IEEE Access
Multidisciplinary | Rapid Review | Open Access Journal

Received 2 June 2024, accepted 23 June 2024, date of publication 28 June 2024, date of current version 8 July 2024.

Digital Object Identifier 10.1109/ACCESS.2024.3420395

RESEARCH ARTICLE

RLISR: A Deep Reinforcement Learning Based Interactive Service Recommendation Model

MINGWEI ZHANG, YINGJIE QU, YAGE LI, XINGYU WEN, AND YI ZHOU
Software College, Northeastern University, Shenyang 110169, China

Corresponding authors: Mingwei Zhang (zhangmw@swc.neu.edu.cn) and Yingjie Qu (quyj814@163.com)

This work was supported in part by the National Natural Science Foundation of China under Grant 61572117.

ABSTRACT An increasing number of services are being offered online, which leads to great difficulties in selecting appropriate services during mashup development. There have been many service recommendation studies and achieved remarkable results to alleviate the issue of service selection challenge. However, they are limited to suggesting services only for a single round or the next round, and ignore the interactive nature in real-world service recommendation scenarios. As a result, existing methods can't capture developers' shifting requirements and obtain the long-term optimal recommendation performance over the whole recommendation process. In this paper, we propose a deep reinforcement learning based interactive service recommendation model (RLISR) to tackle this problem. Specifically, we formulate interaction service recommendation as a multi-round decision-making process, and design a reinforcement learning framework to enable the interactions between mashup developers and service recommender systems. First, we propose a knowledge-graph-based state representation modeling method, wherein we consider both the positive and negative feedbacks of developers. Then, we design an informative reward function from the perspective of boosting recommendation accuracy and reducing the number of recommendation rounds. Finally, we adopt a cascading Q-networks model to cope with the enormous combinational candidate space and learn an optimal recommendation policy. Extensive experiments conducted on a real-world dataset validate the effectiveness of the proposed approach compared to the state-of-the-art service recommendation approaches.

INDEX TERMS Service recommendation, interactive recommender systems, reinforcement learning, knowledge graph, mashup creation.

I. INTRODUCTION

As a result of the fast development of service-oriented computing (SOC) technologies, the quantity and diversity of Web APIs (also called services in this paper) have been growing rapidly. By taking advantages of existing Web APIs, Mashups offer a way to create completely new and innovative services in an agile manner and to substantially accelerate the development circle of Web applications [1]. However, the enormous number of Web APIs makes it increasingly difficult for mashup developers to quickly and accurately find desired ones. To relieve this difficulty, researchers have proposed powerful service recommendation methods [2], [3], [4] in the past decade.

Although conventional service recommender systems have shown their effectiveness in solving the issue of information overload, they treat service recommendation as a one-round prediction task and ignore the interactions between mashup developers and recommender systems, leading to unsatisfied recommendation results and bad user experience. For example, after a developer inputs the requirement description of the mashup to be developed, the recommender will generate top-K recommendation list. Even when there is no required Web API in the list, the conventional recommender won't take further action. In this not uncommon circumstance, the recommender doesn't work to relieve the burden of API selection for mashup developers. This motivates the development of an interactive service recommender system in our work, which encourages developer-recommender interactions to help developers find their required APIs efficiently.

Interactive Service Recommendation (ISR) is formulated as a multi-round decision-making process. In each round, the recommender delivers a list of APIs to the developer and will receive feedbacks from her/him

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

(i.e., the developer selects each recommended API or not), according to which the recommender subsequently derives the next recommendation decision in a sequential manner. The recommendation-feedback interaction is repeated until the end of this visit session of the developer. Taking an ''itinerary planning'' mashup as an example, when the developer selects one service— Google Maps from the first round recommendation list, the developer's requirements are shifting and don't need map services anymore. Then, the recommender should not recommend additional map services (e.g., Microsoft Bing Maps), but other types of services (e.g., weather report services or hotel booking services) in the second round recommendation list. The goal of ISR is to explore developers' dynamic requirements, as well as to exploit the learned mashup profiles, to provide accurate service prediction, so as to optimize the outcome of the entire recommendation sequence.

The most similar works to ours are three recent models: HISR [5], DINRec [6] and iSRec [7]. They all do the next-round service recommendation, i.e., given the target mashup to be developed and some selected services, they make one-round prediction to recommend the following services. Although they can capture developers' shifting requirements to some extent, they only pay attention to the performance of the current round recommendation. For example, they don't consider the issue that they will generate the same ranking list when the current top-K recommendation list contains no ground-truth services. In such circumstance, the received feedbacks that the recommended services in current round are not required by the developer provide useful information that can help the recommender make better recommendations in later rounds. The existing next-round service recommendation approaches ignore the important influences from current round to later ones, and can merely provide locally optimal recommendations for each round. Instead, a more desired approach should focus on learning a globally optimal recommender taking all interaction rounds as a whole.

To capture developers' dynamic requirements and maximize the long-term performance over all interaction rounds, we propose a deep Reinforcement Learning based Interactive Service Recommendation model (RLISR). Specifically, we formulate the problem of interactive service recommendation as a Markov Decision Process (MDP) and design a Reinforcement Learning (RL) framework with three main components (i.e., state representation, reward setting and policy learning) to solve this problem. To obtain an informative state representation, we first construct a Knowledge Graph (KG) to effectively organize and make full use of related information of mashups and APIs, then propose a KG-based state representation learning method by integratedly utilizing the constructed KG, the contents of Web APIs and the historical interactions between the mashup developer and the recommender system. We also consider the positive and negative feedbacks at the same time to guarantee that the recommender can generate a new reasonable recommendation list for the next round even if there is no required services in the current round recommendation list. After that, we design a reward function which comprises ID-based reward signals to encourage the recommender to generate required APIs as accurate as possible and round-based reward signals to prompt the recommender to reduce the number of rounds to complete the whole recommendation. Next, we adopt a cascading Q-networks model to cope with the enormous combinatorial action space and learn an optimal policy to generate a recommendation list. Finally, we validate the effectiveness of our approach by comprehensive experimental results and analysis.

The contributions of our work can be summarized as follows:

- We provide an in-depth analysis of the issue of interactive service recommendation that have not been sufficiently discussed. Unlike existing service recommendation models that only pay attention to the performance of the single round or the next round, we formulate ISR as a multi-round decision-making process and model its target as maximizing the long-term performance of the whole recommendation process.
- We propose a RL-based interactive service recommendation model, which features a KG-based state representation component, an ID-based accuracy and round number sensitive reward function and a cascading Q-networks model based policy learning algorithm. To the best of our knowledge, this is the first effort to leverage deep RL to provide interaction ability for service recommender systems to boost the efficiency of mashup development.
- We conduct extensive experiments on a real-word dataset from ProgrammableWeb. The comparative results demonstrate that our approach achieves higher performance compared to the state-of-the-art service recommendation models.

The remainder of this paper is organized as follows. Section II introduces the related works. In Section III, we formulate the research problem. Section IV presents the proposed RLISR framework in detail. Section V describes the experimental settings, results and analysis. Section VI concludes this paper and proposes the future work.

II. RELATED WORK

Here we briefly review related work on two topics: service recommendation and interactive recommendation.

A. SERVICE RECOMMENDATION

Service recommendation for mashup development has been extensively studied. According to whether there are already some selected services or not, we divide existing service recommendation approaches into two types, i.e., single-round service recommendation and next round service recommendation.

Most service recommendation models are single-round ones, that only predict a top-K recommendation list once.

These methods can further be divided into three categories based on the information used: content-based, collaborative filtering (CF) based, and hybrid service recommendation models. **Content-based service recommendation approaches** typically make recommendations by measuring the similarity of the contents between mashups and services. Early studies used keyword-based approaches such as WordNet [8] and the vector space model [9] to match mashups and services. Although these methods are easy to implement, they can't really understand the semantics and recommend semantically relevant services. Therefore, Li et al. [10], and Zhong et al. [11] used topic modeling to explore the semantic relationships between mashups and services, and bridge the vocabulary gap between mashup developers and service providers. Shi et al. [12] proposed a text expansion and deep-model-based approach to deal with the data sparsity and vocabulary gap problem through expanding the description content at sentence level. To sum up, content-based service recommendation methods have relatively low performance for the limitation of textual and semantic similarities. **CF-based service recommendation methods** generally recommend the services used by the mashups which are most similar to the target mashup. Wu et al. [13] proposed a ratio-based method which get the similarity between developers or between services by comparing the attribute values directly. Zou et al. [14] integrated mashup-based and service-based similarity information into a singleton collaborative filtering algorithm and removed mashups/services dissimilar to the target mashup/service. Cao et al. [15] exploited the implicit co-invocation relationship between services to recommend diverse services for each mashup clusters. Nevertheless, CF-based methods can't guarantee that all services used by the most similar mashups can meet the target mashup's requirements. **Hybrid service recommendation approaches** integrate content and historical interaction information to make recommendations. Yao et al. [16] proposed a novel approach which considered simultaneously both rating data and semantic content data of web services by using a probabilistic generative model. Xiong et al. [17] designed a framework, in which invocation interactions between mashups and services as well as their functionalities are seamlessly integrated into a deep neural network to characterize the complex relations between mashups and services. Botangen et al. [18] proposed a method for data expansion of the matrix decomposition recommendation model using geographical location information and function description. Nguyen et al. [19] proposed an attentional PMF model that mines the relationship between services from both their contextual similarities and invocation history. Generally, hybrid methods achieve higher recommendation performance compared with the aforementioned two kinds of methods. There are many other classic works not listed here due to space limitation. Overall, single-round service recommendation models only recommend services once for a new mashup requirement regardless of the precision and recall of their recommendation results. This makes them inefficient to help developers select required services or even useless in the not uncommon circumstance that there is no ground-truth service in the recommendation list.

