathbf{v}_{\mathrm{seq}_m}\| \|\mathbf{v}_{\mathrm{seq}_{nm_i}}\|}. \tag{10}$$

The similarity between the separate word set of m and that of nm_i, $\mathrm{sim}_{\mathbf{v}_{\mathrm{set}_m}, \mathbf{v}_{\mathrm{set}_{nm_i}}}$, is computed in the same way. The weighted sum of the two similarities is regarded as the content similarity between m and nm_i. In this article, a and b are not set to fixed values, and they act as learnable parameters in the process of model training.

$$\mathrm{sim}_{m,nm_i} = a \times \mathrm{sim}_{\mathbf{v}_{\mathrm{seq}_m}, \mathbf{v}_{\mathrm{seq}_{nm_i}}} + b \times \mathrm{sim}_{\mathbf{v}_{\mathrm{set}_m}, \mathbf{v}_{\mathrm{set}_{nm_i}}}. \tag{11}$$

We select K most similar mashups of m to build its neighbor mashups NM in terms of their content similarities. Next, the interactions between neighbor mashups NM and s are leveraged using the following two strategies.

1) Implicit Neighbor Interaction: When modeling the interactions between mashups and services based on their historical invocations, a general framework is to map mashups and services into the same feature space and then define or learn their interactions in this unified space. For example, MF models first use latent factors to represent mashups and services, and then, utilize their inner product to model their interactions. Some deep-learning-based models, such as NCF, use an MLP to process the latent representations of mashups and services and capture their complex interactions.

Similarly, in this part, we first apply node2vec to learn the latent representation of all existing mashups and services from the mashup-service invocation matrix. In the following step, an intuitive strategy is to employ multiple MLPs to learn the interaction between each neighbor mashup of m and s. Then, we integrate these interactions to learn the interaction between m and s with another MLP. However, the computational complexity of this strategy is too high. Therefore, we adopt a feasible strategy: we calculate a weighted representation of mashup m in the same feature space and use an MLP to capture the interaction of m and s in this feature space.

As a graph embedding method, node2vec has achieved remarkable results in processing graph data with plenty of interactive information among elements [33]. Because the mashup-service invocation matrix can be transformed into a graph where nodes denote mashups/services and edges represent invocations between them, it is feasible to use node2vec to learn low-dimensional representations of mashups and services.

An optimized random walk strategy is used to generate node sequences according to the graph structure derived from the mashup-service invocation matrix. Then, we process the node sequences by the skip-gram model and learn the representation of each node. Compared with the MF-based approach, node2vec captures more complex interactions between mashups and services. After obtaining representations of all existing mashups and services, we calculate the weighted representation of mashup m using the following equation:

$$\mathbf{r}_m = \sum_{nm_i \in NM} \mathrm{sim}_{m,nm_i} \cdot \mathbf{r}_{nm_i} \tag{12}$$

where nm_i is a neighbor mashup of m, sim_{m,nm_i} is the content similarity between m and nm_i, and $\mathbf{r}_{nm_i}$ is the representation of nm_i obtained by node2vec.

Finally, we concatenate the representations of m and candidate service s, and then, compress the concatenation result into an implicit interaction vector by an MLP.

$$\mathbf{ini}_{ms} = \mathrm{MLP}_{\mathrm{INI}}\left(\mathbf{r}_m \oplus \mathbf{r}_s\right) \tag{13}$$

where $\mathbf{r}_s$ denotes the representation of s obtained by node2vec.

2) Explicit Neighbor Interaction: In the second strategy, we learn explicit interactions from the direct invocation history of the neighbor mashups (NM) of m to s. More specifically, we first construct a sparse binary vector $\mathbf{s}_{ms}$ according to the invocation records of NM to s, described as follows:

$$\mathbf{s}_{ms} = \left(I_{nm_1,s}, I_{nm_2,s}, \ldots I_{nm_i,s}, \ldots I_{nm_K,s}\right) \tag{14}$$

where $I_{nm_i,s}$ is an identity signal that indicates whether s is a component service of nm_i ($nm_i \in NM$). Since the number of neighbor mashups is K, the dimension number of $\mathbf{s}_{ms}$ is also K.

Finally, $\mathbf{s}_{ms}$ is fed directly into an MLP, and the explicit interaction between m and s, $\mathbf{eni}_{ms}$, can be learned from the direct invocation history of NM to s.

$$\mathbf{eni}_{ms} = \mathrm{MLP}_{\mathrm{ENI}}\left(\mathbf{s}_{ms}\right). \tag{15}$$

3) Combination of the Two Components: After learning the three types of interactions between m and s, we design an MLP to incorporate them and output the rating of s over m, $\hat{r}_{ms}$. The MLP enables the three interactions to enhance and complement each other and model the relationship between m and s more accurately. Because our approach outputs the possibility of m invoking s or the rating of s over m ($\hat{r}_{ms}$), we adopt a sigmoid activation function in the last layer of this MLP to constrain rating values ranging between 0 and 1 for the implicit feedback recommendation task.

$$\hat{r}_{ms} = \mathrm{MLP}_{\mathrm{fusion}}\left(\mathbf{ci}_{ms} \oplus \mathbf{ini}_{ms} \oplus \mathbf{eni}_{ms}\right). \tag{16}$$

D. Offline Model Learning

Since the goal of our approach is to predict the rating of a service over a mashup, each training sample as an input to MISR consists of a mashup and a service. A positive sample (labeled as 1) is composed of a mashup and its component service, while a negative sample (labeled as 0) is a pair of a mashup and an irrelevant service without actual invocations.

The predicted value of MISR should be approximate to 1 for positive samples and 0 for negative samples. The likelihood function is defined as

$$P\left(Y^+, Y^-|\Theta\right) = \prod_{(m,s)\epsilon Y^+} \hat{r}_{ms} \prod_{(m,s)\epsilon Y^-} \left(1 - \hat{r}_{ms}\right) \tag{17}$$

where $\hat{r}_{ms}$ is the predicted rating of service s over mashup m, Θ is the parameter set, Y^+ represents a set of positive samples, and Y^- denotes a set of negative ones.

Maximizing the likelihood probability of (17) is equivalent to minimizing the loss function described as follows:

$$J = -\sum_{(m,s)\in Y^+\cup Y^-} \left(r_{ms}\log\hat{r}_{ms} + \left(1 - r_{ms}\right)\log \times \left(1 - \log\hat{r}_{ms}\right)\right) \tag{18}$$

Algorithm 1: Training Algorithm of MISR.

