IEEE Access
Multidisciplinary ⋮ Rapid Review ⋮ Open Access Journal

Received 30 October 2024, accepted 16 November 2024, date of publication 25 November 2024, date of current version 23 January 2025.

Digital Object Identifier 10.1109/ACCESS.2024.3505943

RESEARCH ARTICLE

A Cloud API Personalized Recommendation Method Based on Multiple Attribute Features and Mashup Requirement Attention

LIMIN SHEN, YUYING WANG, CHENGYU LI, AND ZHEN CHEN
College of Information Science and Engineering, Yanshan University, Qinhuangdao 066004, China
Hebei Key Laboratory of Software Engineering, Qinhuangdao 066004, China

Corresponding author: Yuying Wang (yuyingwang@stumail.ysu.edu.cn)

This work was supported in part by the National Natural Science Foundation of China under Grant 61772450 and Grant 62102348, in part by the Natural Science Foundation of Hebei Province under Grant F2022203012, in part by the Science and Technology Research Project of Hebei University under Grant QN2020183, and in part by the Project of Hebei Key Laboratory of Software Engineering under Grant 22567637H.

ABSTRACT In current mashup-oriented cloud API recommendation systems, insufficient attention to personalized development requirements remains a common issue, particularly regarding developers' needs for attributes such as functionality similarity and complementarity. This paper proposes a novel approach for personalized cloud API feature representation and recommendation. We construct a graph of the cloud API ecosystem with rich side information and design metapaths to capture and characterize various API features. To fully leverage information from intermediate nodes in the metapaths and emphasize the significance of different instances, we employ a translational distance model and graph neural network techniques to aggregate cloud API feature information. Furthermore, we introduce mashup requirement attention, a mechanism that customizes recommendations based on the specific needs of each mashup project, thereby enhancing the accuracy and personalization of API recommendations. Extensive experiments on real-world datasets demonstrate the effectiveness of the proposed method.

INDEX TERMS Attention mechanism, cloud application programming interface, mashup-oriented, multiple attribute features, personalized recommendation.

I. INTRODUCTION

With the rapid advancement of cloud computing technology, there has been an exponential growth in the number of cloud Application Programming Interfaces (APIs), making them essential for service delivery, function replication, and data export [1]. Lightweight composite services, known as mashups, have gained significant attention due to their flexibility and uniqueness. As web-based data integration applications, mashups create value-added services by combining content from various data sources. This technology enables developers to leverage existing cloud APIs for the rapid development of innovative and practical applications that can swiftly respond to market dynamics and user needs.

The associate editor coordinating the review of this manuscript and approving it for publication was Jolanta Mizera-Pietraszko.

However, the expanding size of API repositories poses a significant challenge: effectively identifying and selecting APIs that align with specific development requirements has emerged as a critical concern in cloud computing.

To address this challenge, recommendation algorithms assist in selecting cloud APIs. The intelligent recommendation of cloud APIs not only improves development efficiency and shortens project cycle time, but also fosters the robust growth and widespread adoption of the cloud API economy. Currently, mashup-oriented cloud API recommendations are primarily based on collaborative filtering (CF) or content. Collaborative filtering (CF) methods, which typically utilize user-API or mashup-API interaction matrices, identify developer preferences through similarity analysis for personalized recommendations [2]. These methods, however, rely heavily on sufficient historical data, and their performance can be

hindered by high-dimensional and sparse matrices, leading to cold-start issues and limited service representation [3], [4] [5]. Content-based approaches, on the other hand, analyze API description documents to match user requirements by extracting semantic features and calculating their similarity [6]. While these approaches avoid the cold-start problem, they often underutilize additional information from the service ecosystem, resulting in poor quality of the obtained semantic features [7], [8].

To overcome the limitations inherent in both CF-based and content-based methods, hybrid approaches have been developed that combine the strengths of both methodologies. These approaches utilize machine learning and deep learning techniques to extract semantic information while also leveraging matrix decomposition and clustering techniques to handle interaction information between mashups and APIs [9], [10], [11]. The task of recommending cloud APIs for mashups involves unique challenges, including the need to understand the complex inter dependencies between APIs and to adapt to the various functionalities required by the mashup. Simple heuristic-based approaches, while effective in straightforward recommendation scenarios, are unable to capture the subtle relationships and rich contextual information required for accurate API recommendations in this domain. Therefore, state-of-the-art techniques such as graph neural networks are employed to address these challenges by leveraging structural information embedded in the API knowledge graph and capturing multi-hop interactions between APIs.

Thus, to enhance the accuracy of API recommendations, increasing work has utilized the extensive knowledge accumulated in the cloud API ecosystem [12]. As shown in Fig. 1, data in the cloud API ecosystem essentially has a graph structure, and knowledge graphs have been employed to introduce side-information into cloud API recommendations, alleviating the issues of data sparsity and result homogeneity [13].

Despite the advancements of knowledge graph-based recommendation methods in mashup-oriented cloud API recommendation, several deficiencies remain. In the API ecosystem, cloud APIs are characterized by a variety of attributes, such as functionality similarity-how closely APIs perform similar tasks-and functionality complementarity-how APIs can work together to enhance overall capabilities. These attribute characteristics are crucial in cloud API recommendation. Specifically, complex mashups require not only cloud APIs with similar functionality for specific functionality enhancement but also APIs with different functionality to complement each other in fulfilling more complex requirements [14]. The similarity among cloud APIs can be characterized by the tagging relationship of functional tags, while complementarity is implicit in the invocation relationships between APIs. These various attribute features will not fully and accurately reflect the diverse characteristics of cloud APIs if learned simultaneously without differentiation.

Moreover, mashups exhibit varying preferences for different types of API attributes based on their requirements. Some mashups tend to integrate multiple cloud APIs with similar functionalities to achieve specific feature enhancements. For instance, in a mapping mashup, APIs such as Google Maps, Google Earth, and Bing Maps may need to be integrated to enhance the richness and accuracy of geographic information services. In this process, the functional similarity between APIs becomes a crucial factor to consider during integration. Conversely, some mashups require cross-category cloud APIs to achieve functional complementarity. For example, a travel service mashup might need to invoke various APIs such as photo, map, and social APIs to provide comprehensive and personalized services. In this scenario, the complementarity between different APIs is crucial for constructing an effective mashup. If various types of attribute features of cloud APIs, delivered through different relationships, are treated equally, some potential preference information of the mashup might be overlooked, subsequently affecting the accuracy and comprehensiveness of the recommendation system.

Confronting the aforementioned problem, this study proposes a personalized cloud API recommendation method that incorporates multiple attribute features and considers attention related to mashup requirements. First, the method constructs a cloud API ecosystem graph, including cloud APIs, mashups, and their functional tags, and designs metapaths to capture information implied by different relationships between nodes. To fully consider the internal information of the nodes in the metapaths and the importance of different metapath instances, the similarity and complementary features of cloud APIs based on different metapaths are encoded and aggregated using the translational distance model and graph neural network techniques. In addition, an attention mechanism associated with mashups is introduced, taking into account the potential preferences of different mashups for similarity and complementarity. This mechanism determines the importance of different attribute characteristics of the cloud API for the requirements of a given mashup, enabling the recommender system to customize the recommendations according to the unique requirements of different mashups. The different attribute features are weighted and fused using these attention weights to obtain a personalized feature representation of the cloud API that is highly relevant to the requirements. Then, based on the mashup's requirements and the corresponding personalized cloud API features, the probability of each candidate cloud API being invoked by a given mashup is predicted, and the cloud API with the highest probability is selected to generate a recommendation list. This approach aims to improve the effectiveness of personalized cloud API recommendations for mashups, providing developers with more accurate and personalized API recommendation services.

The key contributions of this paper are as follows:

(1) We propose a novel method that encodes and aggregates similarity and complementary features of cloud APIs using

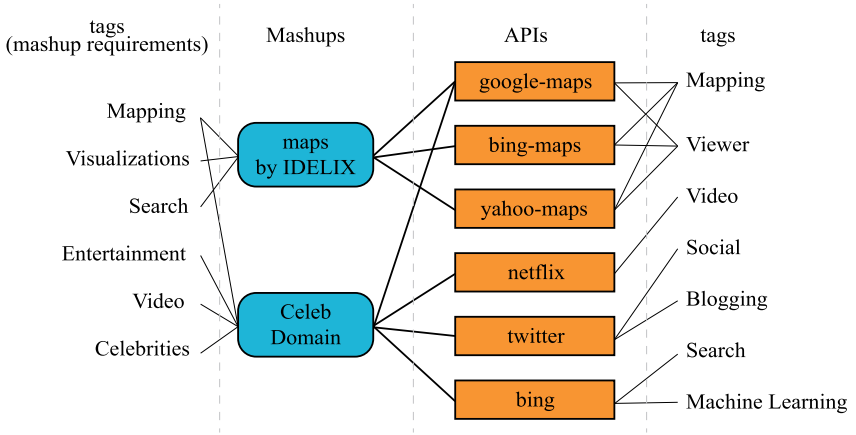


FIGURE 1. A toy example of the cloud API ecosystem.

a combination of translational distance models and graph neural networks, allowing for a more nuanced and accurate representation of API attributes.