To help developers select the remaining services, next round service recommendation models have been proposed in the recent time. Xie et al. [7] proposed a model that can both make single-round recommendation and next round recommendation by integratedly utilizing related information in a way of constructing a heterogeneous information network. Ma et al. [5] designed a model to make next-round recommendation by capturing the interactions among the target mashup, selected services and the following service to recommend with an attention mechanism. Xiao et al. [6] designed a model named DINRec to make the following service recommendation by utilizing the composition and cooperation relationships between services.

Although next round recommendation methods can potentially be used to make multi-round service recommendation, they only focus on the performance of the current recommendation round, and overlook developers' long-term satisfaction over the whole recommendation process.

B. INTERACTIVE RECOMMENDATION

Interactive recommender systems (IRS) [20] have drawn huge attention recently because of its flexible recommendation strategy and the consideration of optimal long-term user experiences. One way to implement IRS is multi-armed bandit (MAB) methods [21], where recommending an item is regarded as picking an arm while user's feedback is collected as the reward of the corresponding arm selection. However, due to the limitation of the MAB modeling, these methods assume that the user preferences remain unchanged during the recommendation process, which goes against the dynamic nature of IRS. During the past years, the landscape of RL research has grown profoundly [22], [23], [24], resulting in more and more RL-based IRS. By formulating multi-step decision making as a Markov decision process, RL inherently considers how users' preferences evolve over time and maximizes their long-term satisfaction with recommendation results. Chen et al. [25] proposed a tree-structured RL-based recommendation framework, for solving three main challenges in IRS, i.e., time inefficiency, cold-start and data inefficiency. Zhou et al. [26] proposed a knowledge graph enhanced Q-learning framework for IRS to address the sample complexity issue. Wu et al. [27] considered the problem of cross-domain interactive recommendation and proposed a novel framework aiming to be doubly-adaptive to both static representation and dynamic interaction in IRS. Lei and Li [28] modeled the user-specific information of both preferences and relationships for IRS with explicit feedbacks. Zou et al. [29] introduced a model to simulate the environments and assist TD-based policy improvement for IRS. Though there still are other representative IRS models, they mainly focus on traditional recommendation fields, and there is no research work on long-term performance oriented interactive service recommendation to the best of our knowledge.

To sum up, interactive service recommendation is a meaningful research topic. On the one hand, handling the interactive nature in service recommendation scenario can boost long-term satisfaction of mashup developers, like popular social platforms (e.g., YouTube, TikTok, and Instagram) make interactive recommendation for their users. On the other hand, interactive service recommendation is a non-trivial problem for it has unique input and output information and feedback mode.

III. PRELIMINARIES

In this section, we introduce the related definitions and formulate the problem of interactive service recommendation for mashup creation. To help readers to understand easily, Table 1 lists the key notations used in the following sections.

TABLE 1. Notations.

Notations	Descriptions
$\mathcal{M} = \{m_1, ..., m_{\lvert\mathcal{M}\rvert}\}$	Set of all mashups in ISR.
$\mathcal{I} = \{i_1, ..., i_{\lvert\mathcal{I}\rvert}\}$	Set of all APIs in ISR.
$\mathcal{G} = \{\mathcal{E}, \mathcal{R}\}$	The service recommendation KG.
$\mathcal{P}_t = \{i_1, ..., i_p\}$	Set of APIs already selected for the developed mashup at round t.
$\mathcal{N}_t = \{i_1, ..., i_n\}$	Set of APIs recommended but not selected at round t.
$\mathcal{O}_t = \mathcal{P}_t \cup \mathcal{N}_t$	Observations of the environment at round t.
$\overline{\mathcal{P}}_t = \{i_1, ..., i_r\}$	Set of APIs remaining unfound of the developed mashup at round t.
$s_t, \mathbf{s}_t$	The current state and its corresponding representation at round t.
$\mathcal{A}_t = \{a^t_{1:K}\}$	Action at round t, i.e., recommending K APIs.
$r_t, r(\cdot,\cdot)$	The reward at round t and its calculation function.
$\Theta_{\mathcal{S}}$	Parameters of state representation network.
$\Theta_{\mathcal{Q}} = \{\Theta_1, ..., \Theta_K\}$	Parameters of K cascading Q-networks.
$\mathcal{D}$	Replay Buffer.

A. SETTING

A service ecosystem in this work refers to a service management platform to improve developer productivity and accelerate Web application delivery with speed and quality, in which there are a set of Mashups $\mathcal{M} = \{m_1, m_2, \ldots, m_{|\mathcal{M}|}\}$, a set of Web APIs $\mathcal{I} = \{i_1, i_2, \ldots, i_{|\mathcal{I}|}\}$, and their invocation information. To provide ISR functionality for a service ecosystem, the interactive model between the recommender system and the mashup developer is designed to be simplistic yet typical. It's set to be a multi-round interaction process that lasts for a period of time. At each round, according to the observations on past interactions, the recommender agent delivers K Web APIs in one list to a mashup developer. The developer provides feedback by picking up some or none of these APIs according to the requirements of the target mashup, and then the recommender agent recommends a new list of K services. This process continues until all the required services are selected out by the developer or the number of recommendation rounds reaches a predifined threshold. The advantage of this simplistic setting is that the recommender system will be easy to access the behavior of the active developer just by observing which services in the recommendation list are selected by her/him.

However, it's worth pointing out the limitations of the ISR setting assumed in the paper and our corresponding ISR model. First, we make service recommendation based on service functionality and don't consider the quality of services like [30]. Additionally, we only focus on a particular type of services, i.e., Web APIs, and don't consider other typical types of services, e.g., edge services [31] and IoT services [32]. Furthermore, it will be more usable and convenient if the developer can remove the selected services that she/he regret choosing, or can do active searching by inputing some information in the recommendation process to facilitate accurate recommendation. We don't consider these complicated circumstances and leave such supports in the future.

B. PROBLEM FORMULATION

The nature of developer interaction with a service recommender system is sequential and the problem of recommending the best services to a mashup developer is not only a prediction problem, but a sequential decision problem as well. This suggests that the ISR problem could be modeled as a Markov Decision Process (MDP) and be solved by RL algorithms. Since RL is able to take into account the long-term developer engagement with the recommender system, it holds the promise to help mashup developer select all required services out as quickly as possible.

To improve the potential recommendation performance of RL models with only a limited amount of service interaction data, we utilize knowledge graph (KG) techniques to guide the RL-based learning method for ISR. Given a service ecosystem, there exist various types of objects (e.g. mashups and Web APIs) and rich relationships among them, which can be modeled by constructing a Service Recommendation Knowledge Graph (SRKG).

Definition 1 (SRKG): The SRKG can be defined as a set of triplets $\mathcal{G} = \{(e_h, r, e_t)|(e_h, e_t \in \mathcal{E}) \wedge (\phi(e_h), \phi(e_t) \in \mathcal{E}), (r \in \mathcal{R}) \wedge (\psi(r) \in \mathcal{R})\}$, where $\mathcal{E}$ is the set of entities, $\phi : \mathcal{E} \rightarrow \mathcal{E}$ is an entity type mapping function and each entity $e \in \mathcal{E}$ belongs to an entity type $\phi(e) \in \mathcal{E}$; $\mathcal{R}$ is the set of relations connecting two entities, $\psi : \mathcal{R} \rightarrow \mathcal{R}$ is a relation type mapping function and each relation $r \in \mathcal{R}$ belongs to a relation type $\phi(r) \in \mathcal{R}$. Each triplet $(e_h, r, e_t) \in \mathcal{G}$ characterizes the semantic relatedness between a head entity $e_h \in \mathcal{E}$ and a tail entity $e_t \in \mathcal{E}$ with the relation $r \in \mathcal{R}$.

According to existing studies [33], [34], not all information accumulated in a service ecosystem contributes to the accuracy of service recommendations. So we construct SRKG by only using the most useful knowledge. Specifically, $\mathcal{E} = \{M, S, C, T\}$ and $\mathcal{R} = \{MS, SM, MC, SC, MT, ST\}$, where M, S, C and T denote four main types of objects, i.e., mashups, Web APIs, categories and topics respectively; MS, SM, MC, SC, MT and ST denote six main types of relations, i.e., the *composed-by/composing* relations between mashups and Web APIs, the *labeled-with* relations from mashups to their categories, and from Web APIs to their categories, the *described-by* relations from mashups to their topics, and from Web APIs to their topics. We adopt a vanilla

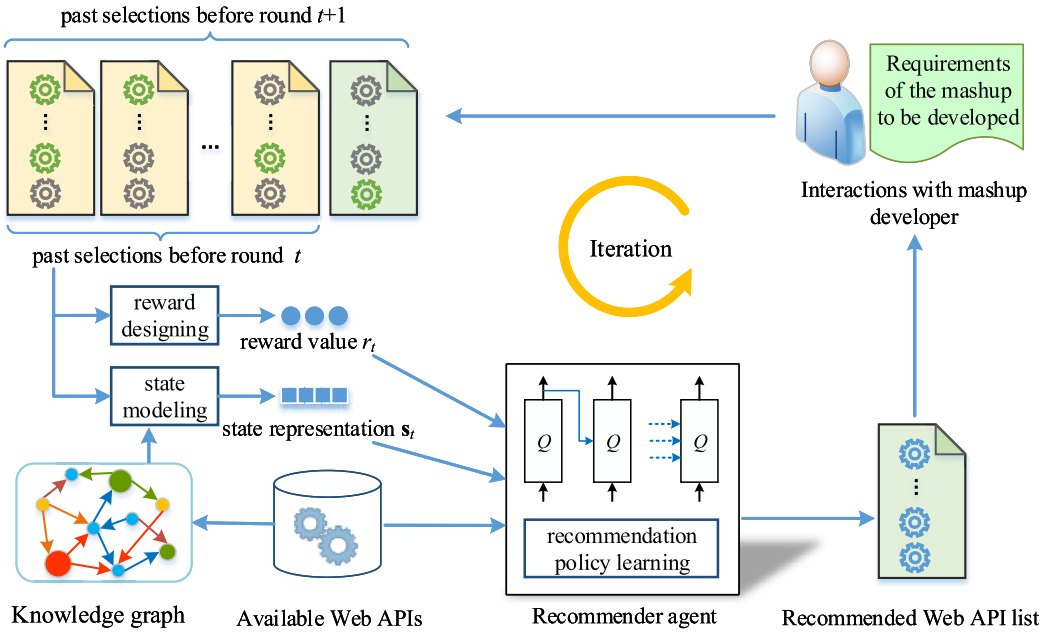


FIGURE 1. The overall framework of RLISR.

probabilistic topic modeling method—LDA to identify the entities with type T and extract relations with types MT and ST from description contents of mashups and Web APIs.

