IEEE Access
Multidisciplinary | Rapid Review | Open Access Journal

Received 8 March 2024, accepted 28 March 2024, date of publication 4 April 2024, date of current version 11 April 2024.

Digital Object Identifier 10.1109/ACCESS.2024.3384487

METHODS

Cooperative Mashup Embedding Leveraging Knowledge Graph for Web API Recommendation

CHUNXIANG ZHANG[1], SHAOWEI QIN[1], HAO WU[1], AND LEI ZHANG[2], (Member, IEEE)
[1]School of Information Science and Engineering, Yunnan University, Kunming 650091, China
[2]School of Electrical and Automation Engineering, Nanjing Normal University, Nanjing 210024, China

Corresponding authors: Shaowei Qin (qinshaowei.ynu@gmail.com) and Hao Wu (haowu@ynu.edu.cn)

This work was supported in part by the National Natural Science Foundation of China under Grant 61962061, in part by Yunnan Provincial Foundation for Leaders of Disciplines in Science and Technology under Grant 202005AC160005, in part by Yunnan High-Level Talent Training Support Plan: Young Top Talent Special Project under Grant YNWR-QNBJ-2019-188, in part by the Postgraduate Research and Innovation Foundation of Yunnan University under Grant ZC-22221418, and in part by the Applied Basic Research Project of Yunnan Province under Grant 2013FB009.

ABSTRACT Creating top-notch Mashup applications is becoming increasingly difficult with an overwhelming number of Web APIs. Researchers have developed various API recommendation techniques to help developers quickly locate the right API. In particular, deep learning-based solutions have attracted much attention due to their excellent representation learning capabilities. However, existing methods mainly use textual or graphical information, and do not fully consider the two, which may lead to suboptimal representation and damage recommendation performance. In this paper, we propose a Cooperative Mashup Embedding (CME) neural framework that integrates knowledge graph embedding and text encoding, using Node2Vec to convert entities into numerical vectors and BERT to encode text descriptions. A cooperative embedding method was developed to optimize the entire model while capturing graph and text data knowledge. In addition, the representations obtained by the framework of the three recommendation models are derived. Experimental results on the ProgrammableWeb dataset indicate that our proposed method outperforms the SOTA methods in recommendation performance metrics Top@{1,5,10}. Precision and Recall have increased from 3% to 11%, while NDCG and MAP have improved from 3% to 6%.

INDEX TERMS Mashup applications, API recommendation, knowledge graph, cooperative embedding.

I. INTRODUCTION

A Web mashup service is created by combining multiple Web services, programming interfaces (APIs), or data sources form a new synthetic service or application that offers additional value and capabilities [1]. Bookfinder is a mobile app for browsing and searching different book titles (Fig. 1). The application integrates several APIs from different service providers, such as Google Books API, Open Library Books API, Amazon Product Advertising API, and eBay Finding API. Mashup development methods liberate the workload of developers, release their innovative ideas, and realize various novel applications without requiring the high skills of developers. As the number of Web APIs continues to grow, a large number of heterogeneous APIs/services with complex relationships compete and collaborate. An ecosystem centered on API data resources has gradually formed [2]. Online service registries like ProgrammableWeb, RapidAPI, and API Harmony are becoming increasingly popular, and thus, the number of Web APIs that can be accessed has continued to proliferate [3]. For example, there are 24,000+ registered Web APIs and 8, 000+ registered mashup applications covering 500 application categories in ProgrammableWeb. With so many resources, developers are burdened with information overload. Consequentially, it is important to enhance the API recommendation model to optimize the application of mashups and reduce the burden of people's needs [4].

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

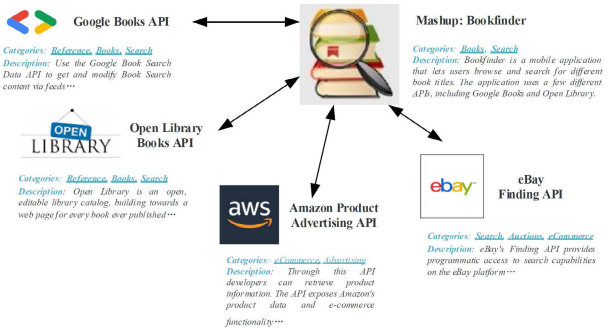


FIGURE 1. An example of mashup.

Researchers from academia and industry have developed a series of recommendation methods and achieved significant results. Research on recommendation systems can be categorized into three main types: content-based, collaborative filtering, and hybrid recommendations. The core of these methods is to evaluate the correlation between mashup development requirements and candidate APIs [5]. Whether the recommended model needs training, it can be roughly divided into learning and memory-based methods [4]. The learning methods mainly use topic models [6], [7], [8], factorization machines [9], [10], deep learning [11], [12], or a combination of several methods. During the model learning, prediction accuracy is improved through model training and continuous parameter adjustment. Deep learning methods offer robust modeling capabilities for various features, fostering extensive applications and gradually becoming the preferred solution. However, most of these works use a single information source, such as deep learning which focuses on using textual features, and knowledge graph embedding which focuses on using graph structure features. These methods have successfully solved some challenges but still face some problems. The method based on graph knowledge can effectively address the problem of data sparsity and achieve high recommendation accuracy [13], [14]. However, how to model user requirements is still a difficult issue, and specifying keyword combinations is not user-friendly for developers because they must have solid domain knowledge and development experience. Content-based requirement understanding can alleviate the requirements for developers. However, it will seriously affect the performance of recommendation algorithms when text features are insufficient. Obviously, without fully considering the knowledge of text data and graph data, it often results in sub-optimal solutions for Mashup-oriented Web API recommendation.

To address this problem, we propose our Cooperative Mashup Embedding (CME) neural framework. This framework integrates knowledge graph embedding and textual encoding in a unified manner. In particular, we first encode the knowledge entailed in the mashup history data with the graph structure and employ graph embedding algorithms [15] to transform the entities of Mashups and APIs into numerical vectors. At the same time, we build a classification model where BERT [16] is employed to encode the textual description of Mashups into vector representation, and the Web API recommendation is modeled and solved as a multi-label classification task. Then, we develop a novel cooperative mashup embedding method to enable knowledge exchange between the two different embedding spaces, by which both knowledge from graph data and text data can be captured simultaneously. Finally, we derive three recommendation models to leverage the obtained representations from CME to realize Web API recommendations.

The main work of this article can be summarized as follows:

- We propose a neural framework that combines knowledge graph embedding and textual encoding in a unified manner, by which both knowledge from graphs and text data can be captured simultaneously.
- We derive three recommendation models for mashup-oriented Web API recommendation to leverage the intermediate presentations of different entities.
- We have conducted intensive experiments and analysis to show the merits of our proposed method compared with the state-of-the-art techniques.

The rest of the article is structured as follows: Section II introduces the preliminary knowledge. Section III shows the overall framework of our model. Section IV details the experimental results and analysis. Section V describes the related work. The paper is summarized in Section VI.

II. PRELIMINARIES

A. PROBLEM DEFINITION

- A **mashup** $m =< F_m, T_m, A_m, X >\in M$ comprises a textual description F_m, a set of categories T_m that indicate its functions, a bundle of APIs A_m, and a set of meta-elements X to prepare for composition [4].
- A Web API $a =< F_a, T_a, X >\in A$ consists of a textual description F_a, a set of categories T_a to index its classification or functions, a set of meta-elements X, such as quality attributes, input ports, and output ports [4].
- **Web API recommendation** is defined as a process where, given a textual description of mashup (it can be seen as the development requirement in our scenario), the recommendation algorithm generates a list of Web APIs according to their relevance to the mashup.

The symbols and notations are given in Table 1.