Input: positive sample set Y^+, negative sample set Y^-, mashup-service invocation matrix MS, and number of epochs p
Output: parameter set Θ
//Model preparation
1. $Y \leftarrow Y^+ \cup Y^-$;
2. $M \leftarrow FindMashup(Y)$, $S \leftarrow FindService(Y)$;
3. **for** each mashup m in M and each service s in S **do**
4. $\mathbf{r}_s \leftarrow node2vec(MS)$, $\mathbf{r}_m \leftarrow node2vec(MS)$;
5. **for** each word t_i in $m.ct$ and each word t_i in $s.ct$ **do**
6. Initialize $\mathbf{e}_{t_i}$ by the pretrained glove model;
7. **end for**
8. **end for**

//Model training
9. $\Theta_{\mathrm{CI}} \leftarrow$ call *Algorithm 2* (p, Y);
10. **for** each mashup m in M **do**
11. Compute $\mathbf{v}_{\mathrm{seq}_m}$ and $\mathbf{v}_{\mathrm{set}_m}$ using (7) and (8), respectively, with the pretrained CI component;
12. **end for**
13. $\Theta_{\mathrm{INI}} \leftarrow$ call *Algorithm 3* (p, Y, $\{\mathbf{v}_{\mathrm{seq}_m}\}_{m\in M}$, $\{\mathbf{v}_{\mathrm{set}_m}\}_{m\in M}$, $\{\mathbf{r}_m\}_{m\in M}$, $\{\mathbf{r}_s\}_{s\in S}$);
14. $\Theta_{\mathrm{ENI}} \leftarrow$ call *Algorithm 4* (p, Y, $\{\mathbf{v}_{\mathrm{seq}_m}\}_{m\in M}$, $\{\mathbf{v}_{\mathrm{set}_m}\}_{m\in M}$, MS);
15. $\Theta_{\mathrm{MLP_{fusion}}} \leftarrow$ call *Algorithm 5* (p, Y, Θ_{CI}, Θ_{INI}, Θ_{ENI});
16. Initialize Θ with Θ_{CI}, Θ_{INI}, Θ_{ENI}, and $\Theta_{\mathrm{MLP_{fusion}}}$;
17. Update Θ by fine-tuning the MISR;
18. **return** Θ.

where r_{ms} denotes the label (0 or 1) of a sample that consists of m and s.

Then, we need to use an optimization algorithm to find the parameters that minimize the loss function of our end-to-end deep-learning-based model. Adaptive moment estimation (Adam) [34], an extension of the stochastic gradient descent (SGD), has been widely used in deep learning applications. It designs independent adaptive learning rates for different parameters by calculating the first-order and the second-order moment estimation of the gradient. Besides, Adam can be applied to large-scale datasets and high-dimensional space. Therefore, we select Adam as our optimization algorithm to update model parameters in this article.

The whole training process is depicted as Algorithm 1.

Lines 1–8 show the preparation for model training. We first employ node2vec to process the invocation matrix and get the vector representation of each mashup and each service. Next, we set the embedding of the words appeared in the content information of mashups and services to be trainable, and use their word embeddings pretrained by the glove model [35] to initialize their corresponding weights in the embedding layer.

The parameters to be optimized in this model mainly include: weight parameters in the embedding layer, parameters in the *text_inception* network, weight parameters used to calculate

Algorithm 2: Training Algorithm of the CI Component.

Input: number of epochs p and sample set Y
Output: parameter set Θ_{CI} of this pretrained component
1. **for** epoch $= 1, \ldots, p$ **do**
2. **for** each sample (m, s) in Y **do**
3. Compute $\mathbf{ci}_{ms}$ using (19);
4. Compute $\hat{r}_{ms}$ using (20);
5. Update Θ_{CI} to minimize J in (18) with Adam;
6. **end for**
7. **end for**
8. **return** Θ_{CI}.

mashup similarities (i.e., a and b), and weight and bias parameters in $\mathrm{MLP}_{\mathrm{CI}}$, $\mathrm{MLP}_{\mathrm{INI}}$, $\mathrm{MLP}_{\mathrm{ENI}}$, and $\mathrm{MLP}_{\mathrm{fusion}}$.

Lines 9–17 demonstrate our training strategy for the MISR. Since MISR is a hierarchical model with several nested MLPs, directly updating all parameters may result in slow convergence. Therefore, we first train each component in MISR separately, then use their parameters to initialize the parameters in MISR, and finally, fine-tune the whole model.

Taking the training algorithm of the CI component, Algorithm 2, as an example, we show how to train an individual component of the MISR. For each sample of (m, s), we only use the CI component to process the content information of m and s (denoted by $m.ct$ and $s.ct$), and then, we obtain a content interaction vector $\mathbf{ci}_{ms}$. Here, the content information of each mashup and service is linked to the corresponding mashup and service. Next, we perform a nonlinear transformation on the vector to directly predict the rating of s over m, $\hat{r}_{ms}$.

$$\mathbf{ci}_{ms} = f_{\mathrm{CI}}\ (m.ct, s.ct) \tag{19}$$

$$\hat{r}_{ms} = f\left(W_{\mathrm{CI}}^{T}\mathbf{ci}_{ms} + b_{\mathrm{CI}}\right) \tag{20}$$

where f_{CI} represents all operations in the CI component, W_{CI} and b_{CI} are the parameters used for the transformation, and f is the sigmoid activation function. Then, we perform backward propagation and update the parameters in the CI component, according to (18)–(20) (see lines 3–5 in Algorithm 2).

Line 3 in both Algorithm 3 and Algorithm 4 aims to find the most similar neighbor mashups for mashup m when training both the INI part and the ENI part. Lines 3–5 in Algorithm 5 indicate that we use the pretrained components to calculate intermediate interaction vectors for each mashup-service instance in sample set Y.

E. Online Prediction and Complexity Analysis

In the online recommendation phase, the MISR predicts the possibility of mashup m invoking service s, and the detailed process is described as follows.

In the CI component, the content information of m and s is transformed into their word embedding form. Then, we extract feature vectors of m and s from their content information by using the *text_inception* network and the average pooling layer, respectively, denoted by $\mathbf{v}_{\mathrm{seq}_m}$, $\mathbf{v}_{\mathrm{seq}_s}$, $\mathbf{v}_{\mathrm{set}_m}$, and $\mathbf{v}_{\mathrm{set}_s}$. Next,

Algorithm 3: Training Algorithm of the INI Part.

Input: number of epochs p, sample set Y, content feature sets of mashups extracted by the pretrained CI component V_{seq} and V_{set}, and latent representation sets of mashups and services by applying node2vec to MS, R_m and R_s
Output: parameter set Θ_{INI} of this pretrained component
1. **for** epoch $= 1, \ldots, p$ **do**
2. **for** each sample (m, s) in Y **do**
3. $NM \leftarrow FindNeighbors(V_{\mathrm{seq}},\ V_{\mathrm{set}})$; //(10)
4. Compute $\mathbf{r}_m$ using (12);
5. Compute $\mathbf{ini}_{ms}$ using (13);
6. $\hat{r}_{ms} = f(W_{\mathrm{INI}}^{T}\mathbf{ini}_{ms} + b_{\mathrm{INI}})$;
7. Update Θ_{INI} to minimize J in (18) with Adam;
8. **end for**
9. **end for**
10. **return** Θ_{INI}.

Algorithm 4: Training Algorithm of the ENI Part.

Input: Number of epochs p, sample set Y, content feature sets of mashups extracted by the pretrained CI component V_{seq} and V_{set}, and mashup-service invocation matrix MS
Output: parameter set Θ_{ENI} of this pretrained component
1. **for** epoch $= 1, \ldots, p$ **do**
2. **for** each sample (m, s) in Y **do**
3. $NM \leftarrow FindNeighbors(V_{\mathrm{seq}},\ V_{\mathrm{set}})$;
4. Construct $\mathbf{s}_{ms}$ using (14) and MS;
5. Compute $\mathbf{eni}_{ms}$ using (15);
6. $\hat{r}_{ms} = f(W_{\mathrm{ENI}}^{T}\mathbf{eni}_{ms} + b_{\mathrm{ENI}})$;
7. Update Θ_{ENI} to minimize J in (18) with Adam;
8. **end for**
9. **end for**
10. **return** Θ_{ENI}.