Considering the mashup developer and the service ecosystem with its constructed SRKG as the environment and the service recommendation algorithm as an RL agent, the ISR process is formulated as an MDP in this work, in which the key components are defined as follows.

- **State $\mathcal{S}$:** This is a continuous state space that describes the environment states. A state $s_t \in \mathcal{S}$ at round t is determined by both the requirements of the mashup to be developed and the interaction history between the mashup developer and the recommender agent before t. SRKG can provide rich and distinguishable information for mashups and Web APIs by their latent knowledge-level connection to generate state representation.
- **Action $\mathcal{A}$:** This is a discrete action space. An action $\mathcal{A}_t = \{a^t_{1:K}\} = \{i^t_1, \ldots, i^t_K\} \in \binom{\mathcal{I}_t}{K}$ is composed of K atomic actions, and each atomic action $a^t_j \in \{a^t_{1:K}\}$ $(1 \leq j \leq K)$ predicts one Web API i^t_j for recommendation. $\binom{\mathcal{I}_t}{K}$ means the set containing all subsets of K items of $\mathcal{I}_t$, where $\mathcal{I}_t \in \mathcal{I}$ are available services to be recommended at round t. Thus, $\mathcal{A}_t$ is a subset of K APIs chosen by the recommender agent based on current state s_t from $\mathcal{I}_t$ to display to the mashup developer.
- **Reward $\mathcal{R}$:** Once the agent chooses a suitable action a_t based on the current state s_t at round t, i.e., recommending a new list of APIs to a developer, the developer will browse these APIs and pick out the APIs to be used. Developer's feedback on the recommended APIs accounts for the reward $r(s_t, \mathcal{A}_t)$, which is used to improve the policy π adopted by the recommender agent.
- **Transition probability $\mathcal{P}$:** The transition probability $p(s_{t+1}|s_t, a_t)$ is defined as the probability of state transition from s_t to s_{t+1} when action a_t is taken by the agent. Because we adopt model-free RL methods, $p(s_{t+1}|s_t, a_t) = 1$, i.e., once a new list of APIs is recommended and the corresponding developer's feedback is provided, the state transition is determined.
- **Discount factor γ:** The discount factor $\gamma \in [0, 1]$ is used to balance between future and immediate rewards. With $\gamma = 0$, the recommender agent becomes *myopic*, i.e., it only focuses on immediate reward. On the contrary, if $\gamma = 1$, the agent becomes *farsighted* and focuses more on future rewards.

With the notations and definitions above, the ISR problem can be formally defined as follows: Given the historical MDP, i.e., $(\mathcal{S}, \mathcal{A}, \mathcal{P}, \mathcal{R}, \gamma)$, the goal is to find a recommendation policy π^* parametrized by θ to maximize the expected discounted cumulative reward over the whole ISR process as

$$\pi^* = \arg\max_{\pi_\theta} \mathbb{E}\left[\sum_{t=0}^{\tau}\sum_{k=1}^{K} \gamma^t r(s_t, \mathcal{A}_t)\right], \tag{1}$$

where τ is the round number of the whole recommendation process.

IV. THE PROPOSED MODEL

In this section, we detail the proposed RLISR model. As illustrated in Fig. 1, it supports the interaction between mashup developers and recommender agent in an iterative way, and

contains three main components: state representation, reward setting and recommendation policy learning. Generally, In RLISR, the recommender agent obtains state embeddings via the state representation component which is modeled by a dedicated neural network and takes the environment as inputs. Then, the agent conducts recommendation under the policy which is learned by the recommendation policy learning component that models the policy as a cascading Q-networks model [24] and updates the policy according to the received rewards. The rewards are calculated by a reward function designed to both consider the recommendation accuracy and the round number to complete the whole recommendation process. At last, the parameters of state representation and policy learning are calculated at the same time through the RL trial-and-error process.

A. SRKG-BASED STATE REPRESENTATION

State representation is a fundamental component in RLISR and should summarize information from the environment containing developers, APIs, and the context. As shown in Fig. 2, we assume that the recommender system is performing the t-th round recommendation. To model and learn a good state representation, we encode the characteristics of 3 kinds of information for a state, i.e., the service recommendation knowledge graph (SRKG), the contents of Web APIs and the historical interactions between the mashup developer and the recommender system. To be specific, a GCN model is utilized to incorporate rich information to generate representations of mashups and APIs, a Doc2vec-based pre-training model is utilized to exploit the contents of Web APIs to enhance API representation, and a GRU model is adopted to encode the historical interactions of the mashup developer.

1) SRKG-BASED ENCODING FOR MASHUPS AND WEB APIS

It's a basic and critical task to represent mashups and APIs for state encoding. As mentioned in Section III-B, we construct SRKG to organize and exploit rich related information for service recommendation. Since SRKG links mashups, APIs and other entities like categories and topics together, we can take advantage of the semantic and correlation information among entities in SRKG for good mashup and API embedding. We build upon the architecture of graph convolution network (GCN) [35] to distill structural and semantic knowledge in SRKG into a low-dimensional dense node representation.

The computation of the node's representation in a single graph convolutional embedding layer is a two-step procedure: information propagation and embedding aggregation. These two procedures can naturally be extended to multiple hops to capture high-order structural proximity among entities. We use the notation k to identify the k-th hop, and denote the representation of a head entity h or a tail entity t at the k-th hop as $\mathbf{e}_h^k \in \mathbb{R}^{d_e}$ or $\mathbf{e}_t^k \in \mathbb{R}^{d_e}$. In each layer, first, we employ knowledge-aware attention mechanism [36] to perform information propagation between a given head

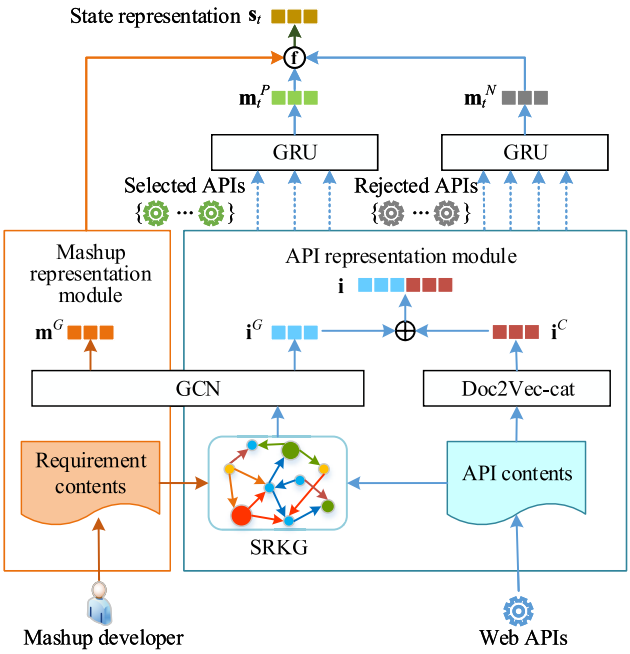


FIGURE 2. SRKG-based state representation, in which mashup representation is generated by SRKG embedding, API representation is generated by SRKG embedding and enhanced with content pre-training, and positive and negative feedbacks are all considered to form the final state representation.

entity h and its neighbors:

$$\mathbf{e}_{\mathcal{N}_h}^{k-1} = \sum_{(h,r,t)\in\mathcal{N}_h} \tilde{\pi}(h,r,t)\mathbf{e}_t^{k-1}, \tag{2}$$

where $\mathcal{N}_h = \{(h,r,t)|(h,r,t) \in \mathcal{G}\}$ denotes the set of triplets in which h is the head entity. $\tilde{\pi}(h,r,t)$ is the normalized value of a coefficient $\pi(h,r,t)$ across all triplets connected with h by adopting the softmax function:

$$\tilde{\pi}(h,r,t) = \frac{\exp(\pi(h,r,t))}{\sum_{(h,r',t')\in\mathcal{N}_v}\exp(\pi(h,r',t'))}, \tag{3}$$

where $\pi(h,r,t)$ acts as a filter controlling the decay degree on each propagation on edge (h,r,t) and indicating how much information being propagated from t to h conditioned on relation r. It's implemented via relational attention mechanism in this paper, which is formulated as follows:

$$\pi(h,r,t) = (\mathbf{W}_r\mathbf{e}_t^{k-1})^\top \tanh\left((\mathbf{W}_r\mathbf{e}_h^{k-1}) + \mathbf{e}^r\right), \tag{4}$$

where $\mathbf{e}^r \in \mathbb{R}^{d_r}$ denotes the embedding of relation r, and $\mathbf{W}^r \in \mathbb{R}^{d_r\times d_e}$ is the transformation matrix of r, projecting entities from the d_e-dimensional entity space into the d_r-dimensional relation space. This makes $\pi(h,r,t)$ dependent on the distance between e_h and e_t in the relation r's space and capable of suggesting which neighbor nodes should be given more attention to capture collaborative signals. We select tanh as the nonlinear activation function and employ inner product on these representations for simplicity.

The second procedure is to aggregate the entity representation $\mathbf{e}_h^{k-1}$ and its neighborhood representation $\mathbf{e}_{\mathcal{N}_h}^{k-1}$ as the

new representation $\mathbf{e}_h^k$ of entity h:

$$\mathbf{e}_h^k = \text{LeakyReLU}\left(\mathbf{W}_k(\mathbf{e}_h^{k-1} + \mathbf{e}_{\mathcal{N}_h}^{k-1}) + \mathbf{b}_k\right), \quad (5)$$

where we sum two representations up and apply a non-linear transformation by setting the activation function as LeakyReLU. $\mathbf{W}_k$ and $\mathbf{b}_k$ are trainable parameters for k-hop neighborhood aggregator.