B. KNOWLEDGE GRAPH EMBEDDING WITH Node2Vec

Knowledge graph embedding is a crucial step in utilizing knowledge graphs for machine learning tasks [17]. Node2Vec [18] is a popular method to learn vector representations for entities in a knowledge graph by utilizing Word2Vec and random walks. In this section, we'll delve into the process of creating entity embeddings using Node2Vec.

TABLE 1. Symbols and notations.

Symbol	Notation
$G = (V, E)$	The knowledge graph G
$m \in M$	The set of Mashups
$a \in A$	The set of Web APIs
i, j, k	The indices for entities
$\mathcal{N}(m_i)$	The top-similar mashups of mashup m_i
$R(m_i, a)$	The relevance between mashup m_i and API a
$\mathrm{Sim}(m_i, m_j)$	The similarity of two mashups
F_{m_i}	The textual functional description of mashup m_i
$\mathrm{FC}_*(), \mathrm{BERT}()$	The forward function of different neural layers
e_a, e_m	The embedding of API a, mashup m from Node2Vec
$e_m^0, \hat{e}_m^1, \hat{e}_m^2$	The vector representation of mashup m with BERT

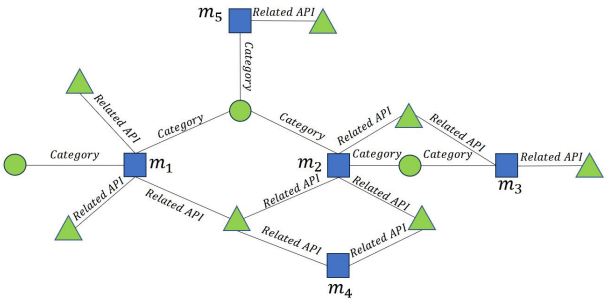


FIGURE 2. An example of knowledge graph. Mashups are represented by rectangles, APIs are represented by triangles, other entities are represented by circles, and edges represent entity relationships.

1) KNOWLEDGE GRAPH CONSTRUCTION

We primarily focus on two types of entities: mashups and APIs. Our graph represents the relationships between entities as undirected edges. For example, the ''Related API'' between an API and a mashup means that an API was invoked by a mashup. The remaining features associated with Web APIs and mashups(such as category, type, company, and so on) are treated as attributes of the respective entities. Fig. 2 depicts a simple knowledge graph.

2) GENERATING NODE SEQUENCES VIA RANDOM WALK

Node2vec is an extension of DeepWalk which combines random walks of both depth-first search (DFS) and breadth-first search (BFS) neighborhoods. It controls whether the model leans more towards BFS or DFS by adjusting the directional parameters. BFS can better reflect the ''structure'' of graph networks, as the sequences generated by BFS often consist of network structures around the current node. DFS can better reflect the ''homogeneity'' of graph networks, as DFS is more likely to travel to nodes far away from the current node.

3) NODE EMBEDDING VIA SKIP-GRAM

Following the same approach as Word2Vec, Node2Vec learns an embedding for each node by Skip-Gram. Once we have obtained the vector representations for entities, we can leverage them for various downstream tasks like classification, clustering, or recommendation systems.

C. TEXT EMBEDDING WITH BERT

BERT, short for Bidirectional Encoder Representation from Transformers, which is a pre-trained language representation model that diverges from traditional unidirectional language models and their shallow concatenations. Instead, BERT employs a novel masked language model to generate profound bidirectional language representations [16]. As a result, BERT has become a fundamental component in forming various downstream tasks. In this study, we utilize BERT as the default text encoder to construct our framework.

1) TEXT PREPROCESSING

BERT encodes each word in a sentence into three vectors: Token Embedding, Segment Embedding, and Position Embedding. We break down the text description of each mashup into a series of subwords and encode them using BERT's vocabulary. Additionally, we append the [CLS] tag to the beginning of the input sequence as the input start for the classification task and use the [SEP] tag to separate the text description and other information.

2) TEXT ENCODING

By feeding the preprocessed text sequence into the BERT model, we acquire an embedding vector for each subword, including the cls vector or the vector of the last token. We utilize the ''cls'' vector since it captures the semantic information and contextual representation of the entire sentence, allowing us to categorize it effectively.

III. METHODOLOGY

A. COOPERATIVE MASHUP EMBEDDING LEVERAGING KNOWLEDGE GRAPH

Our objective is to devise an innovative Collaborative Mashup Embedding (CME) framework that seamlessly integrates the strengths of Node2Vec and BERT. This integration aims to embed mashups, harnessing both graph-based and text-based knowledge, thereby enhancing the performance of mixture-oriented Web API recommendation tasks. As depicted in Fig. 3, the CME framework is structured to facilitate effective feature exchange between two crucial components.

$$\begin{aligned} e_m^0 &= \mathrm{FC}_0(\mathrm{BERT}(F_m)), \\ \hat{e}_m^1 &= \tanh(\mathrm{FC}_1(e_m^0)), \\ \hat{e}_m^2 &= \mathrm{sigmoid}(\mathrm{FC}_2(e_m^0)). \end{aligned} \tag{1}$$

Within the BERT component, we introduce three fully connected (FC) layers, carefully crafted to enhance feature selection and facilitate interaction between the text domain and the knowledge graph domain. The FC_0 layer serves as a linear transformation without an activation function, acting as a pooling mechanism to prevent overfitting and maintain feature stability. After passing through the FC_0 layer of the BERT model, the Mashup text descriptions generate the e_m^0 feature vector. In contrast, the FC_1 and FC_2 layers employ

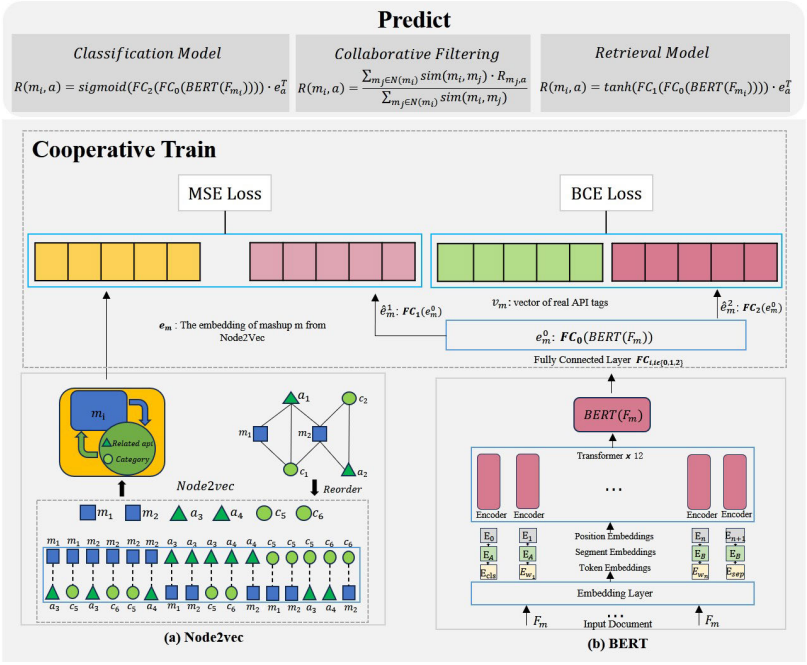


FIGURE 3. The framework of CME. Node2Vec realizes knowledge graph embedding, and BERT fulfills the text embedding. Two modules will exchange and share knowledge from two embedding spaces in cooperative training.

nonlinear transformations through the adoption of tanh and sigmoid activation functions, enabling BERT to learn and execute more complex tasks.

To achieve effective feature exchange, we draw inspiration from the classical encoder-decoder architecture but introduce innovative modifications. Specifically, we incorporate two decoders that decode the output of the BERT encoder into distinct semantic spaces: the knowledge graph embedding space and the API-specific space. In the knowledge graph embedding space, we utilize the embeddings of mashups as training labels, while in the API-specific space, we employ multi-hot vectors of Web APIs as training labels. This design enables our framework to simultaneously capture Mashup-API calling patterns, Mashup/APIs co-occurrence patterns, and textual information of mashups, thereby enriching the representation of mashups and enhancing the accuracy of Web API recommendations.