Algorithm 5: Training Algorithm of $\mathrm{MLP}_{\mathrm{fusion}}$.

Input: number of epochs p, sample set Y, and parameter sets Θ_{CI}, Θ_{INI}, and Θ_{ENI}
Output: parameter set $\Theta_{\mathrm{MLP}_{\mathrm{fusion}}}$
1. **for** epoch $= 1, \ldots, p$ **do**
2. **for** each sample (m, s) in Y **do**
3. Compute $\mathbf{ci}_{ms}$ using (9) with Θ_{CI};
4. Compute $\mathbf{ini}_{ms}$ using (13) with Θ_{INI};
5. Compute $\mathbf{eni}_{ms}$ using (15) with Θ_{ENI};
6. Compute $\hat{r}_{ms}$ using (16);
7. Update $\Theta_{\mathrm{MLP}_{\mathrm{fusion}}}$ to minimize J in (18) with Adam;
8. **end for**
9. **end for**
10. **return** $\Theta_{\mathrm{MLP}_{\mathrm{fusion}}}$.

these features vectors are input into an MLP to obtain a content interaction vector $\mathbf{ci}_{ms}$. For simplicity, in the convolutional layer of the *text_inception* network, we assume that the channel numbers of feature maps in each branch are the same, and the channel numbers in the input and output of each convolution layer are also the same, denoted as C. Thus, the complexity of the convolutional layer is $O(8L \times D \times C^2 + 3L \times C^2 + 3D \times C)$, where L is the length of a word sequence or a word set and D is the dimension of word embeddings. The complexity of the GAP layer in the *text_inception* network is $O(4C \times L)$, and that of the average pooling layer to process separate word sets is $O(D \times L)$.

After getting the feature vectors of mashup m, $\mathbf{v}_{\mathrm{seq}_m}$ and $\mathbf{v}_{\mathrm{set}_m}$, we calculate its content similarity to the existing mashups and obtain the neighbor mashup set NM. The complexity of this processing is $O(P(H + D) + P \log K)$, where H is the dimension of $\mathbf{v}_{\mathrm{seq}_m}$, D is the dimension of $\mathbf{v}_{\mathrm{set}_m}$ as well as that of word embeddings, P is the number of potential neighbor mashups, K is the size of NM, and $P \log K$ is the cost of searching top K values from a list that has P elements.

Next, in the INI component, we compute the representation of m, $\mathbf{r}_m$, according to (12). The complexity is $O(K \times V)$, where V is the dimension of $\mathbf{r}_m$. We then input $\mathbf{r}_m$ and the representation of s obtained by node2vec into $\mathrm{MLP}_{\mathrm{INI}}$ and get an implicit interaction vector of m and s, $\mathbf{ini}_{ms}$. At the same time, the ENI component constructs an invocation vector of NM to s using (15), then inputs it into $\mathrm{MLP}_{\mathrm{ENI}}$, and finally, learns an explicit interaction vector of m and s, $\mathbf{eni}_{ms}$.

Finally, $\mathrm{MLP}_{\mathrm{fusion}}$ integrates multiple forms of interactions between m and s, i.e., $\mathbf{ci}_{ms}$, $\mathbf{ini}_{ms}$, and $\mathbf{eni}_{ms}$, and predicts the possibility of m invoking s.

The complexity of MLPs, including $\mathrm{MLP}_{\mathrm{CI}}$, $\mathrm{MLP}_{\mathrm{INI}}$, $\mathrm{MLP}_{\mathrm{ENI}}$, $\mathrm{MLP}_{\mathrm{fusion}}$, and the MLP in the *text_inception* network, is $O(\sum_{i=1}^{N} n_{i-1} n_i)$, where N is the number of layers in each of the MLPs and n_i is the number of units in the ith layer. Note that we do not consider bias parameters for simplicity.

To sum up, the simplified complexity of predicting the rating of a candidate service over a mashup is $O(L \times D \times C^2 + P(H + D + \log K) + K \times V + \sum_{i=1}^{N} n_{i-1} n_i)$. If the structure of MLPs is fixed, parameters L, D, C, H, and V can be regarded as certain constants. Therefore, the complexity can be rewritten as $O(P \log K + K)$.

After predicting the rating of each candidate service over a mashup, the recommender system outputs the Top-K services with the highest ratings for the target developer. Therefore, the complexity of the whole online recommendation process is $O(Q(P \log K + K) + Q\log Q) = O(Q(P \log K + K + \log Q))$, where Q is the number of candidate services and $Q\log Q$ is the cost of sorting Q elements using the quick sort algorithm.

IV. Experimental Setups and Results

A. Experimental Settings

All experiments were carried out on a workstation with Intel Core 8 Xeon(R) at 3.50 GHz, GeForce GTX 1080, and 32-GB memory, running the Ubuntu 16.04 operating system. The source code implemented based on Keras[2] is available on GitHub.[3]

1) Dataset: We crawled a dataset from ProgrammableWeb, the largest online Web service registry, on July 25, 2016. The mashups and services without functional descriptions, the services that have not been invoked, and the mashups with fewer than two component services were removed from the original dataset. The experimental dataset contains 1979 mashups and 728 services, and the sparsity of the mashup-service invocation matrix is 99.6%. We preprocessed the textual descriptions of these mashups and services by removing punctuation and stop words.

The functional descriptions of the mashups in the test set were used as textual requests to build new mashups, and the recommended lists of services were compared with the actual component services of the mashups for evaluation. Also, we randomly generated some negative samples (i.e., a pair of a mashup and a service without invocation relations), which are six times as many as positive samples, to construct our training dataset.

2) Evaluation Metrics: In this article, we evaluated different recommendation approaches using the fivefold cross-validation technique. The 1979 mashups were divided into fivefolds. In each time, onefold was used for testing, and the others for training. Then, we averaged the results of fivefolds and took them as the final result. We adopted the following evaluation metrics to measure the recommendation performance.

Precision, Recall, and F1-measure at top *N* services in the ranking list are defined as

$$\mathrm{Precision@}N = \frac{1}{|M|} \sum_{m \in M} \frac{|\mathrm{rcmd}\,(m) \cap \mathrm{actual}\,(m)|}{|\mathrm{rcmd}\,(m)|} \tag{21}$$

$$\mathrm{Recall@}N = \frac{1}{|M|} \sum_{m \in M} \frac{|\mathrm{rcmd}\,(m) \cap \mathrm{actual}\,(m)|}{|\mathrm{actual}\,(m)|} \tag{22}$$

$$F1@N = \frac{1}{|M|} \sum_{m \in M} 2 \frac{|\mathrm{rcmd}\,(m) \cap \mathrm{actual}\,(m)|}{|\mathrm{rcmd}\,(m)| + |\mathrm{actual}\,(m)|} \tag{23}$$

where M is the set of mashups in the test set and $|M|$ denotes the size of M. For mashup m, $\mathrm{rcmd}(m)$ is the recommended service list, while $\mathrm{actual}\ (m)$ is its actual component services.

Mean average precision (MAP) at top *N* services in the ranking list is defined as

$$\mathrm{MAP@}N = \frac{1}{|M|} \sum_{m \in M} \frac{1}{N_m} \sum_{i=1}^{N} \left(\frac{N_i}{i} \times I\,(i) \right) \tag{24}$$

where I_i indicates whether a service at the position i in the ranking list is an actual component service of m, N_m is the number of component services of m, and N_i denotes the number of actual component services of m occurred in the top i services of the ranking list.