After performing K layers, we obtain multi representations for entity h, namely $\{\mathbf{e}_h^0, \mathbf{e}_h^1, \ldots, \mathbf{e}_h^K\}$. The output of different layers emphasizes different levels of connectivity information. Therefore, we obtain the final representation by concatenating the representation of each step into a single vector, as shown below.

$$\mathbf{e}_h^* = \mathbf{e}_h^0 \| \mathbf{e}_h^1 \| \cdots \| \mathbf{e}_h^K. \quad (6)$$

For notational convenience, we denote the representation of API node i generated from the previous K information propagation and embedding aggregation steps as $\mathbf{i}^G$ (i.e., $\mathbf{i}^G = \mathbf{e}_h^*$). Generally speaking, $\mathbf{i}^G$ is a mixture of initial representations of i and its neighbors up to K hops away. As the same way, we denote the representation of mashup node m as $\mathbf{m}^G$.

2) CONTENT ENCODING FOR WEB APIS

In ISR scenario, each Web API has its contents (i.e., description content and categories) which are important to identify its functionalities. To incorporate such valuable information into state representation directly, we adapt Doc2vec model [37] to learn a fixed length feature vector for each API. Doc2vec is an unsupervised model to provide fixed-length representations for variable-length pieces of contents, such as sentences, paragraphs, and documents. However, besides description content, the categories a service belongs to are significant to distinguish it. So, we propose a Doc2vec-cat model to encode the description content and categories at the same time, and here have a brief introduction of it.

As shown in Fig. 3, the distributed memory model of paragraph vectors [37] is adopted. Every description content is mapped to a unique vector which is represented by a column in matrix D, every category is mapped to a unique vector which is represented by a column in matrix C, and every word is also mapped to a unique vector which is represented by a column in matrix W. Given an API i, its category vector is formed by averaging the vectors of its categories. Then, the representation of i is formed by concatenating its description vector and its category vector. The API representation is asked to contribute to the prediction task of the next word given many contexts sampled from its description content. Specifically, word vectors are averaged and then concatenated with the representation of i to predict the next word in a context. After being trained, the obtained representation can be used as the content encoding of i, and is denoted as $\mathbf{i}^C$.

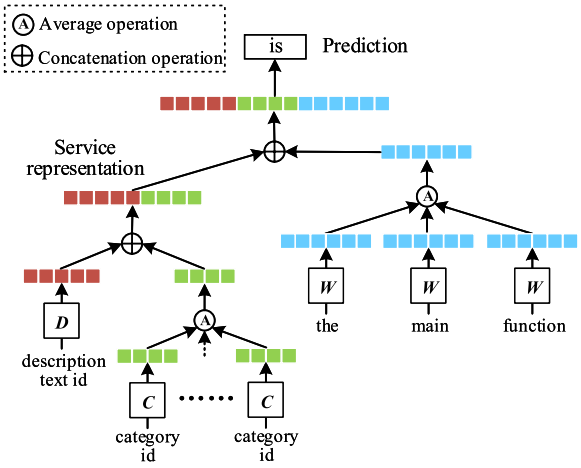


FIGURE 3. Framework of Doc2vec-cat model.

3) HISTORICAL INTERACTION ENCODING

For an API i, we can obtain its content embedding $\mathbf{i}^C$ and its graph embedding $\mathbf{i}^G$ through the above two steps. Its final embedding $\mathbf{i}$ is formed by concatenating these two embeddings, i.e.,

$$\mathbf{i} = \mathbf{i}^C \| \mathbf{i}^G. \quad (7)$$

Then, we can encode the most important information for state representation, i.e., the historical interactions between the mashup developer and the recommender system.

ISR is a sequential decision-making process. At round t, it is natural to form a selected API sequence $\mathcal{P}_t = \{i_1, i_2, \ldots, i_p\}$ and an unselected API sequence $\mathcal{N}_t = \{i_1, i_2, \ldots, i_n\}$ ordered by the recommendation round and the rank in the recommendation list for each round. $\mathcal{P}_t$ is the cumulative positive feedbacks of the mashup developer and represents the already satisfied functional requirements of the target mashup. $\mathcal{N}_t$ is the cumulative negative feedbacks of the mashup developer and represents the unrelated functional requirements or unmatched APIs of the target mashup. We call $\mathcal{P}_t$ and $\mathcal{N}_t$ as the observations $\mathcal{O}_t$ of the environment (i.e., $\mathcal{O}_t = \mathcal{P}_t \cup \mathcal{N}_t$). They are both significant for predicting APIs to be possibly used next to complete the mashup. Due to the sequential nature of $\mathcal{P}_t$ and $\mathcal{N}_t$, any sequential neural model (e.g., LSTM or Transformer) can be adopt to encode them. Here, we utilize a classic neural network—GRU [38] for its simplicity and efficiency. The encoding of mashup developer's historical interactions with the recommender system is calculated as follows:

$$\begin{aligned} \mathbf{m}_t^P &= \text{GRU}(\mathcal{P}_t) \\ \mathbf{m}_t^N &= \text{GRU}(\mathcal{N}_t), \end{aligned} \quad (8)$$

where GRU denotes the whole GRU neural model. The structure of its component blocks is not detailed anymore for it isn't the focus point of this work.

4) THE FINAL STATE REPRESENTATION

Based on the above discussions, we are ready to give the final state representation in our model. For a state s_t, its

representation $\mathbf{s}_t$ is the combination of three representation vectors:

$$\mathbf{s}_t = \mathbf{m}^G + \mathbf{m}_t^P - \mathbf{m}_t^N, \tag{9}$$

where "+" and "−" denote the element-wise addition and subtraction operator respectively. $\mathbf{m}^G$ is the graph embedding of mashup m obtained via Eq.(6), $\mathbf{m}_t^P$ and $\mathbf{m}_t^N$ are the interaction embeddings at round t calculated via Eq.(8). It is worth mentioning that $\mathbf{i}^C$ is obtained through pre-training, while $\mathbf{i}^G$, $\mathbf{m}^G$, $\mathbf{m}_t^P$ and $\mathbf{m}_t^N$ are all learned in an end-to-end manner.

For simplicity, the set of the whole network parameters for computing $\mathbf{s}_t$, including parameters of SRKG-enhanced encoding module and parameters of historical interaction encoding module, is denoted as $\Theta_{\mathcal{S}}$.

B. REWARD SETTING

The reward signal from environment reflects how good or bad the agent is performing through developer's selecting actions. Therefore, designing informative reward signal is critical for success of the recommender agent. In ISR scenario, the ultimate goal is to recommend all required APIs as few rounds as possible. This needs the recommendation list generated in each round is of high performance in terms of accuracy.

1) REWARD DECOMPOSITION

Based on the above consideration, at round t, we define the reward function by integrating two different reward functions:

$$r(s_t, \mathcal{A}_t) = r_{id}(\mathcal{A}_t, \overline{\mathcal{P}}_t) + r_{rd}(\mathcal{A}_t, \overline{\mathcal{P}}_t), \tag{10}$$

where the action $\mathcal{A}_t = \{i_1^t, \ldots, i_K^t\}$ is the recommendation list at round t; $\overline{\mathcal{P}}_t = \{i_1^t, \ldots, i_r^t\}$ denotes the set of APIs remaining unfound of the developed mashup at t; $r_{id}(\cdot,\cdot)$ and $r_{rd}(\cdot,\cdot)$ measure the rewards on recommendation accuracy and round number to complete recommendation, respectively. Next we discuss how to set $r_{id}(\cdot,\cdot)$ and $r_{rd}(\cdot,\cdot)$ for our task.

2) ID-BASED REWARD

In recommender systems, the final performance is usually measured based on the exact match of item IDs, and many ID-based metrics are designed. We borrow a classic metric—NDCG for ID-based reward function design. Formally, given the ground-truth remaining API set $\overline{\mathcal{P}}_t$ and the recommendation list $\mathcal{A}_t$, we define the reward function as:

$$r_{id}(\mathcal{A}_t, \overline{\mathcal{P}}_t) = \sum_{j=1}^{K} \frac{rel_j^{\overline{\mathcal{P}}_t}}{\log_2^{j+1}} \Big/ \sum_{j=1}^{K} \frac{1}{\log_2^{j+1}}, \tag{11}$$

where $rel_j^{\overline{\mathcal{P}}_t}$ is an indicator equaling 1 if the API $i_j^t \in \mathcal{A}_t$ at rank j in the recommendation list is selected by the mashup developer, zero otherwise. As we can see, $r_{id}(\cdot,\cdot)$ is a full position-aware reward function which assigns larger weights on higher rank positions. It encourages the recommendation agent to generate required APIs as early as possible.

3) ROUND-BASED REWARD

In the second reward function, we consider measuring the quality of action $\mathcal{A}_t$ from the perspective of reducing the round number to complete the whole recommendation. This reward function is defined as:

$$r_{rd}(\mathcal{A}_t, \overline{\mathcal{P}}_t) = \begin{cases} p, & \overline{\mathcal{P}}_t - \mathcal{A}_t \neq \varnothing \\ 0, & \overline{\mathcal{P}}_t - \mathcal{A}_t = \varnothing \end{cases} \tag{12}$$

where $p \in (-1, 0)$ is a penalty for not finishing recommendation at the current round t (i.e., $\overline{\mathcal{P}}_t - \mathcal{A}_t \neq \varnothing$). Otherwise, the reward value is 0. This prompts the recommendation agent to finish recommendation as few rounds as possible.

By plugging Eq.(11) and Eq.(12) into Eq.(10), we can derive the final reward function. By providing these two kinds of reward signals, we expect the RL algorithm can be guided to yield a better service recommendation performance.