(2) We introduce a unique attention mechanism that dynamically adjusts the importance of different API features based on the specific requirements of a mashup, enhancing the precision and personalization of API recommendations.

(3) We validate our approach using real-world datasets, showing superior performance in mashup-oriented cloud API recommendations. By incorporating developers' preferences for different functional attributes, our method yields more accurate API recommendations aligned with specific development requirements.

The remainder of the paper is structured as follows: Section II summarizes the related work. Section III introduces the preliminary knowledge. Section IV details the specific methodology used. Section V presents the experimental setup and results. Section VI concludes the paper by summarizing the study results and outlining future work.

II. RELATED WORK

Mashup has gained widespread adoption in software development due to its innovative programming paradigm. By integrating diverse cloud APIs, this technology significantly accelerates the development of innovative applications. In recent years, researchers have focused on developing efficient API recommendation mechanisms to optimize the mashup development process and enhance application performance. This section presents an overview of representative research in this area.

A. CF-BASED METHODS

Collaborative filtering methods mainly rely on user-API or mashup-API interaction matrices to identify developer preferences through similarity analysis for personalized recommendations.

Xue et al. [3] proposed a deep matrix decomposition framework that extracts features directly from users' interaction data with APIs and integrates explicit and implicit ratings to achieve Top-K recommendations. Fletcher [15] proposed a regularized user preference-embedded matrix decomposition method that considers both explicit and implicit personalized preferences, significantly improving the accuracy and diversity of the recommender system. Lian and Tang [4] proposed a technique utilizing neural graph collaborative filtering and higher-order connections between cloud APIs and users. The importance of similarity computation in collaborative filtering algorithms has been widely recognized, and some studies have enhanced the accuracy of similarity computation by incorporating additional information. For example, Botangen et al. [16] integrated geographic and functional relevance into a probabilistic matrix decomposition model to infer potential mashup invocation preferences. Meng et al. [17] considered temporal effects to distinguish between stable Quality of Service (QoS) measurements and temporal QoS metrics in a collaborative filtering model based on traditional neighbor relationships. Wang et al. [5] considered service popularity in matrix decomposition by weighting all elements to avoid recommending popular but useless API services.

These methods train effective recommendation models with a large amount of data, improving recommendation effect and prediction efficiency, but there are some issues: matrix decomposition models require sufficient historical records, while mashup-API matrices are usually high-dimensional and sparse, leading to similar feature vectors and affecting service representation. In addition, new services lack interaction records and face cold-start problems.

B. CONTENT-BASED METHODS

Content-based approaches analyze API description documents by parsing user queries, extracting semantics at different levels, and calculating their similarity to user requirements. The accuracy of matching mainly relies on effective feature extraction from documents, and the effectiveness of data feature mining directly affects the result's ability to meet user requirements.

Gong et al. [7] used functional keywords to index APIs, constructed weighted undirected graphs to represent services, transformed the matching of user requirements with keywords into a minimum Steiner tree problem, and utilized the minimum spanning tree algorithm to recommend service combinations. Chen et al. [18] and Wu et al. [19] further considered service compatibility and popularity, applying deep reinforcement learning and dynamic planning techniques to find the minimum Steiner tree. Topic modeling represents the semantic functions of services more accurately by modeling document vocabularies. Zhong et al. [8] proposed a novel cloud API recommendation method that models mashup descriptions and cloud API components using an author-topic model, exploiting vocabulary features of the reconstructed cloud API profiles. Zhang et al. [20] proposed a new cloud API recommendation method that complements mashup functional descriptions of the called API service description document, identifying and filtering terms that are not service-distinctive to improve recommendation accuracy. Zhang et al. [21] proposed a word embedding-based service representation method using Gaussian mixture model and Word2vec to obtain service representation vectors suitable for subsequent tasks like clustering and recommendation. Bai et al. [22] applied stacked denoising auto-encoders to use noise to simulate uncertainty in user feature descriptions, reconstruct clean input information, eliminate misleading uncertain words, and improve result accuracy. Zhao et al. [23] proposed a cloud API recommendation method combining feature ensemble and learning to rank by integrating textual features, nearest-neighbor features, API-specific features, and tagged features from mashup and cloud APIs. Shi and Liu [24] proposed a cloud API recommendation method based on text extension and deep learning models, using a hierarchical probabilistic topic model to augment sentence-level cloud API descriptions and recommend cloud APIs based on semantic similarity.

While the content-based approach avoids the cold-start problem by mining the functional semantic information of the service, it focuses on the representation of feature vectors in the service description document. There is also a large amount of information in the service ecosystem that can enhance content representation, such as invocation relationships between services and multi-dimensional service attributes, which are underutilized in the above literature, resulting in poor quality of the obtained semantic features of services.

C. HYBRID METHODS

Given the performance limitations of a single recommendation model in mashup recommendation, researchers developed a hybrid recommendation approach that combines collaborative filtering with content-based recommendation. Machine learning and deep learning techniques are utilized to extract semantic information from service description documents, while clustering and matrix decomposition techniques are applied to identify similar services and learn interaction information between mashups and APIs. The hybrid approach incorporates the advantages of multiple models to improve recommendation accuracy.

Cao et al. [9] use the similarity, popularity, and co-occurrence of mashup and API as inputs, employing a factorization machine for feature interaction learning to alleviate the data sparsity problem. Zhang et al. [25] apply topic models and category labels to predict service synergies based on invocation records and recommend a prioritized list of services. Ma et al. [10] use a convolutional neural network for feature extraction, input vector representations of text and tags into a multilayer perceptron combined with the node2vec algorithm to learn low-dimensional representations of mashups and services, and finally obtain the service scores. Xiong et al. [26] combined nonlinear interactions between mashups and APIs learned by a multilayer perceptron with feature similarity extracted by word vectors, and input these results into a fully-connected layer to predict service scores and accomplish the combination recommendation task. Ma et al. [27] further integrated an attention mechanism to compute mashup similarity for interactive service combination recommendations.

Although these approaches go beyond the limitations of a single model, they still cannot fully satisfy the variety of services required by users when making recommendations in a comprehensive service library. Additionally, functionally redundant services may reduce the recommender system's accuracy.

III. PRELIMINARIES

A. CLOUD API ECOSYSTEM GRAPH CONSTRUCTION

On the cloud API publishing and sharing platform, a substantial amount of information about cloud APIs, mashups, and function tags is recorded, as well as information about the invocation and tagging relationships among them [28]. The cloud API ecosystem graph $\mathcal{G} = (\mathcal{V}, \mathcal{E})$ can be constructed using this data, where $\mathcal{V}$ and $\mathcal{E}$ denote the set of nodes and the set of edges, respectively. $\mathcal{V}$ consists of three types of nodes: the set of cloud API nodes $A = \{a_1, a_2, \cdots, a_{|A|}\}$, the set of mashup nodes $M = \{m_1, m_2, \ldots, m_{|M|}\}$, and the set of function tag nodes $T = \{t_1, t_2, \ldots, t_{|T|}\}$. $\mathcal{E}$ includes three types of relationships: the invocation relationship between mashups and cloud APIs, the tagging relationship between cloud APIs and function tags, and the tagging relationship between mashups and function tags [29].

As exemplified in Fig. 1, the mashup "CelebDomain" invokes four cloud APIs, including "google-maps", "netflix", "twitter", and "bing", and has the function tags "mapping", "entertainment", "video" and "celebrities"; the cloud API "Twitter" is tagged with the function tags "Social" and "Blogging."

To effectively analyze these complex relationships, the concept of metapath is introduced. To facilitate the

description of our method in the following sections, the definitions of the metapath and its instances are provided as follows.

Definition 1 (Metapath): A meta-path is defined as a sequence of node types and edge types that captures a specific schema-level path within a heterogeneous graph $\mathcal{G}$. Formally, a meta-path P is represented as $P : NT_1 \xrightarrow{ET_1} NT_2 \xrightarrow{ET_2} \ldots \xrightarrow{ET_l} NT_{l+1}$, where NT_i denotes a node type, and ET_i denotes an edge type. In the cloud API ecosystem graph, an example of a meta-path is P_1 : Mashup $\xrightarrow{\text{invokes}}$ API $\xrightarrow{\text{invoked by}}$ Mashup.

Definition 2 (Metapath Instance): A meta-path instance refers to a specific instantiation of a meta-path within the graph $\mathcal{G}$. It corresponds to a concrete sequence of nodes and edges that conforms to the structure defined by a given meta-path. For instance, given the meta-path P_1 : Mashup $\xrightarrow{\text{invokes}}$ API $\xrightarrow{\text{invoked by}}$ Mashup, a meta-path instance could be the sequence: "CelebDomain" $\xrightarrow{\text{invokes}}$ "Twitter" $\xrightarrow{\text{invoked by}}$ "SocialMediaAggregator".