[2]https://keras.io
[3]https://github.com/ssea-lab/MISR

Normalized discounted cumulative gain (NDCG) at top N services in the ranking list is defined as

$$\mathrm{NDCG@}N = \frac{1}{|M|}\sum_{m\in M}\frac{1}{S_m}\sum_{i=1}^{N}\frac{2^{I(i)}-1}{\log_2(1+i)} \tag{25}$$

where S_m represents the ideal maximum DCG score that can be achieved for m.

3) Baseline Approaches: Most of the previous works mentioned in Section II-C cannot work in the scenario of this article, i.e., recommending services to a new mashup. To evaluate the effectiveness of our approach, we selected six state-of-the-art service recommendation approaches that can work in the scenario for comparison.

1) *Term Frequency–Inverse Document Frequency (TF-IDF)* [10]. The method recommends services using TF-IDF-based cosine similarities between content information of services and a mashup.
2) *Aggregating Functionality, Use history, and Popularity of APIs (AFUP)* [11]. The approach first computes two probabilities that a mashup invokes a service by analyzing their content similarity and the usage history of neighbor mashups and the service, then multiplies them based on Bayes' theorem, and finally, ranks candidates according to their popularity.
3) *Recommendation through Service Factors and Top-K Neighbors (SFTN)* [12]. The authors improve their previous work, the AFUP, by using the hierarchical Dirichlet process (HDP) [36] and probability matrix factorization to process the content information and usage history.
4) *Preference-based Neural Collaborative Filtering Recommender (PNCF)* [14]. The framework compresses all sparse features of users and items in an embedding layer, and then, uses an MLP to model their interaction. However, its feature extraction component does not apply to extract textual features, and we implement two variants for this scenario: PNCF-HDP, which applies the HDP adopted in the SFTN, and PNCF-Deep, which uses our feature extraction strategy.
5) *Service Set Recommendation (SSR)* [37]. The approach clusters services according to their functionalities, and then, constructs service sets. Finally, the service set with the highest utility function score (considering the composability, functional similarity, and popularity) is recommended.

Note that all baseline approaches and the MISR take descriptions and tags as the content information. These two kinds of content information are processed indiscriminately by bag-of-words models in the TF-IDF, AFUP, SFTN, SSR, and PNCF- HDP, while being processed separately by deep learning techniques in the PNCF-Deep and MISR. Moreover, the parameters of the feature extractors and other parameters in the model are jointly trained in the PNCF-Deep and MISR.

4) Parameter Settings: We set the dimension of word vectors to 50 and initialized the vectors with the publicly available 50-dimensional word embeddings trained by the Glove model [35]. The filter numbers in the four branches of the *text_inception* network were set to 10, 10, 20, and 10, respectively. In node2vec, the dimension of each node was set to 25, and other parameters were set according to [30]. In $\mathrm{MLP}_{\mathrm{fusion}}$, the unit numbers of the four layers were set to 128, 64, 32, and 1, respectively. The other MLPs used in the model shared the same structure, where the numbers of units in two layers were set to 100 and 50, respectively. Except for the MLP in the *text_inception* network that selected PReLU [38] as the activation function, other MLPs in the model used ReLU. The learning rate was set to 0.0003 when training each component of the MISR. When we began to fine-tune the MISR based on all the pretrained components, the learning rate was then reduced to 0.0001. For the baseline approaches, we set most of their parameters according to the default settings mentioned in the original references and optimized some parameters when necessary.

B. Performance of MISR

Fig. 3 presents the performance comparison of different approaches, showing that the MISR outperforms all the six baselines across all ranking positions. Since the complexity of the SSR is exceptionally high, we only evaluated its performance when the size of service sets is five.

The TF-IDF performed the worst because it just used the content information, and the representations of mashups and services did not appear to capture the latent semantics of textual descriptions by using the TF-IDF. Besides the content information and popularity, the AFUP and SFTN leverage the usage history of neighbor mashups, but their performances were not as good as expected. The reasons are twofold. First, their feature extraction methods ignored word orders and lost some semantic information. Second, the two probabilities derived from the functionality and usage history were multiplied with the assumption that they are conditionally independent of each other. However, it is hard for the single multiplication to capture the way how the content information and usage history jointly affect service recommendation. Instead, the MISR aims to capture such complicated interactions by a DNN.

We combined different parameter settings to optimize the SSR, but its performance was still weak. There are two main reasons. First, it did not utilize the usage history of neighbor mashups like the MISR. Second, the pruning strategy it employed reduces the probability of the recommended service set hitting actual component services.

The PNCF-Deep performed the best in the six baseline methods. The possible reason is that the PNCF-Deep extracts high-quality features from the content information and learns the deep content interaction between mashups and services. The PNCF-HDP worked worse than the PNCF-Deep, even though it also used an MLP to learn the content interaction. The result indicates the superiority of our strategy for textual feature extraction.

Although the MISR shares a similar content interaction component with the PNCF-Deep, the NDCG@5, MAP@5, Precision@5, Recall@5, and *F1*@5 values of the MISR were higher than those of the PNCF by 15.66%, 19.85%, 16.02%, 15.12%, and 15.63%, respectively. These performance improvements

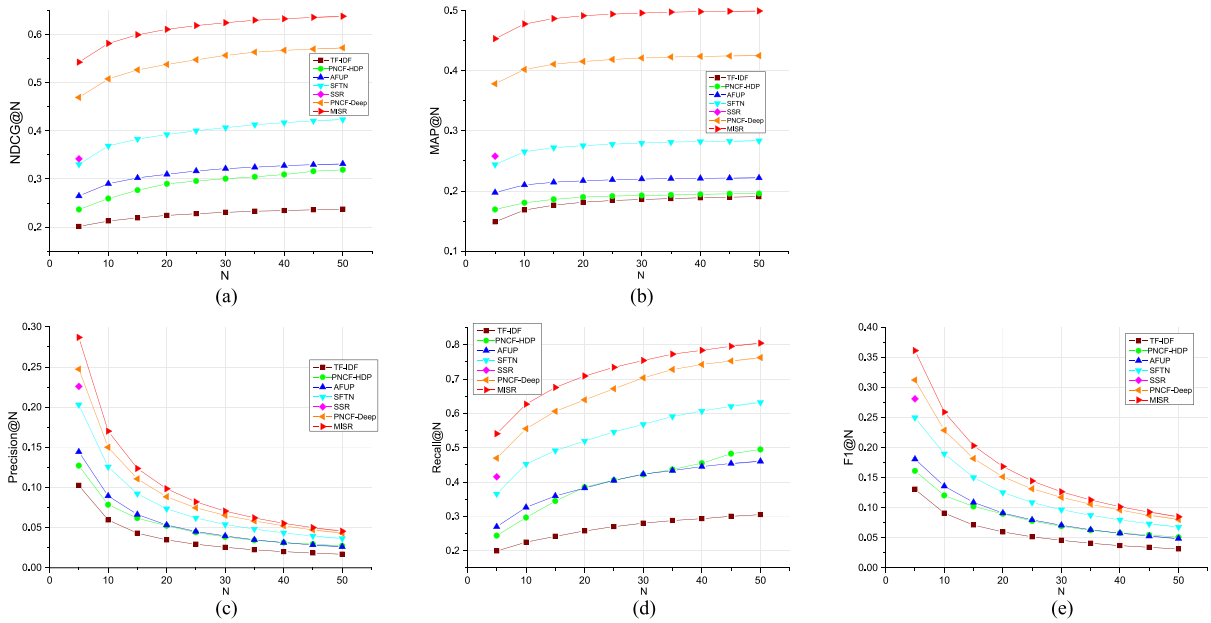


Fig. 3. Performance comparison of different approaches. (a) NDCG@N. (b)MAP@N. (c) Precision@N. (d) Recall@N. (e) F1@N.