C. RECOMMENDATION POLICY LEARNING

After modeling the mashup's state $\mathbf{s}_t$ and designing the reward function $R(s_t, \mathcal{A}_t)$, we can then learn a recommendation policy to combine these two kinds of information for interactive service recommendation. The recommendation policy needs to choose from a combinatorial action space $\binom{\mathcal{I}_t}{K}$, where each action is a subset of K APIs chosen from a larger set $\mathcal{I}_t$ of $|\mathcal{I}_t|$ candidates. The action space can be very large even for moderate $|\mathcal{I}_t|$ (e.g., 1,000) and K (e.g., 5). To meet the challenge of the potentially high computational complexity of the combinatorial action space, we adopt a cascading Q-networks model [24] to learn an optimal policy to estimate the long-term reward for a combination of APIs. It can be regarded as a cascaded deep RL model [22], which utilizes multiple RL agents sequentially to achieve the final solution and each agent solves a sub-problem that is a step towards the solution of a complex problem. However, different with [22], the cascading Q-networks are dependent.

1) CASCADING Q-NETWORKS MODEL

Generally, if the size of the recommendation list is K, the cascading Q-networks model consists of K Q-networks which are connected in a cascading manner and select K optimal items for recommendation in order. It uses the Q-learning framework where an optimal action-value function $Q^*(\mathbf{s}, \mathcal{A})$ will be learned and satisfies $Q^*(\mathbf{s}_t, \mathcal{A}_t) = \mathbb{E}[r(s_t, \mathcal{A}_t) + \gamma \max_{\mathcal{A}_{t+1} \subset \mathcal{I}_{t+1}} Q^*(\mathbf{s}_{t+1}, \mathcal{A}_{t+1})]$. Once the action-value function is learned, an optimal policy for recommendation can be obtained as:

$$\pi^*(\mathbf{s}_t, \mathcal{A}_t) = \arg\max_{\mathcal{A}_t \subset \mathcal{I}_t} (Q^*(\mathbf{s}_t, \mathcal{A}_t)), \tag{13}$$

where $\mathcal{I}_t \subset \mathcal{I}$ is the set of items available at time t.

As illustrated in Fig. 4, the cascading Q-networks model uses a set of K related Q-functions to address the enormous combinatorial action space and generate the optimal K-APIs combination. Denote a recommender action as $\mathcal{A} = \{a_{1:K}\} \subset \mathcal{I}$ and the optimal action as $\mathcal{A}^* = \{a_{1:K}^*\} =$

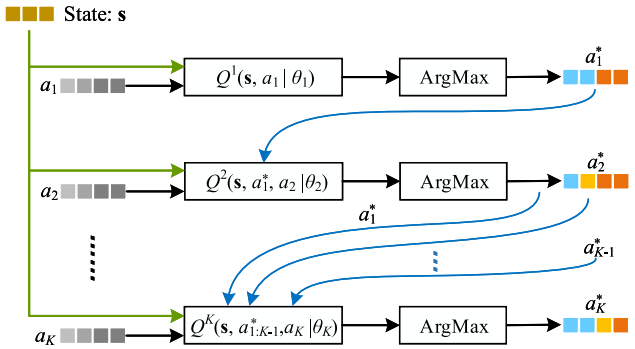


FIGURE 4. Framework of cascading Q-networks model.

$\arg\max_{\mathcal{A}} Q^*(\mathbf{s}, \mathcal{A})$. The cascading Q-networks model is inspired by the key fact:

$$\max_{a_{1:K}} Q^*(\mathbf{s}, a_{1:K}) = \max_{a_1} \left(\max_{a_{2:K}} Q^*(\mathbf{s}, a_{1:K}) \right). \tag{14}$$

Based on this fact, a set of mutually consistent functions $Q^{1*}, \ldots, Q^{K*}$ is defined to obtain each optimal atomic action $a_k^* \in \{a_{1:K}^*\}$ as:

$$\begin{cases} a_1^* = \arg\max_{a_1} \left\{ Q^{1*}(\mathbf{s}, a_1) := \max_{a_{2:K}} Q^*(\mathbf{s}, a_{1:K}) \right\}, \\ a_2^* = \arg\max_{a_2} \left\{ Q^{2*}(\mathbf{s}, a_1^*, a_2) := \max_{a_{3:K}} Q^*(\mathbf{s}, a_{1:K}) \right\}, \\ \ldots \\ a_K^* = \arg\max_{a_K} \left\{ Q^{K*}(\mathbf{s}, a_{1:K-1}^*, a_K) := Q^*(\mathbf{s}, a_{1:K}) \right\}. \end{cases} \tag{15}$$

Thus, an optimal action $\mathcal{A}^*$ can be obtained in $O(K|\mathcal{I}|)$ computations by applying these functions in a cascading manner.

Each Q^{k*} function is parameterized by a multi-layer perceptron (MLP) in this work:

$$Q^{k*} = \mathbf{q}_k^\top \sigma \left(\mathbf{W}_k \left[\mathbf{s} \| \mathbf{i}_1^* \| \cdots \| \mathbf{i}_{k-1}^* \| \mathbf{i}_k \right]^\top + \mathbf{b}_k \right), \ \forall k, \tag{16}$$

where σ is the sigmoid activation function, $\mathbf{s}$ is the state embedding obtained via Eq.(9), $\mathbf{i}_j^* (1 \le j \le k-1)$ is the API embedding obtained via Eq.(7) corresponding to the optimal atomic action $\mathbf{a}_j^*$, and $\mathbf{i}_k$ is the embedding of the candidate API $i_k \in \mathcal{I}$; $\mathbf{W}_k \in \mathbb{R}^{d_n \times (d_s + d_i * k)}$, $\mathbf{q}_k \in \mathcal{R}^{d_n}$ and $\mathbf{b}_k \in \mathcal{R}^{d_n}$ are the set Θ_k of parameters, and d_n, d_s and d_i are the dimension of the MLP hidden layer, of the state embedding and of the API embedding, respectively. We denote the set of all the parameters of the cascading Q-networks model as $\Theta_{\mathcal{Q}}$ (i.e., $\Theta_{\mathcal{Q}} = \{\Theta_1, \ldots, \Theta_K\}$).

2) MODEL TRAINING

With the proposed framework, we can train the parameters of the model through trial-and-error process. During the interactive recommendation process, at round t, the recommender agent (i.e., the cascading Q-functions) gets the current state $\mathbf{s}_t$ from the observations $\mathcal{O}_t$ (i.e., the feedbacks: $\mathcal{P}_t \cup \mathcal{N}_t$) of the mashup developer, and recommends the next list $\mathcal{A}_t$ of K APIs via an ε-greedy policy (i.e., with probability ε choosing a random subset $\mathcal{A}_t$ of size K, and with probability $1-\varepsilon$ choosing the optimal action $\mathcal{A}_t^*$ obtained by the cascading Q-functions). Then the agent receives the reward r_t from the developer's selection and stores the experience $(\mathcal{O}_t, \mathcal{A}_t, r_t, \mathcal{O}_{t+1})$ in the replay buffer $\mathcal{D}$. From $\mathcal{D}$, we sample mini-batch of experiences to update the parameters $\Theta_{\mathcal{S}}$ of state representation and the parameters of the cascading Q-functions $\Theta_{\mathcal{Q}} = \{\Theta_1, \ldots, \Theta_K\}$.

However, the set of Q^{k*} functions need to satisfy a large set of constraints which it is not easy to strictly enforce. At the optimal point, the value of Q^{k*} is the same as Q^* for all k, i.e.,

$$Q^{k*}(\mathbf{s}, a_1^*, \ldots, a_k^*) = Q^*(\mathbf{s}, a_1^*, \ldots, a_K^*), \quad \forall k. \tag{17}$$

These constraints are taken into account in a soft and approximate way in this work. That is, we use the same final target to train all the Q^{k*} functions. For Q^{k*}, we define its corresponding mean-square loss function as:

$$\mathcal{L}(\Theta_k) = \mathbb{E}_{(\mathcal{O}_t, \mathcal{A}_t, r_t, \mathcal{O}_{t+1}) \sim \mathcal{D}} [(y_t - Q^k(\mathbf{s}_t, a_{1:k}^t))^2], \quad \forall k, \tag{18}$$

where $a_{1:k}^t$ are the first k APIs in $\mathcal{A}_t$. y_t is the target value based on the optimal Q^* and defined as:

$$y_t = r(s_t, \mathcal{A}_t) + \gamma Q^K(\mathbf{s}_{t+1}, a_{1:K}^{(t+1)*} | \Theta_K), \tag{19}$$

where $a_{1:K}^{(t+1)*}$ are the optimal atomic actions at round $t+1$ estimated by the K neural network functions according to the new state representation $\mathbf{s}_{t+1}$. Then, all the Q^k networks are fitting against the same target y_t outputted ultimately by the last network Q^K. To alleviate the overestimation problem in original Q-networks, we also adopt the double DQN architecture [39], i.e., each online Q-network Q^k is attached with a target Q-network $Q^{k'}$. The online network back-propagates and updates its weights at each training step. The target network is a duplicate of the online network and updates its parameters with training delay. The target value for each online network Q^k to update is then changed to

$$y_t = r(s_t, \mathcal{A}_t) + \gamma Q^{K'}(\mathbf{s}_{t+1}, a_{1:K}^{(t+1)*} | \Theta'_K), \tag{20}$$

where $a_{1:K}^{(t+1)*}$ are the optimal atomic actions estimated by the online cascading networks, and Θ'_K denotes the parameter of the last target Q-network $Q^{K'}$. The parameters of target Q-networks $\Theta'_{\mathcal{Q}} = \{\Theta'_1, \ldots, \Theta'_K\}$ are updated according to soft assign as:

$$\Theta'_{\mathcal{Q}} = \delta \Theta_{\mathcal{Q}} + (1-\delta) \Theta'_{\mathcal{Q}}, \tag{21}$$

where $\delta \in (0, 1)$ is the update frequency. Based on the loss function defined in Eq.(18), the parameters $\Theta_{\mathcal{Q}} = \{\Theta_1, \ldots, \Theta_K\}$ of the recommender agent besides the parameters $\Theta_{\mathcal{S}}$ of state representation can be updated by performing gradient steps over the above loss.