By leveraging these definitions, the interactions between different entities in the cloud API ecosystem can be systematically explored. The relationships in the ecosystem can be effectively analyzed using metapaths and their instances, as defined above, to facilitate an understanding of the complex interactions in the cloud API ecosystem [30].

B. SIMILARITY AND COMPLEMENTARITY DESCRIPTION

Cloud APIs possess various attributes, including functionality, performance, and security. In developing mashups based on functionality proposed by developers, functional similarity and functional complementarity are crucial for the effective integration of diverse services and resources.

Functional similarity refers to the proximity of two or more cloud APIs in terms of functional attributes [24]. Functional tags of cloud APIs are key identifiers describing the capabilities of the provided services and precisely defining the core functional characteristics of each cloud API.When two or more cloud APIs share one or more functional tags, it indicates a degree of functional similarity among these APIs. In recommender systems, evaluating functional similarity between APIs aids developers in identifying and utilizing similar APIs when building mashup applications, thus maximizing the strengths and features of each API and enhancing the system's functionality, reliability, and user experience. For example, Google Maps, Bing Maps, and Yahoo Maps all offer mapping services and geographic information features. Combining these APIs can provide users with more comprehensive and accurate mapping services, enabling developers to seamlessly integrate them into mashup applications for various purposes such as travel planning, logistics management, and local services.

Functional complementarity refers to the capability of two or more cloud APIs to complement each other functionally to jointly satisfy specific requirements [14]. The simultaneous invocation of cloud APIs during mashup creation reveals their collaborative potential, indicating functional complementarity. In recommender systems, assessing functional complementarity between APIs helps identify combinations that can work together to fulfill complex requirements, thus offering more comprehensive and efficient recommendation services. For example, in e-commerce, a complete shopping experience may involve various segments like product search, price comparison, payment, and logistics. To provide such an experience, developers may need to integrate multiple APIs with complementary features, such as a product search API (e.g., Google Shopping API), a price comparison API (e.g., PriceGrabber API), a payment API (e.g., Stripe API), and a logistics API (e.g., UPS API). These APIs collaborate to provide users with a complete shopping experience, from product search to the final receipt of goods.

C. PROBLEM FORMALIZATION

In the cloud API ecosystem, the invocation relationships between mashups and cloud APIs can be defined as a matrix $Y \in {}^{|M|\times|A|}$, where each element $y_{m,a}$ is represented in binary form. 1 indicates that mashup m invokes API a, while 0 indicates no invocation relationship. The ultimate goal of cloud API recommendation for mashups is to predict the probability of a given mashup having potential interest in candidate cloud APIs. This prediction uses the mashup-API interaction matrix $Y \in {}^{|M|\times|A|}$ and the cloud API ecosystem graph $\mathcal{G} = (\mathcal{V}, \mathcal{E})$, based on specific requirements. The predictive function is represented as follows:

$$\hat{y}_{m,a} = F(m, a|Y, \mathcal{G}, \text{Treq}_m, \Theta), \tag{1}$$

where $\hat{y}_{m,a}$ represents the probability of a mashup invoking a cloud API, $\text{Treq}_m = \{\text{treq}_1, \text{treq}_2, \cdots, \text{treq}_n\} \in T$ represents the set of functional tags required by the mashup, and Θ denotes the model's parameters.

IV. METHODOLOGY

To identify the most suitable cloud API for constructing a mashup that meets a given requirement, it is essential to accurately, comprehensively, and purposefully characterize the feature representation of the cloud API. Fig. 2 illustrates the overall framework of the proposed mashup-orient cloud API recommendation method.The method mainly consists of the following steps:

(1) Node representation and path instances extraction: cloud API and mashup requirement features are embedded into dense vectors. Based on the cloud API ecosystem graph, metapaths are defined and path instances extracted

(2) Path instances encoding and aggregation: path instances are encoded into feature vectors with a translational distance model and aggregated using a graph attention neural network to represent cloud API different features.

(3) Attention-based Fusion: a mashup requirement-related attention fuse similarity and complementary features to generate personalized cloud API representations.

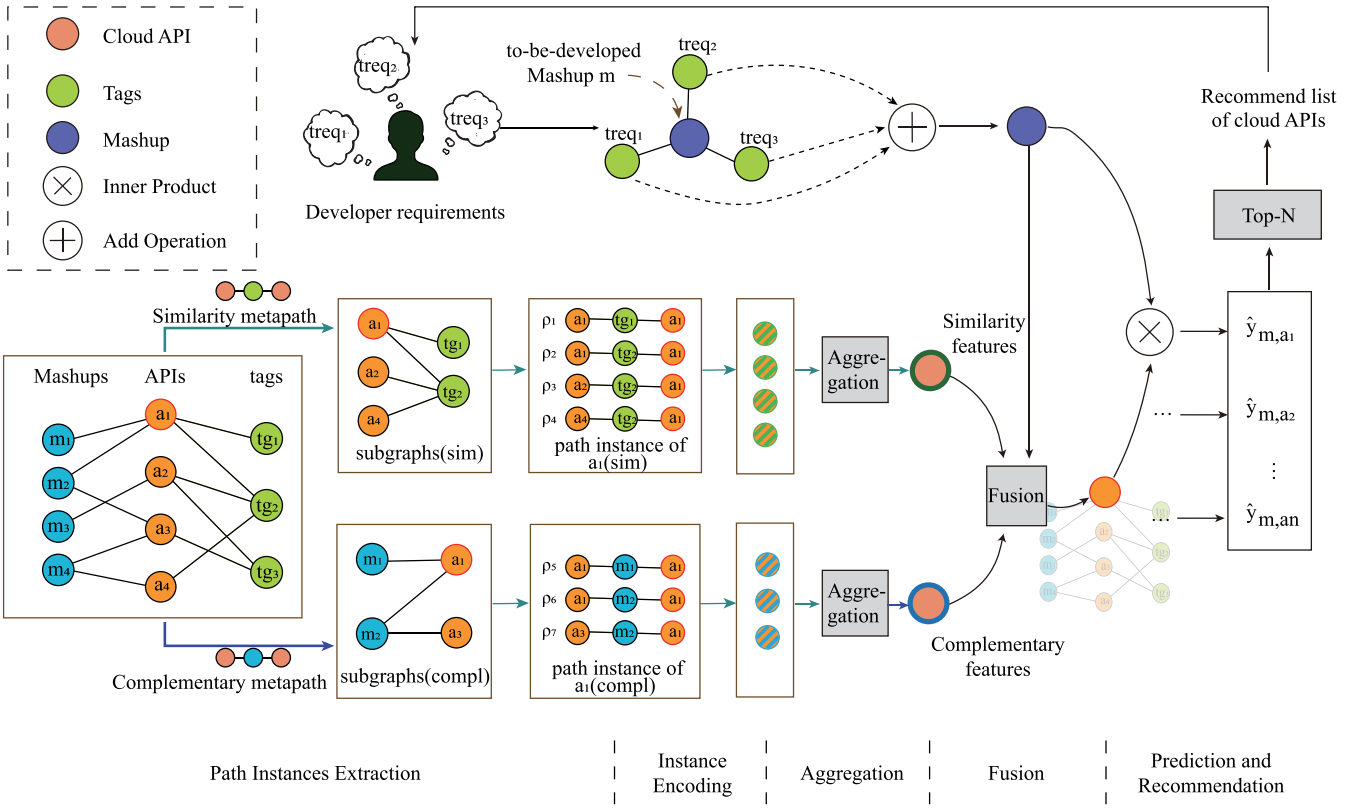


FIGURE 2. The overall framework of the proposed mashup-oriented cloud API recommendation method.

(4) Prediction and recommendation: the invocation probabilities of cloud APIs are calculated and ranked based on mashup requirement and API features, recommending the Top-N APIs with the highest probabilities of invocation.

While traditional heuristic-based approaches offer simplicity and computational efficiency, they lack the ability to model the complex dependencies and diverse functional requirements inherent in mashup creation. These approaches do not take into account the relational structure between api's, which is critical for recommending api's that work well together in a mashup. To overcome these limitations, the approach utilizes graph neural networks (GNN) and translational distance models. The graph neural network aggregates information from neighboring nodes in the knowledge graph to capture relational and contextual information that is critical for accurate recommendations, thus providing a more comprehensive understanding of the API outlook.