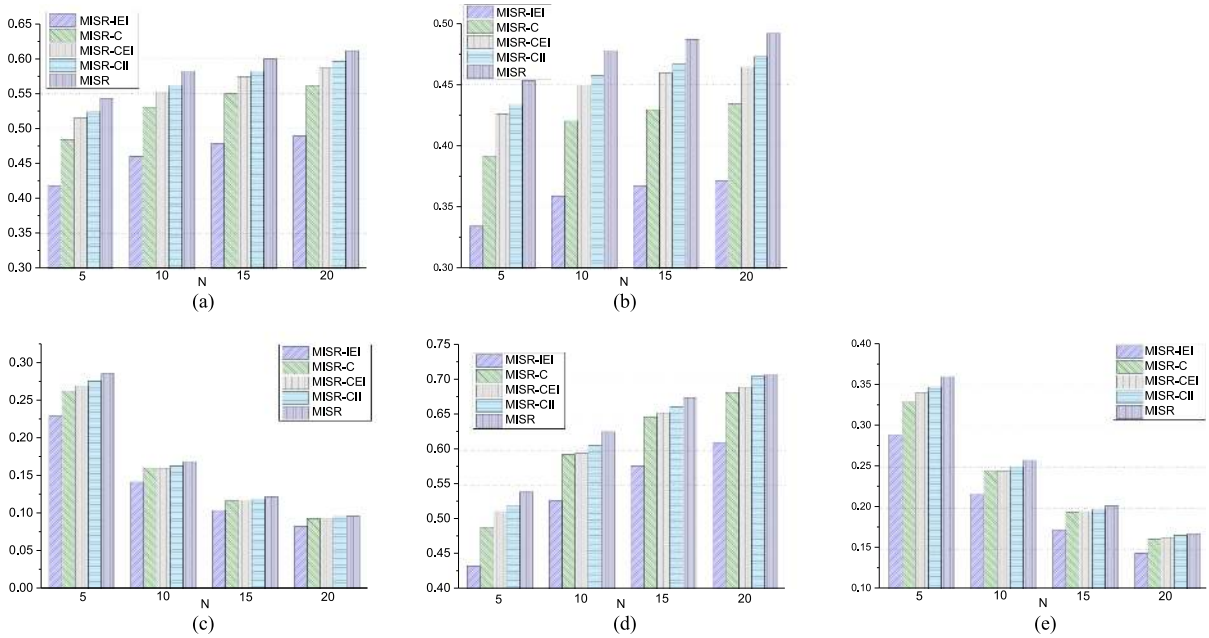


Fig. 4. Performance comparison of different variants of MISR. (a) NDCG@N. (b) MAP@N. (c) Precision@N. (d) Recall@N. (e) F1@N.

mainly benefit from the two elaborate NI parts that can exploit the past interactions between neighbor mashups and services.

C. Ablation Study

As mentioned previously, the MISR utilizes three types of interactions between mashups and services, namely the content interaction and two neighbor interactions. To demonstrate the necessity of the three components of the MISR, we designed the following variants of the MISR for comparison: a model (referred to as MISR-C) only uses the CI component, a model (referred to as MISR-CII) consists of the CI and INI components, a model (referred to as MISR-CEI) has the CI and ENI components, and a model (referred to as MISR-IEI) composed of the INI and ENI components.

According to the comparison among the variants of our approach shown in Fig. 4, the more interactions are involved, the better the recommendation performance can we obtain. Both the MISR-CEI and MISR-CII outperformed the MISR-C, suggesting that both the ENI component and the INI component can indeed learn useful interaction information. Moreover, the MISR performed better than the MISR-CEI and MISR-CII, which

TABLE II
PERFORMANCE OF DIFFERENT CONTENT SIMILARITY COMPUTATION METHODS (MEAN ± S. D.)

Methods	*NDCG@5*	*MAP@5*	*Precision@5*	*Recall@5*	*F1@5*
DeepText+DeepTag	**0.528**±(0.008)	**0.439**±(0.009)	**0.277**±(0.009)	**0.521**±(0.015)	**0.349**±(0.010)
HDPText+DeepTag	0.490±(0.024)	0.398±(0.029)	0.267±(0.002)	0.496±(0.016)	0.334±(0.003)
DeepText+metaPathTag	0.448±(0.020)	0.354±(0.023)	0.247±(0.010)	0.457±(0.016)	0.309±(0.010)
HDPText+metaPathTag	0.419±(0.081)	0.331±(0.078)	0.225±(0.050)	0.418±(0.087)	0.281±(0.061)
Methods	*NDCG@10*	*MAP@10*	*Precision@10*	*Recall@10*	*F1@10*
DeepText+DeepTag	**0.565**±(0.006)	**0.463**±(0.007)	**0.165**±(0.006)	**0.603**±(0.010)	**0.250**±(0.008)
HDPText+DeepTag	0.526±(0.021)	0.420±(0.026)	0.158±(0.003)	0.578±(0.010)	0.240±(0.003)
DeepText+metaPathTag	0.491±(0.017)	0.381±(0.021)	0.151±(0.007)	0.555±(0.020)	0.230±(0.010)
HDPText+metaPathTag	0.447±(0.100)	0.348±(0.089)	0.130±(0.039)	0.481±(0.136)	0.199±(0.059)
Methods	*NDCG@15*	*MAP@15*	*Precision@15*	*Recall@15*	*F1@15*
DeepText+DeepTag	**0.584**±(0.007)	**0.472**±(0.008)	**0.120**±(0.004)	**0.653**±(0.013)	**0.197**±(0.006)
HDPText+DeepTag	0.544±(0.022)	0.429±(0.026)	0.116±(0.002)	0.627±(0.014)	0.189±(0.002)
DeepText+metaPathTag	0.509±(0.017)	0.389±(0.020)	0.111±(0.006)	0.602±(0.025)	0.182±(0.009)
HDPText+metaPathTag	0.462±(0.103)	0.356±(0.091)	0.096±(0.028)	0.523±(0.144)	0.158±(0.046)
Methods	*NDCG@20*	*MAP@20*	*Precision@20*	*Recall@20*	*F1@20*
DeepText+DeepTag	**0.595**±(0.006)	**0.476**±(0.007)	**0.096**±(0.002)	**0.687**±(0.01)	**0.164**±(0.004)
HDPText+DeepTag	0.557±(0.024)	0.434±(0.027)	0.093±(0.002)	0.665±(0.017)	0.159±(0.003)
DeepText+metaPathTag	0.519±(0.017)	0.394±(0.02)	0.088±(0.004)	0.634±(0.023)	0.151±(0.007)
HDPText+metaPathTag	0.474±(0.103)	0.360±(0.092)	0.077±(0.021)	0.559±(0.143)	0.132±(0.036)

indicates that each of the two neighbor interaction components can learn some complementary information missed by the other.