Algorithm 1 Training CME

Require: M, A, GlobalEpoch, LocalEpoch, Batch Size, ...;
Ensure: Model Parameters;
Initialize parameters of Node2Vec, BERT, Full-Connection layers: FC_0, FC_1, FC_2;
Create knowledge graph G, and generate entity embeddings with Node2Vec;
for $i = 1$ *to* GlobalEpoch **do**
  /* Finetune BERT and FC Layers*/
  for $j = 1$ *to* LocalEpoch **do**
    for each mashup/minibatch $m \in M$ **do**
      Forward propagation on BERT and FC_* by Eq. 1;
      Fetch the Mashup embedding e_m as a pseudo label of regression task;
      Transform A_m to a multi-hot vector v_m as label of classification task;
      Calculate loss: $\mathcal{L} = \mathrm{mse}(e_m, \hat{e}_m^1) + \mathrm{bce}(v_m, \hat{e}_m^2)$;
      Minimize $\mathcal{L}$ with back-propagation and gradient descent;
    end for
  end for
  /* Finetune Node2Vec*/
  for $j = 1$ *to* LocalEpoch **do**
    for each mashup/minibatch $m \in M$ **do**
      Perform the forward propagation: $\hat{e}_m^1 = \tanh(\mathrm{FC}_1(e_m^0))$;
      Re-initialize the embedding in Node2Vec: $e_m \leftarrow \hat{e}_m^1$;
    end for
    Re-run Node2Vec to adjust the graph-based embeddings;
  end for
end for
return The model parameters of CME.

1) DURING THE TRAINING PHASE

we adopt an alternating training strategy to optimize both Node2Vec and the encoder-decoder model. Through collaborative training, we align the deep representations of Mashups encoded by BERT with the shallow representations generated by Node2Vec for Mashups, facilitating the integration of textual and graphical data. This process iterates repeatedly until the model converges, significantly enriching knowledge embeddings and enhancing BERT's encoding capabilities for Mashup texts, ultimately improving recommendation accuracy.

2) DURING THE TESTING PHASE

we leverage the diverse representations generated by the CME model to construct various recommendation algorithms. These algorithms take the textual descriptions of Mashups as input and recommend Web APIs based on their association strengths with the input. For classification models, we directly output the probability distribution of all Web APIs, enabling the sorting of APIs based on their probabilities.

B. MASHUP-ORIENTED WEB API RECOMMENDATION MODELS

Based on our framework, we can generate representations of mashups and Web APIs, these vector representations provide us with an opportunity to develop various recommendation algorithms in different principles.

1) CLASSIFICATION MODEL (CM)

This model treats the Web API recommendation as a multi-label classification task that takes the text-based representation of mashup as the classifier's input, and the Web APIs are used as a label for training the classifier. In the recommendation phase, when a new mashup is fed into the classifier, it outputs the probability of each candidate API. A recommendation list will be generated by sorting all candidates by the probability value.

$$R(m_i, a) \triangleq \Pr(\mathrm{a}|\mathrm{m_i}) = \mathrm{sigmoid}(\mathrm{FC}_2(e_m^0)) \tag{2}$$

2) RETRIEVAL MODEL (RM)

This model considers the Web API recommendation as a retrieval process, assuming that both the mashup requirement and the candidate APIs are expressed in the same semantic space. Then, a recommendation list can be generated by calculating the similarity between the mashup and candidate APIs.

$$R(m_i, a) = \tanh(\mathrm{FC}_1(e_m^0)) \cdot e_a^T \tag{3}$$

3) COLLABORATIVE FILTERING (CF)

Collaborative filtering is a classic way to construct recommender systems. It is a commonly accepted notion that users who share similar interests and preferences are likely to prefer similar items. This also applies to API recommendation scenarios, where mashups with similar requirements are more likely to reuse similar combinations of Web APIs. In extreme cases, we can completely reuse the existing solution if a new requirement is the same as a known mashup. Given a fresh mashup m_i, its text description (to indicate the development requirements) will be encoded by BERT. Then, we find a top-similar set $\mathcal{N}(m_i)$ that is similar to m_j based on their text-based representation. The relevance score between the mashup m_i and an API a is decided by aggregating all relevance scores from $\mathcal{N}(m_i)$ to a,

$$R(m_i, a) = \frac{\sum_{m_j \in \mathcal{N}(m_i)} \mathrm{Sim}(m_i, m_j) \cdot R(m_j, a)}{\sum_{m_j \in \mathcal{N}(m_i)} \mathrm{Sim}(m_i, m_j)}, \tag{4}$$

where $\mathrm{Sim}(m_i, m_j) = \mathrm{cosine}(e_{m_i}^0, e_{m_j}^0)$ is the cosine similarity between two mashups, $e_{m_i}^0, e_{m_j}^0$ are the intermediate vector representations of two mashups. Finally, we can generate a recommendation list by sorting all candidate APIs on their relevance to m_i.

To introduce the recommendation process, we have summarized the details of Web API Recommendation in Algorithm 2.

Algorithm 2 Web API Recommendation

Require: Trained model of CME, M_{test}, A;
Ensure: Top-ranked Web APIs;
for each test mashup $m \in M_{test}$ **do**
  Perform the forward propagation: $e_m^0 = \mathrm{FC}_0(\mathrm{BERT}(F_m))$;
  for each Web API $a \in A$ **do**
    if CM is used: **then**
      Calculate relevance $R(m, a)$ by Eq.2;
    end if
    if RM is used: **then**
      Calculate relevance $R(m, a)$ by Eq.3;
    end if
    if CF is used: **then**
      Calculate relevance $R(m, a)$ by Eq.4;
    end if
  end for
  Sort all candidate APIs according to $R(m, a)$;
end for
return Top-ranked APIs for each test mashup

IV. EXPERIMENTS AND ANALYSIS

A. DATASET

The data used in the experiment comes from ProgrammableWeb.com, which lists the Internet company's open Web API and mashup, covering multiple categories. The dataset in this paper contains 6404 mashups, 1629 Web APIs that have been invoked at least once, and 459 categories. Each mashup calls an average of 2.061 Web APIs. For each mashup, the obtained data mainly contains four kinds of metadata: mashup name, description information, category information, and Web API call information. For each API, the data contains three main types of metadata: API name, description information, and main category. Table 2 shows the statistics of the experimental dataset.

TABLE 2. The statistics of the experimental dataset.

Statistics	Value
#Mashups	6404
#Web APIs	1629
#Categories	459
#Web APIs Invocations	13204
#Web API per Mashup	2.061

We split the Mashup dataset into training and testing sets to evaluate performance using an 8:2 ratio. The API collection serves dual purposes - it is used for both training the model and assessing its recommendation performance. This means that all APIs will be considered as candidates, and after ranking, the top-k APIs will form the recommended list. During training data preparation, we must create a knowledge graph, which includes entities and relationships based on the schema depicted in Fig. 2. Notably, the Mashups in the test set do not contribute to creating a knowledge graph, resulting in the absence of entity nodes. As a result, our model training falls under inductive learning instead of transactional learning. It is important to mention that since the task of Web API recommendations aims at assisting mashup creation, the mashups in the testing set are cold-start and lack any input. To address this, we utilize the text to emulate the potential functional needs of mashups.

B. EVALUATION INDICATORS

An important part of recommender system research is to evaluate the performance of recommendation algorithms, and different evaluation measures can evaluate the model from different perspectives. For Web API recommendations, four widely used indicators of Recall, NDCG, Precision, and MAP can evaluate the accuracy of the model, and the higher the value of these four indicators, the better the performance.