A. NODE REPRESENTATION

Functional tags provide succinct functional descriptions of cloud APIs and mashups in the cloud API ecosystem. By leveraging standardized functional tags, cloud API characterization features and mashup requirements can be represented in a unified vector space with fewer parameters. Furthermore, this approach addresses issues related to vocabulary discrepancies and language inconsistencies that frequently emerge from the utilization of non-standardized terminology and disparate language descriptions employed by developers. For example, APIs with analogous functionalities may be described using disparate terminologies or in disparate languages, which could result in potential challenges in accurate recommendations. By focusing on functional tags, it is possible to ensure that the recommendation system is robust to such variances and remains consistent across different contexts [31]. By combining multiple different functional labels, cloud APIs and mashups can be well characterized, which has been found to be successful in many information systems and also applies to the cloud API ecosystem [32].

Therefore, we utilize multi-hot encoding [33], each cloud API or mashup can be represented as a vector in which each dimension corresponds to a functional tag, capturing capture the functionality or requirements. This approach ensures that the representation can handle cases where an API or mashup has multiple functional tags, providing a more flexible encoding.

To achieve this, assume there are n functional tags. We generate a multi-hot encoding vector of length n for each API to represent the functional tags it contains. For example, if an API a has the 1st and 3rd tags, its multi-hot encoding would be $[1, 0, 1, 0, \ldots, 0]$, where 1 indicates that the cloud API has this tag and 0 indicates not.

Next, we convert the multi-hot encoding vector into a lower-dimensional embedding representation. Specifically, we define an embedding matrix $W \in \mathbb{R}^{n\times d}$, where n is the number of functional tags and d is the dimension of the embedding vector. By looking up the corresponding columns in the embedding matrix, each non-zero position in the multi-hot encoding vector is mapped to its corresponding embedding vector. Then, these embedding vectors are averaged to obtain the final feature representation of the cloud API:

$$e_a = \frac{1}{|N_a^{\text{tag}}|} \sum_{k \in N_a^{\text{tag}}} x_{a,k} W_k, \tag{2}$$

where N_a^{tag} denotes the set of tags associated with the cloud API a, $x_{a,k}$ is the value (0 or 1) in the multi-hot encoding vector for the k-th tag, and W_k is the embedding vector for the k-th column of the embedding matrix.

Similarly, for mashup requirements, which are also described by functional tags provided by the developers, the multi-hot encoding vector $x_m \in \{0, 1\}^n$ can be used to represent the features of the mashup. The mashup embedding is calculated as:

$$e_m = \frac{1}{|\text{Treq}_m|} \sum_{k \in \text{Treq}_m} x_{m,k} W_k, \tag{3}$$

where Treq_m represents the set of functional tags required by the mashup, and $x_{m,k}$ indicates whether the k-th tag is present in the multi-hot encoding vector.

Through this process of multi-hot encoding and embedding, the features of both cloud APIs and mashups can be mapped into the same lower-dimensional vector space, achieving a unified feature representation. This method not only captures the relationships between functional tags but also provides more accurate semantic information for the recommendation system.

B. PATH INSTANCES EXTRACTION

In an information graph, two nodes are connected through various semantic paths known as metapaths, which do not rely on direct connections between nodes but are constructed based on higher-order relationships [30]. Using these paths, similarities and complementarities between cloud API nodes can be identified. According to the description in Section III-B, in the cloud API ecosystem graph, if two cloud API nodes form an indirect connection through one or more functional tags as intermediary nodes, this connection pattern indicates that these two cloud APIs have functional similarities. If two cloud API nodes are both connected to the same mashup node, this implies the potential for them to exhibit functional complementarities, as they have been co-invoked for the same application scenario. Based on these relationships between the nodes, the set of metapaths associated with the cloud API is formally defined as follows:

$$\Phi = \{\mathcal{P}_{\text{sim}}, \mathcal{P}_{\text{compl}}\}, \tag{4}$$

where P_{sim} denotes the similar metapath "cloud API - Function Tag - cloud API" and P_{compl} denotes the complementary metapath "cloud API - mashup - cloud API." Different metapaths can reveal various semantics, and a sequence of nodes that follows the pattern defined by a metapath is termed a path instance.

Based on the defined metapaths $\mathcal{P}$sim and $\mathcal{P}$compl, the cloud API ecosystem graph is divided into two distinct subgraphs. In each metapath-based subgraph, the set of path instances where the target cloud API node is the endpoint is obtained, denoted by p^a_{sim} and p^a_{compl}. For example, to extract the path instances related to cloud API node a_1 in Fig. 2, two subgraphs are first divided based on the aforementioned metapaths: one subgraph contains only cloud API nodes and tag nodes, and the other subgraph contains only cloud API nodes and mashup nodes. Path instances that can represent different attributes are acquired in each subgraph. Similar path instances of cloud API node a_1 based on metapath $\mathcal{P}_{\text{sim}}$ include $\rho_1 : a_1 - tg_1 - a_1$, $\rho_2 : a_1 - tg_2 - a_1$, $\rho_3 : a_2 - tg_2 - a_1$, and $\rho_4 : a_4 - tg_2 - a_1$, i.e., $p^{a_1}_{\text{sim}} = \{\rho_1, \rho_2, \rho_3, \rho_3\}$. Similarly, complementary path instances are denoted as $p^{a_1}_{\text{compl}} = \{\rho_5, \rho_6, \rho_7\}$.

C. INSTANCE ENCODING AND AGGREGATION

Given a target cloud API node a and a metapath P, the encoding of path instances is employed to learn the contextual information of the node and capture its hidden features. The formalization of path instances is as follows:

$$\rho : e_u \xrightarrow{r_1} e_v \xrightarrow{r_2} e_a, \tag{5}$$

where ρ denotes an instance of the metapath $\mathcal{P}$ ending at node a, and u, v denote other nodes in the path instance, and r_1, r_2 denote relationships between nodes. e denotes the embedding of a node.

The previous metapath-based graph embedding model [34] only considered information from the two endpoints of the metapath and ignored all intermediate nodes, as shown in Fig. 3 (a), resulting in significant loss of information. Inspired by the knowledge graph embedding technique [35], [36], we adopt a translational distance model similar to TransE to design a path instance encoder, which utilizes all the nodes and relationships in the path instances and encodes them into a unified feature, as shown in Fig. 3 (b).

The basic idea of TransE is to represent associations between entities and relations by vector operations, i.e., for a triple (h, r, t), where h and t denote the head entity and tail entity, respectively, and r denotes the relation, TransE tries to make $h + r \approx t$ hold [36]. This means that the vector of head entities h plus the vector of relations r should be close to the vector of tail entities t. Thus, both entities and relations are modeled and optimized in the same vector space. Slightly different from the knowledge graph embedding method transE described above, there are a total of three nodes and two relations in a meta-path instance, and

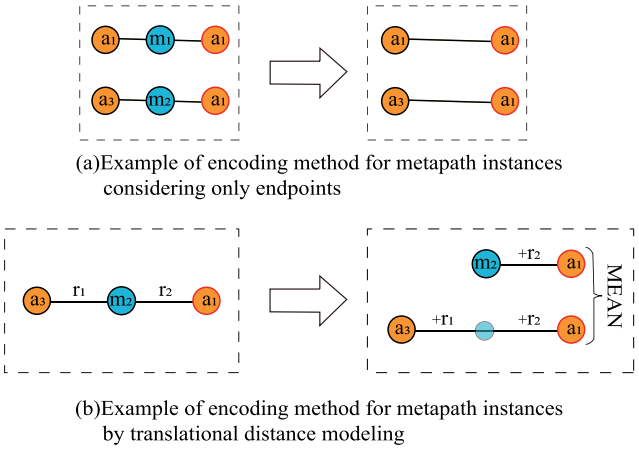


FIGURE 3. Comparison of different encoding methods for metapath instances.

the formulas adapted to meet our needs are as follows:

$$h_a^\rho = \text{MEAN}((e_u + r_1 + r_2), (e_v + r_2)). \tag{6}$$

Moreover, unlike knowledge graph embeddings, in this study relations are not explicitly characterized but are represented by learnable embedding vectors of the same dimension as the nodes. This design enables the model to learn the features of the relationships, thereby improving the representation of path instance encoding.

After the path instances are encoded as independent feature vectors, it is crucial to consider that the significance of different path instances to the target node varies. According to the graph attention neural network algorithm [34], the path instances of the target node are weighted and summed.

$$\alpha_a^\rho = \frac{\exp(\sigma_1(q_p^T(e_a||h_a^\rho)))}{\sum_{\rho' \in p} \exp(\sigma_1(q_p^T(e_a||h_a^{\rho'})))}, \tag{7}$$

where, $||$ represents vector concatenation. p denotes the instance set of the target node a on the metapath $\mathcal{P}$. q_p is the attention vector, and $\sigma_1(\cdot)$ is the activation function. The attention coefficients and the corresponding path instance features are linearly combined to obtain the cloud API node feature representation based on the metapath $\mathcal{P}$ as follows:

$$e_a^{\text{P}} = \sigma_2(\sum\nolimits_{\rho \in p} \alpha_a^\rho h_a^\rho), \tag{8}$$

where $\sigma_2(\cdot)$ denotes the activation function. Based on the metapath set defined in Section IV-B, the above aggregation operation is performed for each metapath, and the feature representation of the cloud API node under each metapath $\{e_a^{\text{sim}}, e_a^{\text{compl}}\}$ is obtained.