To summarize, the training procedure of our RLISR is presented in Algorithm 1. For each mashup m in the training set, we apply ε-exploration technique and employ

the cascading Q-functions to search the optimal action $\mathcal{A}_t = \{a_{1:K}^{(t)}{}^*\}$ to recommend. The system's experiences at each time-step are stored in the replay buffer $\mathcal{D}$ and then a mini-batch of data is sampled from $\mathcal{D}$ to update the parameters of RLISR.

Algorithm 1 Parameters Training for RLISR

Input: The pre-trained content embeddings of APIs $\{\mathbf{i}^C\}$; SRKG $\mathcal{G}$; replay buffer $\mathcal{D}$; discount factor γ; probability ε; update frequency δ.
Output: Parameters $\Theta_{\mathcal{S}}$, $\Theta_{\mathcal{Q}}$.
repeat
  for $m \in \mathcal{M}$ **do**
    $t \leftarrow 0$, $\mathcal{O}_t \leftarrow \varnothing$;
    while $\overline{\mathcal{P}}_t \neq \varnothing$ **do**
      forall $\{i_1, i_2, \ldots, i_n\} \in \mathcal{O}_t$ **do**
        Get $\{\mathbf{i}_1, \mathbf{i}_2, \ldots, \mathbf{i}_n\}$ via Eq.(6) and Eq.(7);
      Get $\mathbf{s}_t$ via Eq.(8) and Eq.(9);
      With probability ε select a random action $\mathcal{A}_t$ of size K, otherwise $\mathcal{A}_t \leftarrow \{a_{1:K}^{(t)}{}^*\}$;
      Recommend $\mathcal{A}_t$ to the developer of mashup m, observe her/his selections and update $\mathcal{O}_{t+1}$ and $\overline{\mathcal{P}}_{t+1}$;
      Calculate reward r_t via Eq.(10);
      Store $(\mathcal{O}_t, \mathcal{A}_t, r_t, \mathcal{O}_{t+1})$ in $\mathcal{D}$;
      $t \leftarrow t + 1$;
    Sample mini-batch of transitions $(\mathcal{O}_t, \mathcal{A}_t, r_t, \mathcal{O}_{t+1}) \overset{\text{iid.}}{\sim} \mathcal{D}$;
    Get $\mathbf{s}_t, \mathbf{s}_{t+1}$ from $\mathcal{O}_t, \mathcal{O}_{t+1}$ via Eq.(7) and Eq.(9);
    Calculate target values y_t via Eq.(20);
    Update $\Theta_{\mathcal{S}}$, $\Theta_{\mathcal{Q}}$ via SGD over the loss Eq.(18);
    Update $\Theta'_{\mathcal{Q}}$ via Eq.(21);
until *converged*;

V. EXPERIMENTS

In this section, we first introduce the dataset used to verify the effectiveness of RLISR, then present the evaluation metrics, baselines and experimental settings. Afterwards, we report and analyze the experimental results.

A. EXPERIMENTAL SETTINGS

1) DATASET DESCRIPTION

Owing to the interactive paradigm of ISR, a straight forward way to train and test it is to conduct online experiments. However, applying a trial-and-error strategy in online scenarios would hurt the users' experience. Therefore, following the experimental protocol of the interactive recommendation research community [25], [26], we use an offline real-world dataset to conduct experiments to obtain solid experimental results. The dataset is crawled from ProgrammableWeb and released publicly[1] [40].

The original dataset consists of 7884 mashups and 23581 APIs. To adapt to multi-round service recommendation evaluation, the mashups with fewer than two APIs were removed. At the same time, APIs that have never been used were removed in the evaluation. The dataset after preprocessing contains 2906 mashups and 1322 APIs. More detailed information is provided in Table 2.

TABLE 2. Statistic information of the experimental dataset.

Parameters	Values
Number of mashups	2906
Number of APIs	1322
Number of interactions	9739
Number of categories	400
Average number of APIs per Mashup	3.35

For the sake of simulating the service recommendation for mashup creation in the real scene, similar to work [41], [42], we sorted the mashups by when they were created. The data before a certain time was selected as the training set, and the part of the data after that time was used as the test set. To be specific, the time is set to Apr. 10th, 2012, the training set includes the previous 2325 mashups and the test set includes the remaining 581 mashups.

2) EVALUATION METRICS

We adopt five widely used metrics for performance evaluation on top-K recommendation results: *Precision*@K, *Recall*@K, $F1$@K, mean average precision (*MAP*@K) and Normalized Discounted Cumulative Gain (*NDCG*@K). Given a list of top K predicted APIs for mashup m, denoted as $\mathcal{A}^m$, and its actual component API set, denoted as $\overline{\mathcal{P}^m}$, The first three metrics are computed by:

$$
\begin{aligned}
Precision@K &= \frac{1}{|\mathcal{M}|}\sum_{m\in\mathcal{M}}\frac{|\mathcal{A}^m \cap \overline{\mathcal{P}^m}|}{K},\\
Recall@K &= \frac{1}{|\mathcal{M}|}\sum_{m\in\mathcal{M}}\frac{|\mathcal{A}^m \cap \overline{\mathcal{P}^m}|}{|\overline{\mathcal{P}^m}|},\\
F1@K &= \frac{1}{|\mathcal{M}|}\sum_{m\in\mathcal{M}} 2\times\frac{\frac{|\mathcal{A}^m\cap\overline{\mathcal{P}^m}|}{K}\times\frac{|\mathcal{A}^m\cap\overline{\mathcal{P}^m}|}{|\overline{\mathcal{P}^m}|}}{\frac{|\mathcal{A}^m\cap\overline{\mathcal{P}^m}|}{K}+\frac{|\mathcal{A}^m\cap\overline{\mathcal{P}^m}|}{|\overline{\mathcal{P}^m}|}}.
\end{aligned}
\tag{22}
$$

Intuitively, *Precision*@K indicates what percentage of recommended top K APIs (i.e., $\mathcal{A}^m$) are really used by mashup m, *Recall*@K indicates what percentage of m's actual component APIs can emerge in $\mathcal{A}^m$, and $F1$@K is a harmonic average, used to solve the problem of precision and recall

[1] https://github.com/HIT-ICES/Correted-ProgrammableWeb-dataset

imbalance. The $MAP@K$ is defined as:

$$MAP@K = \frac{1}{|\mathcal{M}|} \sum_{m \in \mathcal{M}} \frac{\sum_{j=1}^{K} prec_j^m \cdot rel_j^m}{\min(|\overline{\mathcal{O}^m}|, K)}, \quad (23)$$

where $prec_j^m$ is the precision at cut-off j in $\mathcal{A}^m$, rel_j^m is an indicator equaling 1 if the API at rank j is used by m, zero otherwise. $NDCG@K$ is defined as:

$$NDCG@K = \frac{1}{|\mathcal{M}|} \sum_{m \in \mathcal{M}} \left(\sum_{j=1}^{K} \frac{rel_j^m}{\log_2^{j+1}} \Big/ \sum_{j=1}^{K} \frac{1}{\log_2^{j+1}} \right), \quad (24)$$

which is similar to the ID-based reward function—Eq.(11), however averaged by the number of mashups in the test set.

3) BASELINES

To evaluate the performance of the proposed model, we compare it with four state-of-the-art single-round service recommendation baselines and two representative next-round service recommendation models, i.e., [7] and [5].

- **NAFM** [1] is a hybrid factorization machine model which integrates a deep neural network to capture the non-linear and complex feature interactions for service recommendation.
- **coACN** [42] is a deep learning-based service recommendation framework which can effectively learn the bilateral information toward service recommendation.
- **PAREI** [43] is a hybrid service model that optimizes recommendation results by combining explicit and implicit information.
- **SRMG** [44] is a graph-based service recommendation approach that makes recommendations based on service characteristics and historical usage.
- **iSRec** [7] is a service recommendation approach which integrates diverse information of mashups and their component APIs and can be adapt to make single-round and next round service recommendation.
- **HISR** [5] is a service recommendation framework which aims to make next round recommendation by capturing the interactions among the target mashup, selected services, and the next service to recommend.

4) EXPERIMENT SETUP

For RLISR, we use 300-dim glove pre-trained word vectors to learn the content embeddings of APIs. We then use the LDA model with topic number set to 100 to extract the topics of descriptions for both mashups and APIs, and build the SRKG. We set the embedding sizes of SRKG representation and final state representation to 512. We adopt 2-layer MLP for each Q-network, where the embedding size of its hidder layer is 1024 and its final softmax layer is 1322. We also test the graph convolutional embedding layer K in the range of 1 to 5, and finally choose 3. To learn the whole RL model, we employ the Adam optimizer, and use the default learning rate as 0.001, the discount factor to 0.9 and the greedy probability to 0.1. We investigate the top-K recommendation performance with $K = \{5, 10, 15, 20, 25\}$. We implement RLISR based on Tensorflow, a widely used open source machine learning framework.

TABLE 3. Overall performance comparison. The best results are starred, and the second-best results are listed in bold.