Precision focuses on the correctness of the recommendation results, that is, whether the recommended Web API meets the interests and needs of mashups.

$$\text{Prec@N} = \frac{|\text{APIs}_{\text{real}} \cap \text{APIs}_{\text{Top-N}}|}{|\text{APIs}_{\text{Top-N}}|} \tag{5}$$

Recall: A high recall rate indicates that the recommender system can better cover APIs that users may be interested in but does not pay attention to the ranking of recommendation results.

$$\text{Recall@N} = \frac{|\text{APIs}_{\text{real}} \cap \text{APIs}_{\text{Top-N}}|}{|\text{APIs}_{\text{real}}|} \tag{6}$$

NDCG (Normalized Discounted Cumulative Gain): Considering the ranking order and relevance of the recommendation results, the higher the ranking for the API the user is interested in, the better. NDCG values are between 0 and 1, and closer to 1 indicates better sorting quality of recommended results. A common description of NDCG is:

$$\text{DCG@N} = \sum_{i=1}^{N} \frac{2^{\text{rel}_i} - 1}{\log_2(i+1)}, \ \text{IDCG@N} = \sum_{i=1}^{|\text{GT}|} \frac{1}{\log_2(i+1)}, \tag{7}$$

where $\text{GT} = \text{APIs}_{\text{real}} \cap \text{APIs}_{\text{Top-N}}$, $\text{rel}_i = 1/0$ indicates whether the i-th Web API is truly relevant to the current mashup. NDCG@N is achieved by standardizing DCG@N with the ideal candidate list DCG: $\text{NDCG@N} = \frac{\text{DCG@N}}{\text{IDCG@N}}$. IDCG@N is pre-computed by computing all the real-world APIs that participate in developing the target mashup.

AP (Average Precision): AP considers the ranking order and relevance of recommended results and calculates the average precision. A higher AP value indicates better sorting quality of recommendations. Average AP@N obtains MAP@N on all test samples.

$$\text{AP@N} = \sum_{i=1}^{N} \frac{\text{rel}_i * \text{Prec@i}}{\sum_{i=1}^{N} \text{rel}_i} \tag{8}$$

C. EVALUATION METHODS

For comparison, we consider the representative recommendation methods covering different principles, such as topic modeling, random walks, and deep learning.

- SPR uses topic-modeling techniques to effectively model the relationship between mashup descriptions and APIs, providing highly accurate API recommendations based on thorough topical structure analysis [8].
- RWR encodes mashup-specific contexts in a knowledge graph and estimates relevance to Web APIs via random walks based on keyword requirements [5].
- NCF is a classic neural model in recommender systems [19] which combines the naive matrix factorization and multi-perceptron layer-based matrix factorization.
- FC-LSTM is based on LSTM with a functional attention mechanism and a contextual attention mechanism to help select the appropriate services [20].
- MTFM exploits a convolutional neural network to encode the textual requirement of mashup and interact with the features of APIs [4]. The recommendation for Web API is modeled as a multi-label classification task, co-optimized with category judgment.
- FSFM integrates structural relationships in the MAHN to derive embedding vectors and generates requirement semantic vectors for Mashups and APIs, ultimately combining all representation vectors for feature fusion to produce a list of candidate APIs [21].
- BERT-CM is the simple classification model based on BERT where the Web API recommendation is also modeled as a multi-label classification task [22].
- CME-CM. The recommendation algorithm in the principle of classification model with CME.
- CME-RM. The recommendation algorithm in the principle of retrieval model with CME.
- CME-CF. The recommendation algorithm in the principle of collaborative filtering model with CME.

Regarding the baselines, SPR, RWR, NCF, FC-LSTM, FSFM, and MTFM adopt the source code released by Duan et al.[1] For the remaining models, we implement them using Pytorch framework.

D. METHODS PERFORMANCE ANALYSIS

The recommendation performance of different models with cutoffs of N=1,5,10 is presented in Table 3. Considering the baseline models, it is easy to find that FSFM performs best in the baseline model with a N range of 5 to 10, as it integrates the advantages of semantic functions, MAHN structures, and feature fusion. BERT-CM shows excellent performance at N=1. This is mainly due to the advantages of directional encoder representation from transformers. For CME-CF, its ranking performance on N=1 outperforms all baseline models, indicating that traditional collaborative filtering algorithms still have advantages. For CME-RM, its ranking performance is similar to BERT-CM except for top-1 recommendations. For CME-CM, it outperforms all competitors in the Top-5 recommendation. Especially, a performance gain of 3.07% to 11.1% on Precision@N, 3.30% to 10.7% on Recall@N, 3.07% to 6.06% on NDCG@N, 3.07% to 4.29% on MAP@N was achieved respectively compared with BERT-CM. Due to BERT-CM being seen as a degraded version of CME-CM without injection of domain knowledge provided by Node2Vec. This confirms that injecting domain

[1] https://github.com/whale-ynu/MTFM

TABLE 3. Performance comparison of Web API recommendation. The best values regarding baselines and our proposed methods are marked by ***Italic*** and **Bold**.

Methods	Prec@1	Prec@5	Prec@10	Recall@1	Recall@5	Recall@10
NCF	0.3273	0.1331	0.0848	0.2250	0.3894	0.4702
SPR	0.4141	0.1684	0.1084	0.3009	0.5122	0.6312
FC-LSTM	0.3375	0.1457	0.0926	0.2465	0.3984	0.4476
RWR	0.3859	0.1703	0.1027	0.2852	0.5276	0.6098
MTFM	0.5700	0.1835	0.1077	0.3929	0.5411	0.6011
FSFM	0.6220	0.1924	*0.1104*	0.4266	0.5730	0.6273
BERT-CM	*0.6571*	*0.1949*	0.1058	*0.4638*	*0.5984*	*0.6336*
CME-CF	0.6532	0.1961	0.1020	0.4594	0.6033	0.6164
CME-RM	0.4588	0.1924	0.1153	0.3151	0.5745	0.6655
CME-CM	**0.6773**	**0.2166**	**0.1208**	**0.4791**	**0.6564**	**0.7015**
Gains	3.07%	11.1%	9.42%	3.30%	9.69%	10.7%
Methods	**NDCG@1**	**NDCG@5**	**NDCG@10**	**MAP@1**	**MAP@5**	**MAP@10**
NCF	0.3273	0.4373	0.4597	0.3273	0.3985	0.3986
SPR	0.4141	0.5559	0.5828	0.4141	0.5157	0.5113
FC-LSTM	0.3375	0.4462	0.5781	0.3375	0.4091	0.4172
RWR	0.3859	0.5558	0.5735	0.3859	0.5086	0.5059
MTFM	0.5700	0.6364	0.6425	0.5700	0.6102	0.5976
FSFM	0.6220	0.6825	0.6902	0.6220	0.6264	0.6471
BERT-CM	*0.6571*	*0.7027*	*0.7029*	*0.6571*	*0.6854*	*0.6798*
CME-CF	0.6532	0.7168	0.7179	0.6532	0.6968	0.6936
CME-RM	0.4588	0.6109	0.6289	0.4588	0.5619	0.5577
CME-CM	**0.6773**	**0.7414**	**0.7455**	**0.6773**	**0.7148**	**0.7050**
Gains	3.07%	5.51%	6.06%	3.07%	4.29%	3.71%

knowledge into the classifier model through cooperative mashup embedding significantly strengthens the model's expressive capability and thus benefits recommendation performance.