D. MULTIPLE ATTRIBUTE ATTENTION FUSION

After obtaining various attribute features of cloud APIs from different metapaths, the feature representation set $e_a^{\text{sim}}, e_a^{\text{compl}}$ of cloud API node a is derived, containing diverse semantic information. To achieve a comprehensive cloud API representation, these various attribute features must be fused. Existing methods typically utilize mean attention or global self-attention to fuse various semantic features, which evidently ignores the unique requirements of different mashups. In cloud API recommendations for mashups, different mashups have diverse preferences for similarity and complementarity. This implies that semantic information from different metapaths in the cloud API ecosystem graph varies in importance for mashups with distinct requirements.

Therefore, considering the impact of diverse requirements on the comprehensiveness and personalized representation of cloud API features, we designed a mashup requirement-related attention module to fuse multiple attribute features. Using the requirement features of a mashup as a query and the various attribute features of a cloud API as keys and values, we learn the weights of different attribute features to determine their importance.

Specifically, for a given cloud API a, we first concatenate the feature representation vectors under each metapath to obtain $E_a = [e_a^{\text{sim}}, e_a^{\text{compl}}]$ and define $Q = e_m W_Q, K = E_a W_K$, and $V = E_a W_V$. Here, W is the weight coefficient, and Q, K, and V are the outputs of the features after linear transformation. The query Q and key K are used to calculate the attention score, and the weights of each type of attribute feature are obtained through the normalization process:

$$\beta_m = \text{softmax}(\frac{QK^{\text{T}}}{\sqrt{d}}), \tag{9}$$

where d is the dimension of the key matrix, and softmax is the normalized exponential function.

Finally, the weights of each type of attribute features related to the mashup are utilized as personalization filters to fuse different meta-paths of cloud API node representations. The final feature embedding of a cloud API node a for a given requirement mashup m is defined as follows:

$$e_a^m = \sum\nolimits_{P \in \Phi} \beta_m^P e_a^P. \tag{10}$$

E. PREDICTIONS AND RECOMMENDATIONS

After computing the personalized embeddings relevant to the target mashup, incorporating both cloud API similarity and complementary features, the probability of the mashup invoking the target cloud API is predicted.

$$\hat{y}_{m,a} = \sigma_3(e_a^m \cdot e_m). \tag{11}$$

The mashup-oriented cloud API recommendation involves an implicit feedback problem, where a Sigmoid function restricts the output to the (0,1) range to represent the probability of a mashup invoking the cloud API. The N cloud APIs with the highest invocation probabilities are recommended for developers to use in mashup development. The loss function is defined using cross-entropy:

$$\begin{aligned} Loss = -\sum\nolimits_{(m,a) \in Y} & (y_{m,a} \log(\hat{y}_{m,a}) \\ & + (1 - y_{m,a}) \log(1 - \hat{y}_{m,a})), \end{aligned} \tag{12}$$

where Y includes mashup-API interaction data and random negative samples.

To present the proposed recommendation method, a workflow for mashup-oriented cloud API recommendation is provided in Algorithm 1.

Algorithm 1 Cloud API Recommendation Method for Mashups

Require: interaction matrix $\hat{\mathbf{Y}} = (\hat{y}_{ma})_{|M|\times|A|}$, API ecosystem graph $\mathcal{G} = (\mathcal{V}, \mathcal{E})$, metapath set $\Phi = \{\mathcal{P}_{\text{sim}}, \mathcal{P}_{\text{compl}}\}$, mashup requirement set $\text{Req} = \{\text{Treq}_m | m \in M\}$;
Ensure: Cloud API recommendation list;
Initialize cloud API feature representation;
for epoch $= 1, 2, 3, \ldots$ **do**
  for target mashup $m \in M$ **do**
    Calculate requirement representation e_m by Eq.3;
    for API $a \in A$ **do**
      for path $\mathcal{P} \in \Phi$ **do**
        Extract the path instances of API a;
        Encode the path instances by Eq.6;
        Aggregate the path instances of API a in metapath $\mathcal{P}$ by Eq.7-8;
      end for
      Fusion multiple attribute features to obtain the final feature representation $\mathbf{e}_a^m$ related to the mashup requirement by Eq.9-10;
      Calculate the predicted probability $\hat{y}_{ma}$ by Eq.11;
    end for
    Rank the predicted probabilities and create Top-N APIs list;
  end for
  Calculate the loss by Eq.12 and update the parameters.
end for
return Cloud API recommendation list

V. EXPERIMENTATION AND ANALYSIS

A. PREPARATION

1) DATASET AND EXPERIMENT SETTING

To compare and validate the effectiveness of the proposed method, real data were collected from the ProgrammableWeb platform using a web crawler. Specifically, the dataset includes 6019 mashups, 1509 cloud APIs that have been invoked at least once, and 440 attribute tags. The total number of mashup-API invocations is 12564 and the mashup-API matrix has a sparsity of 99.86%, which indicates a high level of sparsity typically found in such recommendation datasets.

After data cleaning, we obtained the refined experimental dataset as shown in Table 1. Notably, this dataset does not contain explicit mashup-API rating data, thus the mashup-oriented cloud API recommendation problem is framed as an implicit feedback task. Specifically, if a mashup has historically invoked a cloud API, the corresponding record is marked as 1, indicating the mashup's requirement for the cloud API. Conversely, cloud APIs that were not used by mashups were marked as 0 to serve as negative samples.

TABLE 1. The statistics of the dataset.

Item type	Statistics
Number of mashups	6019
Number of APIs invoked by all mashups	1509
Number of tags	440
Number of invocations	12564
Average number of Web APIs used per mashup	2.09
Average number of tags used per Web API	2.73
Sparsity	99.86%

For the experimental setup, he dataset was divided into three parts: 80% for training, 10% for validation, and 10% for testing. The embedding size was set to 64, and the learning rate was set to 1 in the experiment. An early stopping strategy was used: the algorithm was deemed converged if the training dataset loss increased or the Recall metrics on the validation set decreased for 10 consecutive epochs.

2) EVALUATION INDICATORS

To evaluate the performance of top-N recommendation, two commonly used metrics are chosen: recall and normalized discounted cumulative gain (NDCG).

Recall is a metric used to evaluate how effectively the system recommends relevant cloud APIs to developers. It measures the proportion of relevant cloud APIs that are successfully recommended to a mashup, and is defined as:

$$\text{Recall} = \frac{|\{\text{Act APIs}\} \cap \{\text{Rec APIs}\}|}{|\{\text{Act APIs}\}|}, \tag{13}$$

where {Act APIs} is the set of APIs that are actually invoked, {Rec APIs} is the set of APIs recommended, $|\cdot|$ denotes the size of the set.

NDCG is a measure of the quality of a recommender system's ranking. It assesses the usefulness of an item based on its position in the ranked list of results. A higher score is assigned to top-ranked relevant recommendations, indicating their importance. The NDCG is defined as follows:

$$NDCG@k = \frac{\sum_{i=1}^{k} \frac{\text{rel}_i}{\log_2(i+1)}}{\sum_{i=1}^{k} \frac{\text{rel}_i^*}{\log_2(i+1)}}, \tag{14}$$

where rel_i is the relevance score of the item at position i in the recommended list.rel_i^* is the relevance score of the item at position i in the ideal (perfectly sorted by relevance) list.

These metrics together provide a comprehensive evaluation of both the accuracy and ranking quality of the recommended cloud APIs.

B. PERFORMANCE COMPARISON

1) COMPARISON METHODS

There is no widely recognized benchmark model dedicated to mashup-oriented cloud API recommendation. To evaluate performance, the proposed methods are compared with several widely used recommendation methods, some of which have been applied in the field of cloud API recommendation.

The baseline recommendation methods selected can be categorized into three groups: interaction-based methods (POP, BPR [37], DMF [38]), content-based methods (DeepFM [12], AFM [39]) and graph-based methods (KGCN [40], KGNNLS [41], KGAT [42]). The following is a brief overview of the aforementioned recommendation methods:

- The POP method makes recommendations based on the popularity of cloud APIs in the dataset.
- The BPR algorithm utilizes the maximum a posterior probability in a Bayesian framework to rank cloud APIs and generate recommendations.
- DMF applies deep learning techniques to extend matrix factorization for recommendations.
- DeepFM combines factorization machines and deep learning techniques to enhance the performance of recommendation systems.
- AFM optimizes the factorization machine by determining the importance of inter-feature interactions through neural network learning.
- KGCN enriches the embedding representation by aggregating information from neighboring nodes to capture the local structure and extract personalized preferences.
- KGNNLS proposes a method based on label smoothness to regularize edge weights and enhance model generalization.
- KGAT combines graph neural networks and knowledge graphs to integrate user behavior data and introduce auxiliary higher-order information.