Top-K	Method	Precision	Recall	F1	NDCG	MAP
Top-5	NAFM	0.1614	0.2460	0.1835	0.4561	0.1919
	coACN	0.1687	0.2603	0.1930	0.4560	0.1925
	SRMG	0.1487	0.2319	0.1713	0.3536	0.1637
	PAREI	0.1553	0.2554	0.1835	0.4055	0.1688
	iSRec	**0.2589**	0.4127	**0.3005**	0.5621	**0.2960**
	HISR	0.2520	**0.4225**	0.2993	**0.6607**	0.2672
	RLISR	**0.2899***	**0.4320***	**0.3294***	**0.7780***	**0.3479***
Top-10	NAFM	0.0960	0.2855	0.1359	0.4671	0.1918
	coACN	0.1170	0.3512	0.1666	0.4784	0.2063
	SRMG	0.1164	0.3501	0.1665	0.3946	0.1858
	PAREI	0.1211	0.3591	0.1729	0.4451	0.1883
	iSRec	0.1380	0.4315	0.1986	0.5587	**0.2894**
	HISR	**0.1547**	**0.4557**	**0.2217**	**0.6592**	0.2681
	RLISR	**0.1990***	**0.4808***	**0.2625***	**0.7764***	**0.3513***
Top-15	NAFM	0.0699	0.3075	0.1083	0.4705	0.1941
	coACN	0.0912	0.4144	0.1432	0.4848	0.2156
	SRMG	0.1023	0.4247	0.1581	0.4052	0.1996
	PAREI	0.1120	0.4022	0.1675	0.4548	0.1952
	iSRec	0.0940	0.4358	0.1476	0.5584	**0.2896**
	HISR	**0.1199**	**0.4725**	**0.1831**	**0.6574**	0.2701
	RLISR	**0.1600***	**0.4832***	**0.2186***	**0.7759***	**0.3505***
Top-20	NAFM	0.0574	0.3343	0.0937	0.4711	0.1967
	coACN	0.0749	0.4512	0.1236	0.4850	0.2196
	SRMG	0.0963	0.4572	0.1529	0.4102	0.2038
	PAREI	**0.1088**	0.4132	**0.1647**	0.4570	0.1968
	iSRec	0.0713	0.4374	0.1176	0.5584	**0.2899**
	HISR	0.1020	**0.4835**	0.1603	**0.6561**	0.2713
	RLISR	**0.1429***	**0.4974***	**0.1973***	**0.7729***	**0.3522***
Top-25	NAFM	0.0494	0.3569	0.0832	0.4716	0.1986
	coACN	0.0650	**0.4892**	0.1110	0.4816	0.2232
	SRMG	0.0940	0.4648	0.1502	0.4103	0.2047
	PAREI	**0.1081**	0.4154	**0.1640**	0.4571	0.1971
	iSRec	0.0576	0.4383	0.0978	0.5583	**0.2901**
	HISR	0.0914	0.4889	0.1454	**0.6547**	0.2718
	RLISR	**0.1316***	**0.5011***	**0.1815***	**0.7704***	**0.3525***

B. OVERALL PERFORMANCE COMPARISON AND ANALYSIS

We compare the performance of the proposed RLISR approach against the six aforementioned methods. As RLISR, iSRec and HISR make service recommendation in a multi-round manner, we provide their results for top-K evaluation in the condition of K-round top-1 recommendation. Table 3 summarizes the experimental results of different algorithms with different number K of recommended APIs.

We have the following observations from Table 3 that: (1) RLISR consistently yields the best performance on all the metrics and K values. In detail, RLISR improves much more over the strongest baselines on the metrics NDCG and MAP than the other 3 metrics. (2) HISR achieves better performance than the other baselines in most cases, and sometimes, iSRec achieves the best results against the other baselines. As mentioned above, HISR and iSRec are the two representative next round service recommendation models. Thus, from the above 2 observations, we can conclude that introducing multi-round recommendation mode can boost the

effectiveness of service recommendation, and introducing RL into multi-round service recommendation can significantly improve the performance of service recommendation.

C. ABLATION STUDY

RLISR unifies several components that contribute to its effectiveness in IRS. In this section, we provide insights on KG-based mashup and API encoding component, GRU-based historical interaction encoding component, negative feedback incorporating component and RL-based interactive service recommendation component and how they are indispensable to ISR by performing an ablation study on various design choices of RLISR. We designed the following variants of RLISR for comparison:

- **RLISR-Cont:** a variant of RLISR, which uses only pre-trained content vectors of mashups and APIs during the state representation phase.
- **RLISR-KG:** a variant of RLISR, which uses SRKG-based vectors of mashups and APIs (i.e., the API representation isn't enhanced by content pre-training) during the state representation phase.
- **RLISR-Avg:** a variant of RLISR, which uses the global mean layer instead of the GRU for historical interaction encoding to obtain the state representation.
- **RLISR-Pos:** a variant of RLISR, which does not take the unselected APIs (i.e., negative feedbacks) into account.
- **RLISR-Sim:** a variant of RLISR, which uses cosine similarity instead of an reinforcement learning model to make service recommendation.

TABLE 4. Performance comparison of different variants of RLISR.

variants	Precision	Recall	F1	NDCG	MAP
RLISR-Cont	0.0593	0.3422	0.0961	0.4549	0.1737
RLISR-KG	0.0740	0.4569	0.1226	0.6224	0.2617
RLISR-Avg	0.0718	0.4496	0.1190	0.6195	0.2573
RLISR-Pos	0.0772	0.4658	0.1275	0.6210	0.2642
RLISR-Sim	0.0560	0.3317	0.0915	0.4003	0.1500
RLISR	0.0782	0.5030	0.1303	0.6477	0.2657

We summarize the experimental results in Table 4 and have the following findings:

- Comparing with content-based mashup and API encoding (i.e., RLISR-Cont), using SRKG (i.e., RLISR-KG) can achieve significant performance improvement. Moreover, if API representation is further enhanced by content pre-training (i.e., RLISR), our model can still achieve slight improvement.
- Replacing GRU model with Average Pooling operation for historical interaction encoding degrades the model's performance. Thus, considering the sequential nature of feedbacks is necessary.
- The viriant RLISR-Pos underperform RLISR slightly, suggesting that considering the negative feedbacks can improve the recommendation performance of our model.

FIGURE 5. Influence of topic number and embedding size on the RLISR effect.

- Removing the RL-based recommendation component (i.e., RLISR-Sim) degrades the model's performance significantly. It verifies the effectiveness of the adopted RL model for ISR.

D. HYPERPARAMETER ANALYSIS

As analyzed in ablation study, SRKG plays an critical role for the performance of our model. The number of topics in SRKG and the embedding size of SRKG representation are two core hyperparameters for our model. To get deep insights on RLISR, we explored its sensitivity on these 2 hyperparameters. Fig. 5 shows the recommendation performance under different combinations of the 2 hyperparameters.

We set the topic number $T = \{80, 90, 100, 110, 120\}$ and the dimensionality $dim = \{128, 256, 512\}$ in turn. As shown in Fig 5, it can be found that (1) under the same embedding size, the recommendation effect does not improve obviously with the increase of the topic number; (2) under the same topic number, the performance improvement does not show regularity with the increase of dimensionality; (3) our model generally achieves the best performance when the topic number is 100 and the embedding size is set to 512. Thus, we keep these 2 values in other experiments.

TABLE 5. Performance comparison of different ISR mode. The best results are listed in bold.

Top-K	Round	Precision	Recall	F1	NDCG	MAP
Top-5	5	**0.2899**	**0.4320**	**0.3294**	**0.7780**	**0.3479**
	1	0.1552	0.2512	0.1809	0.4356	0.1756
Top-10	10	**0.1990**	**0.4808**	**0.2625**	**0.7764**	**0.3513**
	5	0.1500	0.3938	0.2040	0.7001	0.3038
	2	0.1227	0.3860	0.1779	0.6437	0.2508
	1	0.1072	0.3404	0.1549	0.4724	0.1864
Top-15	15	**0.1600**	**0.4832**	**0.2186**	**0.7759**	**0.3505**
	5	0.1255	0.4441	0.1844	0.7133	0.3194
	3	0.1009	0.4435	0.1580	0.6502	0.2591
	1	0.0836	0.3960	0.1323	0.4801	0.1932
Top-20	20	**0.1429**	**0.4974**	**0.1973**	**0.7729**	**0.3522**
	5	0.1097	0.4839	0.1694	0.7120	0.3251
	4	0.0879	0.4815	0.1433	0.6514	0.2637
	1	0.0709	0.4464	0.1179	0.4895	0.1982
Top-25	25	**0.1316**	**0.5011**	**0.1815**	**0.7704**	**0.3525**
	5	0.0782	0.5030	0.1303	0.6477	0.2657
	1	0.0599	0.4687	0.1028	0.4915	0.2001

E. ISR MODE ANALYSIS

Our model recommends APIs for a target mashup in a multi-round interactive manner. So, to recommend a given number of APIs in a whole recommendation process, our model can work in different mode. Taking the top-10 recommendation as an example, RLISR can make 10-round-top-1, 5-round-top-2, 2-round-top-5 or 1-round-top-10 recommendation. We further investigate the impact of round numbers with constraint of the same number of total recommended APIs. The results are listed in the Table 5.

We study the performance of top-K recommendation with different K values {5, 10, 15, 20, 25}. From Table 5, we can see that (1) no matter which value K is, our model achieves the best performance when it recommends one API each round and makes K round recommendation; (2) the performance of our model improves gradually along with the increase of the round number on whatever K values and metrics. It verifies that RLISR can capture and exploit developer's feedbacks effectively and increase recommendation efficiency significantly for mashup developers.

VI. CONCLUSION

In this paper, we introduce a new service recommendation setting—multi-round interactive service recommendation, which can support mashup developers to express her/his dynamic requirements and aims at obtaining the optimal long-term recommendation performance over the whole rounds and improving the efficiency of service selection. To fit the new setting, we propose a deep RL-based interactive service recommendation model—RLISR. It constructs a KG and utilizes both positive and negtive feedbacks to enhance state representation; it is guided by an informative reward function to improve recommendation accuracy and reduce number of rounds; and its optimal policy is learned by adopting a cascading Q networks model from enormous action space. Extensive experiments on a real-world dataset demonstrate that our method gains a significant improvement compared with the state-of-the-art baselines.

In the future, we plan to incorporate more information, such as mashup developers, API providers and social connection between APIs into RLISR to promote the accuracy of recommendations. We also plan to investigate more reinforcement learning models such as DDPG to further boost the performance.

REFERENCES

[1] 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.