E. COMPUTATIONAL COSTS ANALYSIS

In this section, we quantitatively analyze the computational cost with the selected models. We utilize a Dell Precision 7920 Tower Workstation equipped with Ubuntu 22.04 LTS, an Intel Xeon Gold 6234 Processor, 128GB of memory, and an Nvidia Quadro RTX8000 GPU. As shown in Table 4, due to the use of BERT as the text encoder, the BERT-CM and CME models are 6 times larger than MTFM and 4 times larger than FSFM, the single-round training cost of BERT-CM is more than that of them. Due to the integration of graph embedding, alternate training, and other mechanisms, a single-round training of CME requires more time, approximately five times that of BERT-CM, under the same configuration. However, the training time of CME can be significantly reduced by increasing batch size. For example, doubling the batch size can save 40% of the training time.

TABLE 4. Comparison of the computational cost.

Model	Time per epoch (s)	Model size(G)
NCF	16.80s	0.015G
SPR	9.95s	0.111G
FC-LSTM	4.05	0.004G
MTFM	25.58s	0.018G
FSFM	18.84s	0.031G
BERT-CM	64.26s	0.122G
CME	340.32s	0.124G

F. CASE-STUDY ANALYSIS

In this study, we aim to evaluate different methods for suggesting APIs for mashups, which are Web applications that combine data or functionality from multiple sources. Table 5 shows the top five suggestions for four target mashups, which cover different types of applications.

Maps Compare compares the mapping service APIs of different providers. BERT-CM and CME-RM suggest the three target APIs well compared to other methods. CME-CM not only perfectly suggests all target APIs but also suggests a much more relevant candidate *Yahoo Geocoding* which is a geocoding Web service that helps developers make their applications location-aware. **Seesu** uses vk.com, soundcloud.com, ex.fm, last.fm, and youtube.com to give users a music player and socialize. Since there are six APIs involved and the text information is insufficient, it is hard to suggest the right candidates. Most of the methods can only match **YouTube** and **Last.fm** while CME-CM significantly outperforms them by identifying four real APIs. **eSignature Gateway** provides document signature functionality for all email clients and includes electronic storage, data analytics, and workflow capabilities. Most methods only select one option. BERT-CM, FSFM, RWR, and MTFM can match both **DocuSign Enterprise** and **Twilio SMS**. CME-CM performed better in this situation, matching three targets. **BlockWild** has a relatively sufficient text description. Most models can suggest three target APIs: **Google Maps**, **Facebook**, and **Twitter**. In contrast, CME-CM can predict

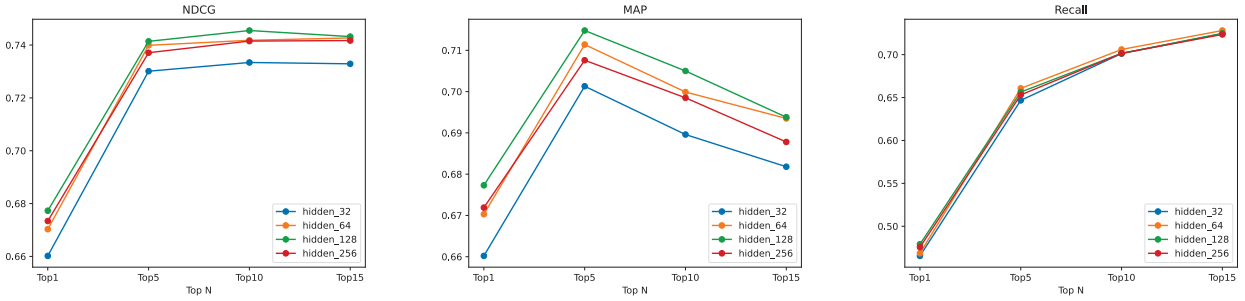


FIGURE 4. Impact of the dimensionality of FC_1 layer w.r.t CME-CM.

four target APIs, including **Upcoming.org**, a service that helps you share and keep track of your events, which all remaining competitors omit.

G. PARAMETER IMPACT

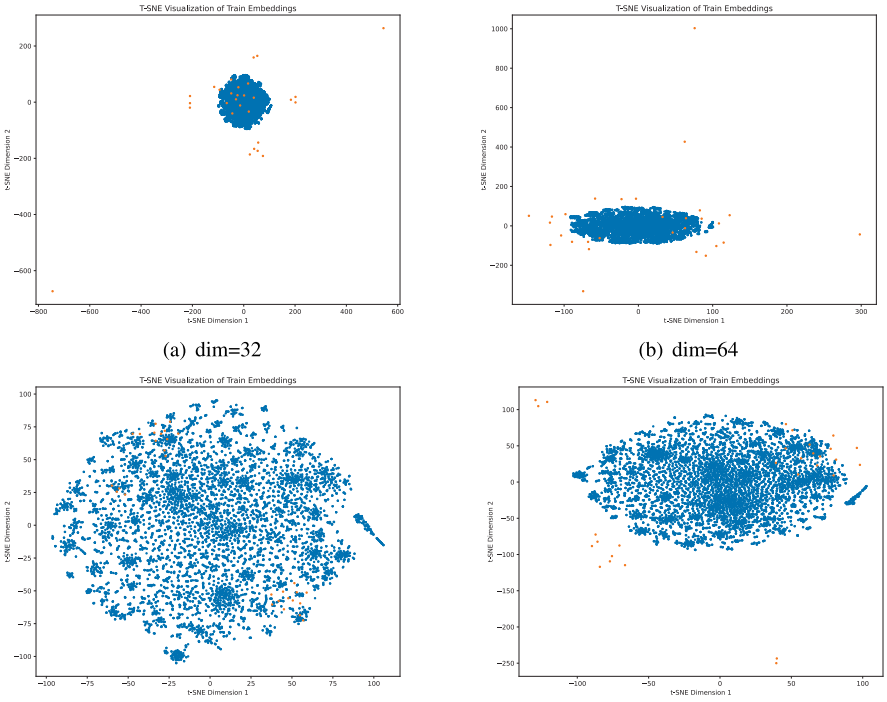
In the CME framework, the mashup embeddings of known nodes from the knowledge graph are used as pseudo labels to co-optimize and inject additional knowledge into the BERT module. An important way to measure knowledge graph embedding quality is to observe its embedding dimension's impact on recommendation performance. The dimensionality of the FC_1 layer is equivalent to the node embedding dimension. Moreover, the FC_2 layer acts as a feature selection bottleneck, and only by compressing global features into an appropriate dimension, the downstream classification task can achieve optimal performance [23]. Therefore, we explored the influence of the dimensionality of FC_1 and FC_2. To simplify hyperparameter settings, we always keep the dimensions of the two layers the same.

To provide both qualitative and quantitative analysis, Fig. 4 shows how FC_1 affects the recommendation accuracy of CME-CM, and Fig. 5 shows how FC_2 affects the visualization of mashup representations. Generally speaking, smaller dimensions lead to information loss, which will limit the expressive ability of the model and make it impossible to distinguish tiny differences between different representations (see Fig. 5a and 5b). Relatively speaking, increasing the embedding dimension helps to enhance the expressive power. From Figs.4a and 4b, it can be seen that the ranking indicator of recommendation results has significantly improved while also improving the performance of the recall rate with the increase of embedding dimension. Larger dimensions introduce sparsity problems, which are also not conducive to similarity calculation and recommendation (see Fig.4c), also cause model overfitting and hinder the model's expressive ability (see Fig. 5d). For our scenario, 128 is a better choice for FC_1 and FC_2 according to the analysis.

V. RELATED WORKS

Web API recommendation for mashups is a crucial task that has been studied extensively by researchers. There are two main approaches to this problem: content-based and collaborative filtering (CF).