2) COMPARISON RESULTS

In the domain of mashup-oriented cloud API recommendation, traditional heuristics method, such as popularity-based (POP) methods and Bayesian personalized ranking (BPR), are frequently utilized due to their simplicity and ease of implementation. However, these methods exhibit substantial limitations in effectively capturing the complex dependencies and functional requirements of application programming interfaces within mashup contexts. For instance, the POP approach relies exclusively on the frequency of usage of application interfaces and neglects the specific functional dependencies that exist between application interfaces and their usage within the mashup ecosystem. Although BPR is more sophisticated, it fails to leverage the structural information inherent in API interactions, resulting in suboptimal performance in complex recommendation scenarios.

The performance comparison in Table 2 shows that methods integrating side information (DeepFM, AFM) and graph-based approaches (KGCN, KGNNLS, KGAT, and our proposed method) significantly outperform traditional methods like POP, BPR, and DMF on recall and NDCG metrics. This trend underscores the value of leveraging additional data sources and structural relationships for more effective recommendation strategies. Methods that incorporate side information can capture attributes beyond basic interactions, while graph-based methods model the connections and dependencies between APIs, providing a richer representation that boosts recommendation accuracy.

Our proposed method stands out among graph-based approaches, achieving at least a 5% improvement in recall and a 10% increase in NDCG compared to other techniques. The reasons behind this enhancement lie in our method's ability to integrate graph structural features, interaction data, and edge information (e.g., cloud API features), enabling it to capture the multidimensional relationships and dependencies more effectively than the existing methods. The use of multiple attribute features, such as functionality similarity and complementarity, contributes to a more personalized and precise recommendation list.

GNN-based models excel in capturing multidimensional relationships and dependencies among APIs, thereby significantly enhancing the accuracy of mashup-oriented recommendations. The proposed approach employs advanced techniques to deliver recommendations that are both accurate and tailored to the specific needs of developers, thereby outperforming existing methodologies. Specifically, the experiments substantiate the premise that integrating edge information (e.g., cloud API features) into recommendation algorithms enhances both relevance and accuracy. By integrating graph structural features, interaction information, and edge information into a personalized recommendation framework for mashup applications, significant improvements in recommendation accuracy and ranking rationality are attained. These findings illustrate the critical role of diverse information in the efficacy of cloud API recommender systems and highlight the potential of the proposed method, which integrates multiple attribute features, in delivering efficient and personalized recommendations.

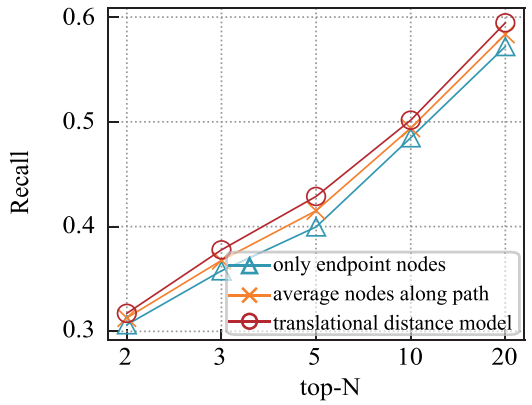
C. IMPACT OF INSTANCE ENCODERS

After completing the path instance extraction, we design a path instance encoder based on the translational distance model, aiming to enhance the accuracy of the recommender system through a finer path representation. To evaluate the impact of the proposed encoder on the performance of the recommender system, we experimentally compare it with several other commonly used encoding methods. Specifically, we consider three different encoding strategies: (1) only the information of endpoint nodes is utilized, ignoring intermediate nodes; (2) all the information of the nodes along the path is simply averaged; (3) all the information of nodes and relationships on the path instances is considered based on the translational distance model, which is our proposed method.

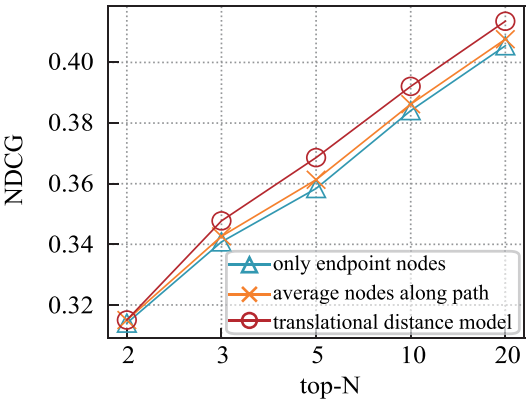
The experimental results are shown in Fig. 4. Taking top-10 recommendations as an example, we find that the encoder based on the translational distance model improves recall by 3.46% and NDCG by 2.08% compared to the method that only considers endpoints. The method that utilizes only the information of endpoint nodes, while ignoring intermediate nodes, fails to capture the full path structure, resulting in lower accuracy. Similarly, the

TABLE 2. Comparison of baseline methods and proposed method.

Methods	Recall					NDCG				
	top-2	top-3	top-5	top-10	top-20	top-2	top-3	top-5	top-10	top-20
POP	0.0075	0.0207	0.0368	0.0554	0.2339	0.0069	0.0136	0.0203	0.0263	0.0685
BPR	0.0725	0.0873	0.1205	0.1784	0.2437	0.0622	0.0696	0.0833	0.1019	0.1184
DMF	0.2136	0.2479	0.3172	0.3927	0.4945	0.1916	0.2087	0.2375	0.2625	0.2882
deepFM	0.1706	0.2149	0.2892	0.3772	0.4669	0.1432	0.1756	0.2154	0.2482	0.2796
AFM	0.2491	0.2928	0.3563	0.4122	0.5265	0.2214	0.2452	0.2835	0.3068	0.3159
KGCN	0.2386	0.2386	0.3430	0.3918	0.5388	0.2141	0.2141	0.2683	0.2892	0.3189
KGNNLS	0.2306	0.2907	0.3718	0.4502	0.5473	0.2033	0.2381	0.2770	0.3060	0.3245
KGAT	0.1936	0.2512	0.3335	0.4264	0.5662	0.1640	0.1952	0.2299	0.2713	0.2978
Ours	**0.3172**	**0.3778**	**0.4288**	**0.5016**	**0.5945**	**0.3150**	**0.3477**	**0.3668**	**0.3921**	**0.4135**



(a)Recall



(b)NDCG

FIGURE 4. The impact of path instance encoding methods on Recommendation performance.

method that averages the features of all nodes along the path considers the node information but overlooks the relationships between the nodes, which also affects the model's precision. Compared to this averaging approach, our proposed method improves recall by 1.53% and NDCG by 1.52%. These significant improvements indicate that the translational distance model effectively captures the intrinsic structure of path instances, thus providing a richer and more precise instance representation for the recommender system.

These results suggest that incorporating intermediate node information and the relationships between nodes plays a crucial role in enhancing the recommendation quality. The translational distance model's ability to encode complex path information allows it to uncover latent connections that simpler encoding strategies might miss. As a result, it facilitates a more comprehensive understanding of the interactions within the graph, leading to improved recommendation accuracy. This demonstrates the importance of using sophisticated encoding techniques to fully exploit the structural information present in path instances for cloud API recommendation systems.

D. IMPACT OF MULTIPLE ATTRIBUTE FEATURES FUSION METHODS

In this study, we explore the effectiveness of different attention mechanisms in fusing multiple attribute features of cloud APIs. In our experiments, we compare the performance of equal attention, global self-attention, and our proposed mashup-related attention mechanism. Fig. 5 illustrates the performance of these three mechanisms with various top-N values.

The experimental results show that equal self-attention and global self-attention have similar effects when fusing cloud API features, which suggests that both similar and complementary features are important for cloud API recommendation without considering specific mashup requirements. However, equal attention treats all the different attribute characteristics as equally important, completely ignoring the requirement preferences of a specific mashup, whereas global self-attention focuses on the global characteristics of the whole cloud API ecosystem without considering the specific requirements of a mashup. As a result, both approaches are unable to provide personalized recommendations targeted to each mashup, thus limiting their effectiveness in achieving highly personalized recommendations.