[2] M. Boulakbech, N. Messai, Y. Sam, and T. Devogele, "Deep learning model for personalized web service recommendations using attention mechanism," in *Proc. ICSOC* (Lecture Notes in Computer Science), vol. 14419. Cham, Switzerland: Springer, 2023, pp. 19–33.

[3] 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./Aug. 2023.

[4] 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.

[5] Y. Ma, X. Geng, J. Wang, K. He, and D. Athanasopoulos, "Deep learning framework for multi-round service bundle recommendation in iterative mashup development," *CAAI Trans. Intell. Technol.*, vol. 8, no. 3, pp. 914–930, Sep. 2023.

[6] Y. Xiao, J. Liu, R. Hu, B. Cao, and Y. Cao, "DINRec: Deep interest network based API recommendation approach for mashup creation," in *Proc. WISE* (Lecture Notes in Computer Science), vol. 11881. Cham, Switzerland: Springer, 2019, pp. 179–193.

[7] 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, Jun. 2019.

[8] C. Fellbaum, "WordNet," in *Theory and Applications of Ontology: Computer Applications*. Berlin, Germany: Springer, 2010, pp. 231–243.

[9] C. Platzer and S. Dustdar, "A vector space search engine for web services," in *Proc. 3rd Eur. Conf. Web Services*, Nov. 2005, p. 9.

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

[11] 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.

[12] M. Shi 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.

[13] X. Wu, B. Cheng, and J. Chen, "Collaborative filtering service recommendation based on a novel similarity computation method," *IEEE Trans. Services Comput.*, vol. 10, no. 3, pp. 352–365, May 2017.

[14] G. Zou, M. Jiang, S. Niu, H. Wu, S. Pang, and Y. Gan, "QoS-aware web service recommendation with reinforced collaborative filtering," in *Proc. ICSOC* (Lecture Notes in Computer Science), vol. 11236. Cham, Switzerland: Springer, 2018, pp. 430–445.

[15] 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.

[16] L. Yao, Q. Z. Sheng, Anne. H. H. Ngu, J. Yu, and A. Segev, "Unified collaborative and content-based web service recommendation," *IEEE Trans. Services Comput.*, vol. 8, no. 3, pp. 453–466, May 2015.

[17] 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.

[18] K. A. Botangen, J. Yu, S. Yongchareon, L. H. Yang, and Q. Z. Sheng, "Integrating geographical and functional relevance to implicit data for web service recommendation," in *Proc. ICSOC* (Lecture Notes in Computer Science), vol. 11895. Cham, Switzerland: Springer, 2019, pp. 53–57.

[19] M. Nguyen, J. Yu, Q. Bai, S. Yongchareon, and Y. Han, "Attentional matrix factorization with document-context awareness and implicit API relationship for service recommendation," in *Proc. ACSW*, 2020, p. 17.
[20] T. Silva, N. Silva, H. Werneck, C. Mito, A. C. M. Pereira, and L. Rocha, "iRec: An interactive recommendation framework," in *Proc. SIGIR*, 2022, pp. 3165–3175.
[21] Y. Liu, Y. Xiao, Q. Wu, C. Miao, J. Zhang, B. Zhao, and H. Tang, "Diversified interactive recommendation with implicit feedback," in *Proc. AAAI*, 2020, pp. 4932–4939.
[22] S. M. T. Zaidi, P. Chadalavada, H. Ullah, A. Munir, and A. Dutta, "Cascaded deep reinforcement learning-based multi-revolution low-thrust spacecraft orbit-transfer," *IEEE Access*, vol. 11, pp. 82894–82911, 2023.
[23] Y. Zhang, R. Li, Y. Zhao, R. Li, Y. Wang, and Z. Zhou, "Multi-agent deep reinforcement learning for online request scheduling in edge cooperation networks," *Future Gener. Comput. Syst.*, vol. 141, pp. 258–268, Apr. 2023.
[24] X. Chen, S. Li, H. Li, S. Jiang, Y. Qi, and L. Song, "Generative adversarial user model for reinforcement learning based recommendation system," in *Proc. ICML*, vol. 97, 2019, pp. 1052–1061.
[25] H. Chen, C. Zhu, R. Tang, W. Zhang, X. He, and Y. Yu, "Large-scale interactive recommendation with tree-structured reinforcement learning," *IEEE Trans. Knowl. Data Eng.*, vol. 35, no. 4, pp. 4018–4032, Apr. 2023.
[26] S. Zhou, X. Dai, H. Chen, W. Zhang, K. Ren, R. Tang, X. He, and Y. Yu, "Interactive recommender system via knowledge graph-enhanced reinforcement learning," in *Proc. SIGIR*, 2020, pp. 179–188.
[27] J. Wu, Z. Xie, T. Yu, H. Zhao, R. Zhang, and S. Li, "Dynamics-aware adaptation for reinforcement learning based cross-domain interactive recommendation," in *Proc. SIGIR*, 2022, pp. 290–300.
[28] Y. Lei and W. Li, "Interactive recommendation with user-specific deep reinforcement learning," *ACM Trans. Knowl. Discovery Data*, vol. 13, no. 6, p. 61, 2019.
[29] L. Zou, L. Xia, P. Du, Z. Zhang, T. Bai, W. Liu, J. Nie, and D. Yin, "Pseudo dyna-Q: A reinforcement learning framework for interactive recommendation," in *Proc. WSDM*, 2020, pp. 816–824.
[30] L. Ren and W. Wang, "A granular SVM-based method for top-N web services recommendation," *IEEE Trans. Services Comput.*, vol. 15, no. 1, pp. 457–469, Jan. 2022.
[31] D. Zhao, Z. Zhou, W. Zhang, S. Deng, X. Xue, and W. Gaaloul, "CSTL: Compositional signal temporal logic for adaptive edge service monitoring," *IEEE Trans. Services Comput.*, vol. 17, no. 2, pp. 482–496, Mar./Apr. 2024.
[32] D. Zhao, Z. Zhou, Z. Cai, S. Yangui, and X. Xue, "ASTL: Accumulative STL with a novel robustness metric for IoT service monitoring," *IEEE Trans. Mobile Comput.*, vol. 22, no. 10, pp. 5751–5768, Oct. 2023.
[33] 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.
[34] 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.
[35] T. N. Kipf and M. Welling, "Semi-supervised classification with graph convolutional networks," in *Proc. ICLR*, 2017, pp. 1–14.
[36] X. Wang, X. He, Y. Cao, M. Liu, and T. Chua, "KGAT: Knowledge graph attention network for recommendation," in *Proc. SIGKDD*, 2019, pp. 950–958.
[37] Q. V. Le and T. Mikolov, "Distributed representations of sentences and documents," in *Proc. ICML*, vol. 32, 2014, pp. 1188–1196.
[38] K. Cho, B. van Merrienboer, Ç. Gülçehre, D. Bahdanau, F. Bougares, H. Schwenk, and Y. Bengio, "Learning phrase representations using RNN encoder–decoder for statistical machine translation," in *Proc. EMNLP*, 2014, pp. 1724–1734.
[39] H. van Hasselt, A. Guez, and D. Silver, "Deep reinforcement learning with double Q-learning," in *Proc. AAAI*, 2016, pp. 2094–2100.
[40] M. Liu, Z. Tu, Y. Zhu, X. Xu, Z. Wang, and Q. Z. Sheng, "Data correction and evolution analysis of the ProgrammableWeb service ecosystem," *J. Syst. Softw.*, vol. 182, Dec. 2021, Art. no. 111066.
[41] B. Bai, Y. Fan, K. Huang, W. Tan, B. Xia, and S. Chen, "Service recommendation for mashup creation based on time-aware collaborative domain regression," in *Proc. IEEE Int. Conf. Web Services*, Jun. 2015, pp. 209–216.
[42] R. Yan, Y. Fan, J. Zhang, J. Zhang, and H. Lin, "Service recommendation for composition creation based on collaborative attention convolutional network," in *Proc. IEEE Int. Conf. Web Services (ICWS)*, Sep. 2021, pp. 397–405.
[43] 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.
[44] T. Yu, D. Yu, D. Wang, and X. Hu, "Web service recommendation for mashup creation based on graph network," *J. Supercomput.*, vol. 79, no. 8, pp. 8993–9020, May 2023.

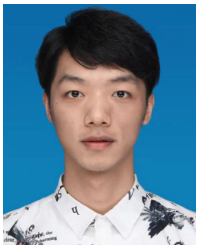
MINGWEI ZHANG was born in Tsingtao, Shandong, China, in 1979. He received the B.E., M.E., and Ph.D. degrees in computing science and technology from Northeastern University, Shenyang, China, in 2002, 2005, and 2011, respectively.

From 2019 to 2020, he was a Visiting Scholar with the Royal Melbourne Institute of Technology, Australia. He has been a Teacher, since 2005, and an Associate Professor with the Software College, Northeastern University, since 2016. His research interests include recommender systems, graph representation, and service computing.

YINGJIE QU was born in Tangshan, Hebei, China, in 1998. She received the bachelor's degree in software engineering from Northeast Normal University, Jilin, China, in 2021. She is currently pursuing the master's degree in software engineering with the Software College, Northeastern University, Shenyang, China. Her research interests include service computing and recommender systems.

YAGE LI was born in Changde, Hunan, China, in 1998. He received the bachelor's degree in metallurgical engineering from Northeastern University, Shenyang, China, in 2021, where he is currently pursuing the master's degree in software engineering with the Software College. His research interests include reinforcement learning and recommender systems.

XINGYU WEN was born in Liaoyang, Liaoning, China, in 1998. He received the bachelor's degree in software engineering from Northeastern University, Shenyang, China, in 2021, where he is currently pursuing the master's degree in software engineering with the Software College. His research interests include deep learning and itinerary recommendation.



YI ZHOU was born in Bengbu, Anhui, China, in 2000. He received the bachelor's degree in IoT engineering from Anhui University, Hefei, China, in 2021. He is currently pursuing the master's degree in software engineering with the Software College, Northeastern University, Shenyang, China. His research interests include service computing and recommender systems.