Compared with the MISR, the MISR-IEI ablates the CI component and outputs worse recommendation results. Also, the prediction performance of the MISR-IEI is inferior to that of the MISR-C. This result indicates that in the process of developing new mashups, developers pay more attention to the degree of functionality matching between candidate services and their requests, though the interaction experience obtained from neighbor mashups is also beneficial to the development process.

Finally, the MISR-C performed far better than the TF-IDF. This result demonstrates that the MISR does extract useful text features from the content information of mashups and services and can capture their functional interactions by an MLP.

D. Selection of Neighbor Mashups

In this article, a prerequisite of neighbor interactions is to find neighbor mashups with similar requests to the target mashup. More specifically, the content similarity between mashups and the size of neighbor mashups are two critical factors in searching for neighbor mashups. Therefore, we conducted two experiments to investigate their respective impact on the selection of neighbor mashups in this subsection.

1) Impact of Content Similarity on the NI Component: Finding similar neighbor mashups in terms of the content similarity between mashups is essential to the performance of the NI component. As introduced in Section III-C, we calculate the similarities between mashup texts and mashup tags, respectively, and then, compute the weighted sum of them to obtain a scalar similarity. In particular, the MISR extracts text features of mashups by the *text_inception* network and takes their Cosine similarity as text similarity (denoted as *DeepText*). The MISR then obtains tag features of mashups using a pooling layer and takes their Cosine similarity as tag similarity (denoted as *DeepTag*). The similarity calculation strategy of the MISR is denoted as *DeepText + DeepTag*.

As we know, there are many alternative ways to calculate the content similarity between mashups (i.e., text similarity and tag similarity). A commonly used way to calculate the text similarity of two documents is to extract their feature vectors by the HDP (a representative of traditional feature extractors) and calculate the Cosine similarity between their HDP features (denoted as *HDPText*) [12]. The simplest and most popular way to calculate tag similarity is the method adopted in [24]–[26] (denoted as *metaPathTag*).

$$metaPathTag\left(m_i, m_j\right) = \frac{2 \times \left|\mathrm{Tag}_{m_i} \cap \mathrm{Tag}_{m_j}\right|}{\left|\mathrm{Tag}_{m_i}\right| + \left|\mathrm{Tag}_{m_j}\right|} \quad (26)$$

where Tag_{m_i} and Tag_{m_j} denote the tag set of mashup m_i and m_j, respectively, and $|\mathrm{Tag}_m|$ is the size of the tag set of mashup m.

To evaluate the impact of the content similarity calculation methods on the NI component's performance, we compared ours (*DeepText + DeepTag*) with three variants: Variant 1 (*DeepText + metaPathTag*), Variant 2 (*HDPText + DeepTag*), and

TABLE III
RECOMMENDATION PERFORMANCE WITH DIFFERENT K (MEAN ± S. D.)

K	*NDCG@5*	*MAP@5*	*Precision@5*	*Recall@5*	*F1@5*
10	0.5301±(0.0152)	0.4398±(0.0153)	0.2803±(0.0101)	0.5294±(0.0149)	0.3532±(0.0108)
20	0.5408±(0.0234)	0.4520±(0.0247)	0.2836±(0.0124)	0.5339±(0.0253)	0.3568±(0.0155)
30	**0.5429**±(0.0202)	**0.4532**±(0.0226)	**0.2869**±(0.0106)	**0.5406**±(0.0238)	**0.3610**±(0.0140)
40	0.5371±(0.0099)	0.4484±(0.0121)	0.2829±(0.0054)	0.5323±(0.0104)	0.3558±(0.0055)
50	0.5347±(0.0192)	0.4437±(0.0208)	0.2824±(0.0105)	0.5322±(0.0224)	0.3553±(0.0130)
K	*NDCG@10*	*MAP@10*	*Precision@10*	*Recall@10*	*F1@10*
10	0.5683±(0.0165)	0.4637±(0.0161)	0.1663±(0.0047)	0.6154±(0.0192)	0.2533±(0.0070)
20	0.5800±(0.0240)	0.4769±(0.0259)	0.1687±(0.0062)	0.6214±(0.0262)	0.2566±(0.0095)
30	**0.5818**±(0.0185)	**0.4777**±(0.0217)	**0.1702**±(0.0040)	**0.6267**±(0.0190)	**0.2589**±(0.0062)
40	0.5753±(0.0104)	0.4719±(0.0123)	0.1678±(0.0027)	0.6175±(0.0108)	0.2552±(0.0035)
50	0.5729±(0.0170)	0.4683±(0.0196)	0.1674±(0.0050)	0.6162±(0.0190)	0.2545±(0.0074)
K	*NDCG@15*	*MAP@15*	*Precision@15*	*Recall@15*	*F1@15*
10	0.5875±(0.0160)	0.4730±(0.0159)	0.1218±(0.0038)	0.6675±(0.0185)	0.2002±(0.0058)
20	0.5985±(0.0230)	0.4861±(0.0257)	0.1228±(0.0041)	0.6716±(0.0235)	0.2019±(0.0066)
30	**0.5997**±(0.0179)	**0.4867**±(0.0217)	**0.1237**±(0.0022)	**0.6754**±(0.0203)	**0.2031**±(0.0039)
40	0.5946±(0.0092)	0.4816±(0.0118)	0.1229±(0.0023)	0.6692±(0.0070)	0.2016±(0.0031)
50	0.5928±(0.0177)	0.4782±(0.0201)	0.1226±(0.0038)	0.6704±(0.0216)	0.2014±(0.0061)
K	*NDCG@20*	*MAP@20*	*Precision@20*	*Recall@20*	*F1@20*
10	0.6001±(0.0172)	0.4782±(0.0166)	0.0975±(0.0033)	0.7048±(0.0229)	0.1670±(0.0056)
20	**0.6113**±(0.0227)	0.4913±(0.0256)	0.0983±(0.0031)	**0.7101**±(0.0228)	**0.1684**±(0.0054)
30	0.6111±(0.0177)	**0.4914**±(0.0216)	**0.0984**±(0.0020)	0.7088±(0.0202)	**0.1684**±(0.0036)
40	0.6077±(0.0079)	0.4868±(0.0112)	0.0982±(0.0021)	0.7094±(0.0131)	0.1682±(0.0035)
50	0.6042±(0.0172)	0.4829±(0.0199)	0.0975±(0.0025)	0.7045±(0.0216)	0.1670±(0.0043)

Variant 3 (*HDPText* + *metaPathTag*). Note that we replaced the content similarity method in the MISR with the three variants, respectively, to compare their performance.

As shown in Table II, our method (*DeepText* + *DeepTag*) performs the best, followed by *HDPText* + *DeepTag*, and *DeepText* + *metaPathTag* outperforms *HDPText* + *metaPathTag*. This result indicates that our deep-learning-based feature extractor, the *text-inception* network, performed better than the HDP in extracting feature vectors for the NI component. One possible reason is that the CNN-based feature extractor can extract text features with richer semantics. Another reason lies in the effective combination between the *text-inception* network and other components of the MISR. This combination enables the *text-inception* network to generate task-specific text features for the NI component.