In contrast, our proposed mashup-related attention mechanism demonstrates significant performance benefits across all top-N values, with improvements in recall ranging from 6% to 17% and in NDCG from 6% to 12% compared to the other mechanisms. These performance improvements highlight the ability of the attention mechanism associated with mashups to

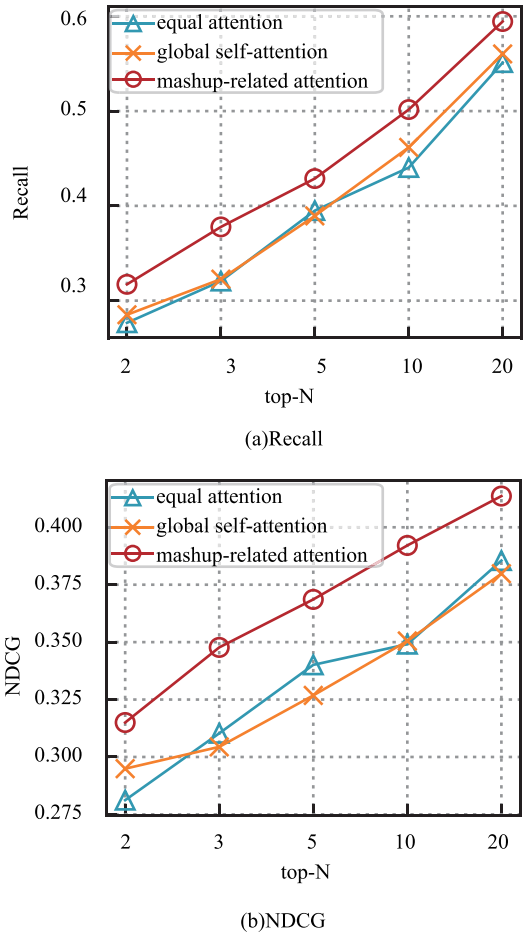


FIGURE 5. The impact of multiple attribute features fusion methods on recommendation performance.

recognize the importance of different attribute features based on the specific requirements of each mashup, and the superior ability of cloud API recommendation method that leverage this attention mechanism to meet the requirements of highly personalized mashups.

E. ABLATION EXPERIMENT

In this study, we propose a cloud API recommendation method combining metapath and attention mechanisms. By distinguishing the similar and complementary features of cloud APIs, we gain a deeper understanding of their intrinsic connections and their roles in diverse requirement scenarios. The proposed method leverages the ''cloud API-Function Tag-cloud API'' metapath to capture similarity features and the ''cloud API-mashup-cloud API'' metapath to identify complementary features, thus providing a comprehensive representation of cloud API characteristics.

To validate the effectiveness of the proposed method, we conducted a series of ablation experiments, with the results presented in Table 3. Specifically, 'w/o ATA' indicates the removal of the similar metapath 'cloud API-Tags-cloud API' from the experiment, while 'w/o AMA' indicates the removal of the complementary metapath 'cloud API-mashup-cloud API'. These experiments aim to separately evaluate the impact of different metapaths on the performance of the recommender system.

The experimental results clearly indicate that combining similarity and complementary features significantly enhances the recommender system's performance compared to considering only a single feature. When similarity features are excluded ('w/o ATA'), the recommendation's ability to identify APIs that fulfill the same functional requirements diminishes, leading to a significant recall drop to 80% and NDCG drop to 47% for top-10 recommendations. This is because without capturing similarity through the 'cloud API-Tags-cloud API' metapath, the system fails to adequately recognize APIs with similar functionalities, which are crucial in scenarios where multiple services provide comparable solutions.

On the other hand, when complementary features are excluded ('w/o AMA'), recall drops to 84% and NDCG drops to 78%. The absence of the 'cloud API-mashup-cloud API' metapath hinders the system's ability to identify APIs that complement existing functionalities, reducing its capability to recommend APIs that expand the mashup's functionality. Complementary features are essential for enriching the mashup with diverse services, thus their absence significantly impacts recommendation quality.

Moreover, excluding the attention mechanism ('w/o attention') causes recall to drop to 89% and NDCG to 84%. The attention mechanism plays a critical role in assigning appropriate weights to different path instances based on their relevance to the target node, enabling the model to prioritize more informative features. Without this mechanism, the recommendation system treats all path instances equally, leading to less accurate feature extraction and a consequent decline in performance.

After comprehensive analysis, we conclude that the application of metapath and attention mechanisms in each metapath is vital for enhancing the performance of cloud API recommendation systems. This combined approach enables better alignment with mashup-oriented cloud API requirements, providing more accurate and efficient recommendation services.

F. CASE STUDY

To visualize the usability of the method, we provide an example of a mashup-oriented cloud API recommendation. Suppose a developer needs to create a mashup displaying items being purchased or viewed by the user's friends on a shopping platform and providing the user with a rating system as a reference. Based on the functional requirements, the developer can select a combination of functional tags Social, eCommerce, Reference from the cloud API ecosystem to represent the required features of the mashup.

An attention graph neural network was used to learn the similarity and complementarity features of candidate cloud APIs respectively. Then, the attention score for similarity

TABLE 3. Comparison results of ablation experiments.

Method	Recall					NDCG				
	top-2	top-3	top-5	top-10	top-20	top-2	top-3	top-5	top-10	top-20
w/o ATA	0.0133	0.2015	0.3064	0.4022	0.5160	0.0126	0.1112	0.1554	0.1857	0.2103
w/o AMA	0.2104	0.2553	0.2690	0.4202	0.5227	0.2250	0.2485	0.2541	0.3042	0.3282
w/o attention	0.2324	0.2818	0.3817	0.4462	0.5288	0.2391	0.2642	0.3052	0.3289	0.3490
Ours	**0.3172**	**0.3778**	**0.4288**	**0.5016**	**0.5945**	**0.3150**	**0.3477**	**0.3686**	**0.3921**	**0.4135**

TABLE 4. Top-5 recommended list of the case.

Rank	Cloud API	probability	Function tags	Real invoke
1	Twitter	0.974	social,blogging	No
2	**Youtube**	0.966	Video,Media	Yes
3	**amazon-product-advertising**	0.917	Advertising,eCommerce	Yes
4	**Facebook**	0.892	Social,Webhooks	Yes
5	**Flickr**	0.865	Photos,Video	Yes

TABLE 5. Top-5 recommended list of ablation experiments.

Method	Rank	Cloud API	First tag	Real invoke
Only similarity	1	yahoo-maps	Mapping	No
	2	Twitter	social	No
	3	google-maps	Mapping	No
	4	google-earth	Mapping	No
	5	**Youtube**	Video	Yes
Only complementarity	1	reuters-spotlight	News	No
	2	Ribbit	Telephony	No
	3	google-maps	Mapping	No
	4	Flickr	Photos	No
	5	Twitter	social	No
Self-attention	1	google-maps	Mapping	No
	2	Flickr	Photos	No
	3	**Youtube**	Video	Yes
	4	Twitter	social	No
	5	**Facebook**	social	Yes

and the attention score for complementarity of cloud APIs are computed for the mashup, which are 0.3024 and 0.6976, respectively. Based on these scores, the features of candidate cloud APIs are fused and used to predict the likelihood of the mashup invoking the respective cloud APIs. The top-ranked candidate cloud APIs are recommended to the mashup, and the Top 5 recommendation list is shown in Table 4.

The mashup described actually invokes six APIs: amazon-product-advertising (Advertising, eCommerce), amazon-a9-opensearch (Search), youtube (Video, Media), facebook (Social, Webhooks), amazon-simpledb (Database), and amazon-marketplace-web-service (eCommerce). The method successfully recommended three cloud APIs in the Top 5 recommendations, at the 2nd, 3rd, and 4th positions. The attention score suggests that both similarity and complementarity should be considered when selecting APIs for this functional mashup, with a higher requirement for complementarity. The recommended cloud APIs do contain diverse functional tags, providing versatility.

To further demonstrate the effectiveness of the proposed method, the results of three cloud API recommendation ablation experiments for this mashup are shown in Table 5. When the recommendation considers only similarity, it usually includes multiple cloud APIs with similar functionality, which is not suitable for all types of mashups. When the recommendation considers only complementarity, it recommends multiple types of APIs but overlooks the functional similarity between them, which may result in the recommended APIs failing to satisfy the functionality of the mashup. Using a global self-attention mechanism to fuse different types of attribute features, the self-attention scores for similarity and complementarity of cloud APIs for the mashup to be developed are 0.5110 and 0.4890, respectively. This approach fails to provide personalized recommendations for mashups. To address these problems, the proposed method employs a mashup-associated attention mechanism to achieve a personalized cloud API recommendation list.

VI. CONCLUSION

In this paper, we propose an innovative approach for personalized feature representation and recommendation of cloud APIs through analysis of the problems in the cloud API mashup recommendation system. We recognize that to satisfy developers' personalized development requirements, the recommender system must comprehensively consider

multiple attribute features of cloud APIs, especially similarity and complementarity. Firstly, proposed method involves the construction of a cloud API ecosystem graph enriched with side-information and the formulation of metapaths to capture and characterize the multiple attributes of cloud APIs. Utilizing the translational distance model and graph neural network techniques, we successfully aggregated the attribute features of cloud APIs within different metapaths. Additionally, we achieved the intelligent fusion of various attribute features by introducing an attention mechanism closely related to mashup development requirements. This fusion mechanism optimizes feature weights and generates highly relevant personalized representations of cloud APIs, thereby providing strong support for predicting the likelihood of a mashup invoking the APIs. Extensive experimental evaluations on real datasets validate its effectiveness. The experimental results show that our recommendation method has significant advantages in recalling correct cloud APIs and improving recommendation performance, demonstrating the potential of our approach in meeting personalized development requirements. However, our approach also has some limitations. One limitation is that the current model may not fully capture the dynamic and evolving nature of the cloud API ecosystem. Additionally, the model's reliance on pre-defined metapaths might limit its adaptability to diverse and complex development scenarios.