Content-based methods involve representing mashup requirements and candidate Web APIs as vectors and calculating their similarity. For instance, Xia et al. [7] proposed clustering services into categories and recommending them based on their relevance to the mashup requirements. Zhong et al. [8] reconstructed service profiles using an author-topic model to identify important features that could be used for recommendation. Another content-based approach is to use deep learning techniques, such as those proposed by Xue et al. [24], who developed a method for generating mashups using real-world data integration and natural language processing. Xiong et al. [11] also employed deep learning to integrate invocation interactions and functionality into their recommendation system. In addition, Shi et al. [20] combined collaborative filtering and text content methods to characterize the complex relationships between mashups and Web APIs. Their approach leveraged both user behavior and semantic information to provide accurate recommendations. Kang et al. [25] leveraged attentional factorization machines to capture the complex feature interactions and the matching importance. Cao et al. [26] employed bilinear graph representation and deep factorization machines to develop recommendation models. Wu et al. [4] treated the Web API recommendation problem as a multi-label classification task [12] and introduced a framework called MTFM, which generated representations of requirements and modeled feature interactions between mashups and Web APIs. Wang et al. [21] integrates deep neural networks to capture complex nonlinear structures and semantic information of requirements, emphasizing the significance of different feature levels to deliver compound and diverse API recommendations. Overall, these studies demonstrated the potential of content-based recommendation strategies for improving the efficiency and accuracy of Web API recommendation systems. Additionally, more recent efforts try to enhance the text presentation, a.k.a., pose more efforts on outstanding the semantics of requirements by using a more powerful representation of learning techniques [22], [27]. By leveraging different techniques and approaches, they aim to provide better support for mashup development and promote the reuse of existing Web APIs.



(a) dim=32 (b) dim=64

(c) dim=128 (d) dim=256

FIGURE 5. The visualization of representations in FC_2 layer by TSNE.

Collaborative filtering (CF) achieved recommendations by mining the correlation between mashups and Web APIs. Yao et al. [9] proposed a matrix factorization (MF) method with implicit correlation regularization to solve the recommendation problem and enhance the diversity of recommendations. Similarly, Fletcher [28] regularized MF with implicit embeddings of user preference for API recommendation. Yin et al. [29] proposed a framework that contains joint MF and cognitive knowledge mining, by which the hidden relationships among users and APIs can be mined to enhance the ensemble model. Wang et al. [5] proposed a graph-based framework for API recommendation of mashup creation. The authors created a comprehensive knowledge graph that captured various aspects of mashup development, including developer preferences, API functionalities, and usage patterns. They employed a random walk approach to traverse the graph and identify candidate APIs for a given requirement. Wang et al. [13] further introduced an unsupervised deep method based on random walks on the knowledge graph. Specifically, the low-dimensional embedding representations of entities were learned from truncated random walks by treating walks as the equivalent of sentences. Ma et al. [30] proposed a deep neural network called MISR to analyze three types of interactions between services and mashups. The neural network learns to extract hidden structures and features from these interactions, allowing it to recommend services that are compatible with a given mashup. Wang et al. [31] used a graph-based method called CRN to represent relationships between mashups and Web APIs. The approach calculates semantic similarities between developer requirements and mashups to identify candidate mashup nodes in the graph and then uses the graph to find relevant Web APIs.

Some studies use a combination of methods for recommendations to better the recommendation performance. For example, Cao et al. [32] first developed a two-level topic model leveraging the relationship among Mashup services and then performed clustering on Mashups. Finally, they designed a CF algorithm that exploits the co-invocation pattern of Web APIs to recommend diverse APIs for each cluster of Mashups. Liu et al. [14] proposed a dynamic graph neural network-based model to tackle the evolution of service and the semantic gap between services and mashups. Besides, some works concentrate on other aspects of Web API recommendation, such as the diversity and compatibility of API candidate sets [33], [34], [35]. For instance, Kou et al. [36] solved an automated Web API recommendation task as a nondeterministic polynomial problem. Chen et al. [35] proposed a keyword-based deep-reinforced Steiner tree search to recommend compatible services for mashup creation. Qi et al. [37] described a new approach to recommending Web APIs for mashup creation, which utilizes historical mashup creations to ensure compatibility among the recommended APIs and includes a textual description mining step to precisely capture the developers' functional requirements. Qi et al. [38] further proposed a correlation graph-based approach for personalized and compatible interface recommendations in mobile APP development.

TABLE 5. Case-study analyses where the API is the right answer.

Mashup: Maps Compare
Text: Maps Compare compares the mapping service APIs provided by Google Maps, Bing Maps, Yahoo Maps, and Google.
Related APIs: Google Maps, Yahoo Maps, Google Earth, Bing Maps
Categories: Comparisons, Mapping

Method	Top 5 Recommendations of APIs
SPR	Google Maps, YouTube, Flickr, Amazon Product Advertising, Twitter,
RWR	Google Maps, Amazon Product Advertising, Twitter,eBay, YouTube
MTFM	Google Maps, Yahoo Maps, GeoNames, Flickr, Google Earth
FSFM	Google Maps, Yahoo Maps, YouTube, Flickr, Twitter
BERT-CM	Yahoo Maps, Google Maps, Bing Maps, Google Maps Elevation, Bing
CME-CF	Google Maps, Parse, Trimet, Spy Dialer, FanSnap
CME-RM	Yahoo Maps, YouTube, Twitter, Flickr, Facebook
CME-CM	Google Maps, Bing Maps, Google Earth, Yahoo Maps, Yahoo Geocoding

Mashup: Seesu
Text: Uses vk.com (vkontakte.ru), soundcloud.com, ex.fm, last.fm and youtube.com to give you music player and socialize.
Related APIs: YouTube, Last.fm, Discogs, SoundCloud, VK, exfm
Categories: Aggregation, Music, Video, Search

Method	Top 5 Recommendations of APIs
SPR	YouTube, Last.fm, Google Maps, Twitter, Flickr
RWR	Google Maps, YouTube, Twitter, Flickr, Last.fm
MTFM	Last.fm, MusicBrainz, Spotify Echo Nest, YouTube, LyrDB
FSFM	Last.fm, SoundCloud, Twitter, YouTube, Rdio
BERT-CM	SoundCloud, Last.fm, YouTube, Spotify Echo Nest, Rdio
CME-CF	YouTube, Last.fm, SoundCloud, Parse, TriMet
CME-RM	YouTube, Last.fm, Google Maps, Twitter, Flickr
CME-CM	YouTube, Last.fm, SoundCloud, Spotify Echo Nest, Discogs

Mashup: eSignature Gateway
Text: Signature Gateway leverages Amazon S3, Clockwork SMS, DocuSign, Google Storage, and Twilio APIs to give users document sign functionality regardless of the email client used. In addition, eSignature implements electronic storage, data analytics, and workflow capabilities.
Related APIs: Amazon S3, DocuSign Enterprise, Twilio SMS, Google Storage, Clockwork SMS
Categories: Electronic Signature, Messaging

Method	Top 5 Recommendations of APIs
SPR	Google Maps, Twitter, DocuSign Enterprise, Amazon Product Advertising, YouTube
RWR	DocuSign Enterprise, Google Maps, Twilio, Twitter, Twilio SMS
MTFM	DocuSign Enterprise, Twilio SMS, Google App Engine, Salesforce, Amazon Product Advertising
FSFM	DocuSign Enterprise, Twilio SMS, Google Maps, Google App Engine, Twitter
BERT-CM	DocuSign Enterprise, Twilio, Box, Twilio SMS, Ispeech Automate Speech Recognition
CME-CF	DocuSign Enterprise, Parse, Trimet, Spy Dialer, FanSnap
CME-RM	DocuSign Enterprise, Twilio, Google Maps, Box, Adobe Sign
CME-CM	DocuSign Enterprise, Twilio, Box, Amazon S3, Twilio SMS

Mashup: BlockWild
Text: BlockWild let's you post information directly to a local map where it can be discovered in your community by location and categories of content, such as: classifieds, events, rentals, social groups, services, etc. Search locally or wherever you are planning to explore.
Related APIs: Upcoming.org, Google Maps, Facebook, Twitter, Yelp Fusion
Categories: Localization, Search, Mapping, Events, Classifieds, Directories

Method	Top 5 Recommendations of APIs
SPR	Google Maps, Twitter, YouTube, Flickr, Facebook
RWR	Google Maps, Twitter, TouTube, Flickr, Facebook
MTFM	Google Maps, Yahoo Local Search, Twitter, Google Geocoding, Facebook
FSFM	Google Maps, Twitter, Google Geocoding, TouTube, Facebook
BERT-CM	Google Maps, Google Base, Oodle, Google Adsense, Facebook
CME-CF	Google Maps, Parse, Trimet, Spy Dialer, Fansnap
CME-RM	Google Maps, Twitter, Flickr, YouTube, Facebook
CME-CM	Google Maps, Upcoming.org, Twitter, Yahoo Geocoding, Facebook

Gong et al. [39] focused on the privacy-preservation issue regarding Web API recommendations under edge computing scenarios.