It is evident from Table II that *DeepText*+*DeepTag* outperforms *DeepText*+*metaPathTag*, and *HDPText*+*DeepTag* performs better than *HDPText*+*metaPathTag*. Compared with *metaPathTag*, our *DeepTag* can extract better tag features to facilitate the identification of similar neighbor mashups for the NI component. The meta-path-based approach regards tags as plain symbols without semantics, while *DeepTag* maps tags into a semantic space using an embedding layer. We initialized the embeddings of some tags with their pretrained word embeddings in [35] and updated the embedding of all tags when training the MISR. Therefore, in this way, *DeepTag* can capture richer semantics of tags than *metaPathTag*.

2) Impact of the Size of Neighbor Mashups: The size of neighbor mashups of the target mashup, K, determines how many similar mashups are involved in the ENI and INI parts. Therefore, the setting of K is a critical factor that affects the performance of our approach. To study the impact of K on the recommendation performance, we set its value from 10 to 50 by step 10, while fixing the other parameters.

As shown in Table III, when K increases from 10 to 30, the performance results of the MISR in terms of the five evaluation metrics are increasing. This result is probably because the increase of the size of neighbor mashups can help our approach

learn the usage history of more similar mashups. Instead, the recommendation performance of the MISR becomes worse when K exceeds 30. The reason probably lies in that some noisy data, i.e., mashups with low similarity to the target mashup, were introduced into the learning of neighbor interactions. Therefore, we set K to 30 in our experiments.

E. Threats to Validity

Some potential factors may threaten the validity of our article, and we discuss them in this subsection.

1) Internal Validity: The internal validity concerns the authenticity of the experimental results. The threats to the internal validity of our article fall into two main aspects: evaluation criterion and parameter settings.

There is no suitable evaluation dataset at present, including actual mashup requirements and component services. In our experiments, the content information of mashups registered at the ProgrammableWeb is regarded as the functional requirements provided by users when developing mashups. However, user requirements in real-life scenarios may differ from mashup descriptions regarding language style and expression pattern, which is a threat to the internal validity of this article. According to our analysis, mashup descriptions provided by different service providers embody the functionalities of mashups and have different description styles, which display high similarities to user requirements. Besides, this evaluation criterion has also been used in experiments of the existing service recommendation approaches [13], [24]. Hence, we argue that this threat to the internal validity of our article is not severe.

Since the source code of most of the baseline approaches is not publically available, we implemented them and used their default parameter settings mentioned in their original papers. There is no guarantee that they have reached their optimal performance stated in the corresponding papers, which is another threat to the internal validity of our article. To mitigate this threat, we asked two master students who are familiar with these approaches to examine our implementation code and optimize them as needed.

2) External Validity: The external validity concerns the generalizability of the experiment results. It is challenging for the dataset used in our experiments to represent all the real-world scenarios accurately. To mitigate this threat, ProgrammableWeb, the largest real-world repository of web APIs, mashups, and applications, was selected as the experimental dataset. The mashups and services provided by over 1000 companies or personal developers make ProgrammableWeb a typical representative of service registries. To further minimize the generalizability issue, we divided the crawled dataset into five groups and performed fivefold cross validation in the experiment. Even so, evaluations of more large-scale real-world datasets are still needed in the future.

The MISR is designed for an utterly cold-start scenario where a user only inputs functional requirements. Last but not least, another threat to the external validity of this article is whether the approach can be generalized into other real scenarios where developers have selected one or more component services. Because the CI and NI components can work in these scenarios, the MISR can still work, and the threat is also not severe. We need to further improve the MISR by leveraging the information of services that have been selected and modeling complex interactions among candidate services, the selected services, and the target mashup. This study will be our future work.

V. Conclusion and Future Work

In this article, we proposed a multiplex interaction-oriented service recommendation approach, referred to as MISR, for developing new mashups without any component service. Three types of interactions between services and mashups were incorporated into a DNN to model their explicit and implicit relationships. Experiments on a real-world dataset demonstrated that the proposed approach was able to outperform several state-of-the-art service recommendation methods regarding five evaluation metrics.

In the future, we plan to improve our approach in the following aspects. First, the MISR is currently designed to recommend services for new mashups without any component. We will extend the model framework to make it applicable to service recommendation scenarios where developers have selected one or more services. Second, we will also consider the composability or composition patterns between services, e.g., two services developed by the same company are more likely to be invoked together.

References

[1] N. Zhang, J. Wang, and Y. Ma, "Mining domain knowledge on service goals from textual service descriptions," *IEEE Trans. Serv. Comput.*, to be published, doi: 10.1109/TSC.2017.2693147.

[2] Q. He *et al.*, "Efficient keyword search for building service-based systems based on dynamic programming," in *Proc. Int. Conf. Serv.-Oriented Comput.*, 2017, pp. 462–470.

[3] Q. He *et al.*, "Keyword search for building service-based systems," *IEEE Trans. Software Eng.*, vol. 43, no. 7, pp. 658–674, Jul. 2017.

[4] M. Al-Hassan, H. Lu, and J. Lu, "A semantic enhanced hybrid recommendation approach: A case study of e-Government tourism service recommendation system," *Decis. Support Syst.*, vol. 72, pp. 97–109, 2015.

[5] L. Yu, J. Zhou, J. Zhang, F. Wei, and J. Wang, "Time-aware semantic web service recommendation," in *Proc. IEEE Int. Conf. Serv. Comput.*, 2015, pp. 664–671.

[6] W. Gao, L. Chen, J. Wu, and A. Bouguettaya, "Joint modeling users, services, mashups, and topics for service recommendation," in *Proc. IEEE Int. Conf. Web Serv.*, 2016, pp. 260–267.

[7] C. Lin, A. K. Kalia, J. Xiao, M. Vukovic, and N. Anerousis, "NL2API: A framework for bootstrapping service recommendation using natural language queries," in *Proc. IEEE Int. Conf. Web Serv.*, 2018, pp. 235–242.

[8] C. Li, R. Zhang, J. Huai, and H. Sun, "A novel approach for API recommendation in mashup development," in *Proc. IEEE Int. Conf. Web Serv.*, 2014, pp. 289–296.

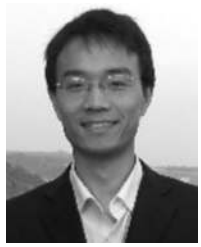
[9] Z. Gao *et al.*, "SeCo-LDA: Mining service co-occurrence topics for composition recommendation," *IEEE Trans. Serv. Comput.*, vol. 12, no. 3, pp. 446–459, May/Jun. 2019.

[10] 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. Serv. Comput.*, vol. 8, no. 5, pp. 674–687, Sep./Oct. 2015.

[11] A. Jain, X. Liu, and Q. Yu, "Aggregating functionality, use history, and popularity of apis to recommend mashup creation," in *Proc. Int. Conf. Serv.-Oriented Comput.*, 2015, pp. 188–202.

[12] P. Samanta and X. Liu, "Recommending services for new mashups through service factors and top-K neighbors," in *Proc. IEEE Int. Conf. Web Serv.*, 2017, pp. 381–388.