Our solution can be viewed as a hybrid approach that combines the strengths of knowledge graph embedding and text encoding to provide improved Web API recommendations. Unlike existing methods, our approach focuses on creating a novel cooperative mashup embedding (CME) method that enables knowledge exchange between two distinct embedding spaces. By doing so, we can utilize on-shelf neural components to establish a learning framework without modifying the underlying functionality of knowledge graph embedding and text encoding. Moreover, the CME method allows us to derive three unique recommendation models that capitalize on the obtained representations to provide Web API recommendations. These models offer fresh perspectives on how to develop recommender systems for Web APIs, setting our approach apart from conventional methods.

VI. CONCLUSION

We have presented a novel framework for mashup-oriented Web API recommendations, which combine a cooperative mashup embedding strategy and three recommendation algorithms. Our approach leverages pre-trained neural components, such as BERT, to reduce computational complexity and improve accuracy. The CME method enables knowledge sharing across domains, leading to more accurate recommendations. The derived recommendation models provide personalized recommendations that cater to the unique needs of mashup developers.

For future work, we plan to explore the use of more advanced components to further enhance the effectiveness of representation learning. For instance, we aim to replace Node2Vec with graph neural networks for knowledge graph embedding. Additionally, we intend to optimize the model using multi-task learning or contrastive learning, which can potentially improve the overall performance of the recommendation system.

REFERENCES

[1] M. Imran, S. Soi, F. Kling, F. Daniel, F. Casati, and M. Marchese, "On the systematic development of domain-specific mashup tools for end users," in *Web Engineering*. Berlin, Germany: Springer, 2012, pp. 291–298.

[2] W. Tan, Y. Fan, A. Ghoneim, M. A. Hossain, and S. Dustdar, "From the service-oriented architecture to the Web API economy," *IEEE Internet Comput.*, vol. 20, no. 4, pp. 64–68, Jul. 2016.

[3] M. Tang, H. Zhou, and X. Guo, "The API-mashup ecosystem: A comprehensive study of ProgrammableWeb," *Int. J. Embedded Syst.*, vol. 15, no. 2, pp. 132–138, 2022.

[4] H. Wu, Y. Duan, K. Yue, and L. Zhang, "Mashup-oriented Web API recommendation via multi-model fusion and multi-task learning," *IEEE Trans. Services Comput.*, vol. 15, no. 6, pp. 3330–3343, Nov. 2022.

[5] X. Wang, H. Wu, and C.-H. Hsu, "Mashup-oriented API recommendation via random walk on knowledge graph," *IEEE Access*, vol. 7, pp. 7651–7662, 2019.

[6] B. Cao, M. Tang, and X. Huang, "CSCF: A mashup service recommendation approach based on content similarity and collaborative filtering," *Int. J. Grid Distrib. Comput.*, vol. 7, no. 2, pp. 163–172, Apr. 2014.

[7] 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.

[8] Y. Zhong, Y. Fan, W. Tan, and J. Zhang, "Web service recommendation with reconstructed profile from mashup descriptions," *IEEE Trans. Autom. Sci. Eng.*, vol. 15, no. 2, pp. 468–478, Apr. 2018.

[9] L. Yao, X. Wang, Q. Z. Sheng, B. Benatallah, and C. Huang, "Mashup recommendation by regularizing matrix factorization with API co-invocations," *IEEE Trans. Services Comput.*, vol. 14, no. 2, pp. 502–515, Mar. 2021.

[10] B. Cao, M. Peng, Y. Qing, J. Liu, G. Kang, B. Li, and K. K. Fletcher, "Web API recommendation via combining graph attention representation and deep factorization machines quality prediction," *Concurrency Comput., Pract. Exper.*, vol. 34, no. 21, p. e7069, Sep. 2022.

[11] 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, Nov. 2018.

[12] H. Wu, S. Qin, R. Nie, J. Cao, and S. Gorbachev, "Effective collaborative representation learning for multilabel text categorization," *IEEE Trans. Neural Netw. Learn. Syst.*, vol. 33, no. 10, pp. 5200–5214, Oct. 2022.

[13] X. Wang, X. Liu, J. Liu, X. Chen, and H. Wu, "A novel knowledge graph embedding based API recommendation method for mashup development," *World Wide Web*, vol. 24, no. 3, pp. 869–894, May 2021.

[14] M. Liu, Z. Tu, H. Xu, X. Xu, and Z. Wang, "DySR: A dynamic graph neural network based service bundle recommendation model for mashup creation," *IEEE Trans. Services Comput.*, vol. 16, no. 4, pp. 2592–2605, Jul. 2023.

[15] P. Ristoski, J. Rosati, T. Di Noia, R. De Leone, and H. Paulheim, "RDF2Vec: RDF graph embeddings and their applications," *Semantic Web*, vol. 10, no. 4, pp. 721–752, May 2019.

[16] J. Devlin, M.-W. Chang, K. Lee, and K. Toutanova, "BERT: Pre-training of deep bidirectional transformers for language understanding," in *Proc. Conf. North Amer. Chapter Assoc. Comput. Linguistics, Hum. Lang. Technol.* Minneapolis, MI, USA: Association for Computational Linguistics, vol. 1, Jun. 2019, pp. 4171–4186. [Online]. Available: https://aclanthology.org/N19-1423

[17] Q. Wang, Z. Mao, B. Wang, and L. Guo, "Knowledge graph embedding: A survey of approaches and applications," *IEEE Trans. Knowl. Data Eng.*, vol. 29, no. 12, pp. 2724–2743, Dec. 2017.

[18] A. Grover and J. Leskovec, "node2vec: Scalable feature learning for networks," in *Proc. 22nd ACM SIGKDD Int. Conf. Knowl. Discovery Data Mining*, San Francisco, CA, USA, Aug. 2016, pp. 855–864.

[19] X. He, L. Liao, H. Zhang, L. Nie, X. Hu, and T.-S. Chua, "Neural collaborative filtering," in *Proc. 26th Int. Conf. World Wide Web*, Perth, WA, Australia. New York, NY, USA: ACM, Apr. 2017, pp. 173–182.

[20] M. Shi, Y. Tang, and J. Liu, "Functional and contextual attention-based LSTM for service recommendation in mashup creation," *IEEE Trans. Parallel Distrib. Syst.*, vol. 30, no. 5, pp. 1077–1090, May 2019.

[21] X. Wang, M. Xi, and J. Yin, "Functional and structural fusion based Web API recommendations in heterogeneous networks," in *Proc. IEEE Int. Conf. Web Services (ICWS)*, Jul. 2023, pp. 91–96.

[22] X. Wang, P. Zhou, Y. Wang, X. Liu, J. Liu, and H. Wu, "ServiceBERT: A pre-trained model for web service tagging and recommendation," in *Proc. 19th Int. Conf. Service-Oriented Comput. (ICSOC)*, in Lecture Notes in Computer Science, vol. 13121. Berlin, Germany: Springer, 2021, pp. 464–478.

[23] J. Liu, W.-C. Chang, Y. Wu, and Y. Yang, "Deep learning for extreme multi-label text classification," in *Proc. 40th Int. ACM SIGIR Conf. Res. Develop. Inf. Retr.*, Tokyo, Japan, Aug. 2017, pp. 115–124.

[24] Q. Xue, L. Liu, W. Chen, and M. C. Chuah, "Automatic generation and recommendation for API mashups," in *Proc. 16th IEEE Int. Conf. Mach. Learn. Appl. (ICMLA)*, Cancun, Mexico, Dec. 2017, pp. 119–124.

[25] G. Kang, J. Liu, Y. Xiao, B. Cao, Y. Xu, and M. Cao, "Neural and attentional factorization machine-based Web API recommendation for mashup development," *IEEE Trans. Netw. Service Manage.*, vol. 18, no. 4, pp. 4183–4196, Dec. 2021.

[26] B. Cao, L. Zhang, M. Peng, Y. Qing, G. Kang, and J. Liu, "Web service recommendation via combining bilinear graph representation and xDeepFM quality prediction," *IEEE Trans. Netw. Service Manage.*, vol. 20, no. 2, pp. 1078–1092, Jan. 2023.

[27] C. Sang, X. Deng, and S. Liao, "Mashup-oriented Web API recommendation via full-text semantic mining of developer requirements," *IEEE Trans. Services Comput.*, vol. 16, no. 4, pp. 2755–2768, Feb. 2023.

[28] K. Fletcher, "Regularizing matrix factorization with implicit user preference embeddings for Web API recommendation," in *Proc. IEEE Int. Conf. Services Comput. (SCC)*, Jul. 2019, pp. 1–8.

[29] Y. Yin, Q. Huang, H. Gao, and Y. Xu, "Personalized Apis recommendation with cognitive knowledge mining for industrial systems," *IEEE Trans. Ind. Informat.*, vol. 17, no. 9, pp. 6153–6161, Sep. 2021.

[30] Y. Ma, X. Geng, and J. Wang, "A deep neural network with multiplex interactions for cold-start service recommendation," *IEEE Trans. Eng. Manag.*, vol. 68, no. 1, pp. 105–119, Feb. 2021.

[31] Y. Wang, A. Zhou, Q. Huang, X. Wang, and B. Jiang, "PAREI: A progressive approach for Web API recommendation by combining explicit and implicit information," *Inf. Softw. Technol.*, vol. 162, Oct. 2023, Art. no. 107269.

[32] B. Cao, X. F. Liu, M. M. Rahman, B. Li, J. Liu, and M. Tang, "Integrated content and network-based service clustering and Web APIs recommendation for mashup development," *IEEE Trans. Services Comput.*, vol. 13, no. 1, pp. 99–113, Jan. 2020.

[33] F. Wang, L. Wang, G. Li, Y. Wang, C. Lv, and L. Qi, "Edge-cloud-enabled matrix factorization for diversified APIs recommendation in mashup creation," *World Wide Web*, vol. 25, no. 5, pp. 1809–1829, Sep. 2022.

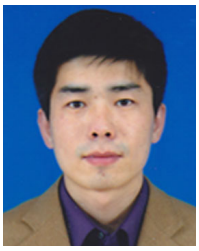
[34] W. Gong, X. Zhang, Y. Chen, Q. He, A. Beheshti, X. Xu, C. Yan, and L. Qi, "DAWAR: Diversity-aware Web APIs recommendation for mashup creation based on correlation graph," in *Proc. 45th Int. ACM SIGIR Conf. Res. Develop. Inf. Retr.*, Madrid, Spain, Jul. 2022, pp. 395–404.

[35] H. Chen, H. Wu, J. Li, X. Wang, and L. Zhang, "Keyword-driven service recommendation via deep reinforced Steiner tree search," *IEEE Trans. Ind. Informat.*, vol. 19, no. 3, pp. 2930–2941, Mar. 2023.

[36] H. Kou, J. Xu, and L. Qi, "Diversity-driven automated Web API recommendation based on implicit requirements," *Appl. Soft Comput.*, vol. 136, Mar. 2023, Art. no. 110137.
[37] L. Qi, H. Song, X. Zhang, G. Srivastava, X. Xu, and S. Yu, "Compatibility-aware Web API recommendation for mashup creation via textual description mining," *ACM Trans. Multimedia Comput., Commun., Appl.*, vol. 17, no. 1s, pp. 1–19, Jan. 2021.
[38] L. Qi, W. Lin, X. Zhang, W. Dou, X. Xu, and J. Chen, "A correlation graph based approach for personalized and compatible Web APIs recommendation in mobile APP development," *IEEE Trans. Knowl. Data Eng.*, vol. 35, no. 6, pp. 5444–5457, Jun. 2023.
[39] W. Gong, W. Zhang, M. Bilal, Y. Chen, X. Xu, and W. Wang, "Efficient Web Apis recommendation with privacy-preservation for mobile app development in Industry 4.0," *IEEE Trans. Ind. Informat.*, vol. 18, no. 9, pp. 6379–6387, Sep. 2022.

CHUNXIANG ZHANG received the B.S. degree in computer science from Henan University, Kaifeng, China, in 2019. He is currently pursuing the master's degree with Yunnan University. His current research interests include service computing, recommender systems, and deep learning.

SHAOWEI QIN received the B.S. degree in computer science from Tianjin Agricultural University, Tianjin, China, in 2018. He is currently pursuing the Ph.D. degree with Yunnan University. He has published in IEEE TRANSACTIONS ON NEURAL NETWORKS AND LEARNING SYSTEMS, *Knowledge-Based Systems*, *Applied Intelligence*, and *Neural Computing and Applications*. His current research interests include deep learning and text mining.



HAO WU received the Ph.D. degree in computer science from Huazhong University of Science and Technology, in 2007. He is currently a Professor with the School of Information Science and Engineering, Yunnan University, China. He has published more than 80 papers in peer-reviewed journals and conferences, e.g., IEEE TRANSACTIONS ON PARALLEL AND DISTRIBUTED SYSTEMS, IEEE TRANSACTIONS ON MOBILE COMPUTING, IEEE TRANSACTIONS ON NEURAL NETWORKS AND LEARNING SYSTEMS, IEEE TRANSACTIONS ON SERVICES COMPUTING, IEEE TRANSACTIONS ON INSTRUMENTATION AND MEASUREMENT, IEEE TRANSACTIONS ON INDUSTRIAL INFORMATICS, IEEE TRANSACTIONS ON NETWORK AND SERVICE MANAGEMENT, *NeuNet*, *WWWJ*, *FGCS*, *KBS*, *PUC*, *ASE*, ACL, NAACL, ICWS, and ICSOC. He has coauthored two monographs published in World Scientific. His research interests include service computing, edge computing, and web intelligence.

LEI ZHANG (Member, IEEE) received the Ph.D. degree from Southeast University, in 2011. He was a Research Fellow with IPAM, UCLA, in 2008. He is currently an Associate Professor with the School of Electrical and Automation Engineering, Nanjing Normal University. He has published more than 50 papers in peer-reviewed journals and conferences, e.g., IEEE TRANSACTIONS ON MOBILE COMPUTING, IEEE TRANSACTIONS ON SERVICES COMPUTING, IEEE TRANSACTIONS ON INSTRUMENTATION AND MEASUREMENT, IEEE TRANSACTIONS ON INDUSTRIAL INFORMATICS, IEEE TRANSACTIONS ON INDUSTRIAL ELECTRONICS, IEEE TRANSACTIONS ON HUMAN-MACHINE SYSTEMS, IEEE TRANSACTIONS ON EMERGING TOPICS IN COMPUTATIONAL, IEEE TRANSACTIONS ON NETWORK AND SERVICE MANAGEMENT, and *ACM TECS*. His research interests include machine learning, service computing, the IoT, and computer vision.