[13] R. Xiong, J. Wang, N. Zhang, and Y. Ma, "Deep hybrid collaborative filtering for web service recommendation," *Expert Syst. Appl.*, vol. 110, pp. 191–205, 2018.
[14] L. Chen, A. Zheng, Y. Feng, F. Xie, and Z. Zheng, "Software service recommendation base on collaborative filtering neural network model," in *Proc. Int. Conf. Serv.-Oriented Comput.*, 2018, pp. 288–403.
[15] J. Bobadilla, F. Ortega, A. Hernando, and A. Gutierrez, "Recommender systems survey," *Knowl.-Based Syst.*, vol. 46, pp. 109–132, 2013, doi: 10.1016/j.knosys.2013.03.012.
[16] R. Yera and L. Martínez, "Fuzzy tools in recommender systems: A survey," *Int. J. Comput. Intell. Syst.*, vol. 10, no. 1, pp. 776–803, 2017.
[17] J. Lu, D. Wu, M. Mao, W. Wang, and G. Zhang. "Recommender system application developments: A survey," *Decis. Support Syst.*, vol. 74, pp. 12–32, 2015.
[18] M. Aznag, M. Quafafou, and Z. Jarir, "Leveraging formal concept analysis with topic correlation for service clustering and discovery," in *Proc. IEEE Int. Conf. Web Serv.*, 2014, pp. 153–160, doi: 10.1109/ICWS.2014.33.
[19] Z. Zheng, H. Ma, M. R. Lyu, and I. King, "Collaborative web service QoS prediction via neighborhood integrated matrix factorization," *IEEE Trans. Serv. Comput.*, vol. 6, no. 3, pp. 289–299, Jul./Sep. 2013.
[20] X. Chen, X. Liu, Z. Huang, and H. Sun, "Regionknn: A scalable hybrid collaborative filtering algorithm for personalized web service recommendation," in *Proc. IEEE Int. Conf. Web Serv.*, 2010, pp. 9–16.
[21] J. Liu, M. Tang, Z. Zheng, X. F. Liu, and S. Lyu, "Location-aware and personalized collaborative filtering for web service recommendation," *IEEE Trans. Serv. Comput.*, vol. 9, no. 5, pp. 686–699, Sep./Oct. 2016.
[22] Y. Hu, Q. Peng, X. Hu, and R. Yang, "Time aware and data sparsity tolerant web service recommendation based on improved collaborative filtering," *IEEE Trans. Serv. Comput.*, vol. 8, no. 5, pp. 782–794, Sep./Oct. 2015.
[23] G. Zou, M. Jiang, S. Niu, H. Wu, S. Pang, and Y. Gan, "QoS-aware web service recommendation with reinforced collaborative filtering," in *Proc. Int. Conf. Serv.-Oriented Comput.*, 2018, pp. 430–445, doi: 10.1007/978-3-030-03596-9_31.
[24] F. Xie, J. Wang, R. Xiong, N. Zhang, Y. Ma, and K. He, "An integrated service recommendation approach for service-based system development," *Expert Syst. Appl.*, vol. 123, pp. 178–194, 2019.
[25] T. Liang, L. Chen, J. Wu, H. Dong, and A. Bouguettaya, "Meta-path based service recommendation in heterogeneous information networks," in *Proc. Int. Conf. Serv.-Oriented Comput.*, 2016, pp. 371–386.
[26] F. Xie, L. Chen, D. Lin, Z. Zheng, and X. Lin, "Personalized service recommendation with mashup group preference in heterogeneous information network," *IEEE Access*, vol. 7, pp. 16155–16167, 2019.
[27] D. M. Blei, A. Y. Ng, and M. I. Jordan, "Latent dirichlet allocation," *J. Mach. Learn. Res.*, vol. 3, pp. 993–1022, 2003. [Online]. Available: http://jmlr.csail.mit.edu/papers/v3/blei03a.html.
[28] H. Cheng *et al.*, "Wide & deep learning for recommender systems," in *Proc. Workshop Deep Learn. Recommender Syst.*, 2016, pp. 7–10, doi: 10.1145/2988450.2988454.
[29] X. He, L. Liao, H. Zhang, L. Nie, X. Hu, and T.-S. Chua, "Neural collaborative filtering," in *Proc. Int. Conf. World Wide Web*, 2017, pp. 173–182, doi: 10.1145/3038912.3052569.
[30] A. Grover and J. Leskovec, "node2vec: Scalable feature learning for networks," in *Proc. ACM SIGKDD Int. Conf. Knowl. Discovery Data Mining*, 2016, pp. 855–864, doi: 10.1145/2939672.2939754.
[31] Y. Kim, "Convolutional neural networks for sentence classification," in *Proc. Conf. Empirical Methods Natural Lang. Process.*, 2014, pp. 1746–1751. [Online]. Available: https://www.aclweb.org/anthology/D14-1181
[32] C. Szegedy, V. Vanhoucke, S. Ioffe, J. Shlens, and Z. Wojna, "Rethinking the inception architecture for computer vision," in *Proc. IEEE Conf. Comput. Vision Pattern Recognit.*, 2016, pp. 2818–2826.
[33] P. Goyal and E. Ferrara, "Graph embedding techniques, applications, and performance: A survey," *Knowl.-Based Syst.*, vol. 151, pp. 78–94, 2018, doi: 10.1016/j.knosys.2018.03.022.
[34] D. P. Kingma and J. Ba, "Adam: A method for stochastic optimization," 2015, *arXiv: 1412.6980*, [Online]. Available: https://arxiv.org/abs/1412.6980
[35] J. Pennington, R. Socher, and C. Manning, "Glove: Global vectors for word representation," in *Proc. Conf. Empir ical Methods Natural Lang. Process.*, 2014, pp. 1532–1543. [Online]. Available: https://www.aclweb.org/anthology/D14-1162
[36] M. D. Hoffman, D. M. Blei, C. Wang, and J. W. Paisley, "Stochastic variational inference," *J. Mach. Learn. Res.*, vol. 14, pp. 1303–1347, 2013. [Online]. Available: http://jmlr.org/papers/v14/hoffman13a.html
[37] W. Gao and J. Wu, "A novel framework for service set recommendation in mashup creation," in *Proc. IEEE Int. Conf. Web Serv.*, 2017, pp. 65–72, doi: 10.1109/ICWS.2017.17.
[38] K. He, X. Zhang, S. Ren, and J. Sun, "Delving deep into rectifiers: surpassing human-level performance on imagenet classification," in *Proc. IEEE Conf. Comput. Vision Pattern Recognit.*, 2015, pp. 1026–1034, doi: 10.1109/ICCV.2015.123.



Yutao Ma (M'10) received the Ph.D. degree in computer science from Wuhan University, Wuhan, China, in 2007.

He is currently an Associate Professor with the School of Computer Science, Wuhan University. He was with the Institute of China Electronic System Engineering Corporation, Beijing, China, as a Postdoctoral Fellow and has been a Visiting Scholar with the Department of Electronic and Computer Engineeringy, Lehigh University, Bethlehem, PA, USA. His research interests include the development and maintenance of large-scale software service systems. He has authored and coauthored more than 50 peer-reviewed papers and received two best paper awards at international conferences.

Dr. Ma is currently a Senior Member of the China Computer Federation (CCF) and a member of the CCF Technical Committee on Services Computing.

Xiao Geng received the B.S. degree in computer science from the Central University of Finance and Economics, Beijing, China, in 2018. He is currently working toward the master's degree with the School of Computer Science, Wuhan University, Wuhan, China.

His current research interests include services computing, recommender systems, and deep learning.

Jian Wang (M'11) received the Ph.D. degree in computer science from Wuhan University, Wuhan, China, in 2008.

He is currently a Lecturer with the School of Computer Science, Wuhan University. His current research interests include services computing and software engineering. He has authored and coauthored more than 40 peer-reviewed papers.

He is currently a Member of the IEEE, a member of the China Computer Federation (CCF), and a member of the CCF Technical Committee on Services Computing.