In the future, we aim to address these limitations by incorporating more dynamic and temporal aspects into the model and exploring adaptive metapath design methodologies. We plan to explore and utilize multidimensional information such as version, quality data, and vendor information to enhance cloud API feature representation. Furthermore, we will focus on designing adaptive metapaths to better capture the diverse and evolving information within the cloud API ecosystem. These enhancements will help us to better utilize the diverse information in the cloud API ecosystem and further improve the accuracy and efficiency of recommendation systems.

REFERENCES

[1] Y. Wang, J. Chen, Q. Huang, X. Xia, and B. Jiang, "Deep learning-based open API recommendation for mashup development," *Sci. China Inf. Sci.*, vol. 66, no. 7, Jul. 2023, Art. no. 172102.

[2] J. Xiang, W. Chen, Y. Wang, B. Liang, Z. Liu, and G. Kang, "Interactive web API recommendation for mashup development based on light neural graph collaborative filtering," in *Proc. IEEE Int. Conf. Comput. Supported Coop. Work Des. (CSCWD)*, Rio de Janeiro, Brazil, 2023, pp. 1926–1931.

[3] H. Xue, X. Dai, J. Zhang, S. Huang, and J. Chen, "Deep matrix factorization models for recommender systems," in *Proc. 26th Int. Joint Conf. Artif. Intell. (IJCAI)*, Melbourne, VIC, Australia, Aug. 2017, pp. 1–7.

[4] S. Lian and M. Tang, "API recommendation for mashup creation based on neural graph collaborative filtering," *Connection Sci.*, vol. 34, no. 1, pp. 124–138, Dec. 2022.

[5] 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.

[6] 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, Aug. 2023.

[7] W. Gong, C. Lv, Y. Duan, Z. Liu, M. R. Khosravi, L. Qi, and W. Dou, "Keywords-driven web APIs group recommendation for automatic app service creation process," *Software, Pract. Exper.*, vol. 51, no. 11, pp. 2337–2354, Nov. 2021.

[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] B. Cao, J. Liu, Y. Wen, H. Li, Q. Xiao, and J. Chen, "QoS-aware service recommendation based on relational topic model and factorization machines for IoT mashup applications," *J. Parallel Distrib. Comput.*, vol. 132, pp. 177–189, Oct. 2019.

[10] 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.

[11] 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.

[12] 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 Computation: Pract. Exper.*, vol. 34, no. 21, p. e7069, Sep. 2022.

[13] Y. Chen, C. Gao, X. Ren, Y. Peng, X. Xia, and M. R. Lyu, "API usage recommendation via multi-view heterogeneous graph representation learning," *IEEE Trans. Softw. Eng.*, vol. 49, no. 5, pp. 3289–3304, May 2023.

[14] P. He, W. Qi, X. Liu, L. Liu, D. You, L. Shen, and Z. Chen, "Association rule guided web API complementary function recommendation for mashup creation: An explainable perspective," in *Proc. Int. Conf. Comput. Supported Coop. Work Social Comput. (CSCWSC)*, Singapore, 2023, pp. 73–83.

[15] K. K. Fletcher, "A quality-aware web API recommender system for mashup development," in *Proc. IEEE Int. Conf. Serv. Comput. (SCC)*, San Diego, CA, USA, Jun. 2019, pp. 1–10.

[16] K. A. Botangen, J. Yu, Q. Z. Sheng, Y. Han, and S. Yongchareon, "Geographic-aware collaborative filtering for web service recommendation," *Expert Syst. Appl.*, vol. 151, Aug. 2020, Art. no. 113347.

[17] S. Meng, Q. Li, S. Chen, S. Yu, L. Qi, W. Lin, X. Xu, and W. Dou, "Temporal-sparsity aware service recommendation method via hybrid collaborative filtering techniques," in *Proc. Int. Conf. Serv.-Oriented Comput.*, Hangzhou, China, Nov. 2018, pp. 12–15.

[18] 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.

[19] S. Wu, S. Shen, X. Xu, Y. Chen, X. Zhou, D. Liu, X. Xue, and L. Qi, "Popularity-aware and diverse web APIs recommendation based on correlation graph," *IEEE Trans. Computat. Social Syst.*, vol. 10, no. 2, pp. 771–782, Apr. 2023.

[20] J. Zhang, Y. Fan, J. Zhang, and B. Bai, "Learning to build accurate service representations and visualization," *IEEE Trans. Services Comput.*, vol. 15, no. 3, pp. 1551–1563, May 2022.

[21] X. Zhang, J. Liu, M. Shi, and B. Cao, "Word embedding-based web service representations for classification and clustering," in *Proc. IEEE Int. Conf. Serv. Comput. (SCC)*, 2021, pp. 34–43.

[22] B. Bai, Y. Fan, W. Tan, and J. Zhang, "DLTSR: A deep learning framework for recommendations of long-tail web services," *IEEE Trans. Services Comput.*, vol. 13, no. 1, pp. 73–85, Jan. 2020.

[23] H. Zhao, J. Wang, Q. Zhou, X. Wang, and H. Wu, "Web API recommendation with features ensemble and learning-to-rank," in *Proc. CCF Conf. Big Data Comput. Appl. Services (BigDataCAS)*, Wuhan, China, Sep. 2019, pp. 26–28.

[24] 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.

[25] J. Zhang and Y. Fan, "Recommending collaborations with newly emerged services for composition creation in cloud manufacturing," *Int. J. Comput. Integr. Manuf.*, vol. 34, no. 3, pp. 307–326, Mar. 2021.

[26] 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.

[27] 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.

[28] H. Yu, C. J. Woodard, and S. S. Sim, "Innovation in the programmable web: Characterizing the mashup ecosystem," in *Proc. Int. Conf. Service-Oriented Comput. (ICSOC)*, Sydney, NSW, Australia, Dec. 2008, pp. 1–10.
[29] K. Benouaret, R. Valliyur-Ramalingam, and F. Charoy, "Web service composition: A survey of techniques and tools," *ACM Comput. Surveys*, vol. 48, no. 3, pp. 1–41, 2013.
[30] Y. Dong, N. V. Chawla, and A. Swami, "Metapath2vec: Scalable representation learning for heterogeneous networks," in *Proc. 23rd ACM SIGKDD Int. Conf. Knowl. Discov. Data Min.*, Halifax, NS, Canada, Aug. 2017, pp. 135–144.
[31] K. Chen, H. Mao, X. Shi, Y. Xu, and A. Liu, "Trust-aware and location-based collaborative filtering for web service QoS prediction," in *Proc. IEEE 41st Annu. Comput. Softw. Appl. Conf. (COMPSAC)*, vol. 2, Turin, Italy, Jul. 2017, pp. 143–148.
[32] 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.
[33] C. Li, L. Zheng, S. Wang, F. Huang, P. S. Yu, and Z. Li, "Multi-hot compact network embedding," in *Proc. 28th ACM Int. Conf. Inf. Knowl. Manage.*, Beijing, China, Nov. 2019, pp. 459–468.
[34] X. Wang, H. Ji, C. Shi, B. Wang, Y. Ye, P. Cui, and P. S. Yu, "Heterogeneous graph attention network," in *Proc. WWW*, 2019, pp. 2022–2032.
[35] Z. Wang, J. Zhang, J. Feng, and Z. Chen, "Knowledge graph embedding by translating on hyperplanes," in *Proc. 28th AAAI Conf. Artif. Intell. (AAAI)*, Montreal, QC, Canada, 2014, pp. 1112–1119.
[36] A. Bordes, N. Usunier, A. Garcia-Duran, J. Weston, and O. Yakhnenko, "Translating embeddings for modeling multi-relational data," in *Proc. Adv. Neural Inf. Process. Syst. (NeurIPS)*, Lake Tahoe, NV, USA, Dec. 2013, pp. 5–10.
[37] S. Rendle, C. Freudenthaler, Z. Gantner, and L. Schmidt-Thieme, "BPR: Bayesian personalized ranking from implicit feedback," in *Proc. 25th Conf. Uncertainty Artif. Intell. (UAI)*, Montreal, QC, Canada, Jun. 2009, pp. 18–21.
[38] T. Ramathulasi and M. R. Babu, "Enhanced PMF model to predict user interest for web API recommendation," in *Handbook of Research on Advances in Data Analytics and Complex Communication Networks*. Pennsylvania, PA, USA: IGI Global, 2022, pp. 131–139.
[39] Y. Cao, J. Liu, M. Shi, B. Cao, T. Chen, and Y. Wen, "Service recommendation based on attentional factorization machine," in *Proc. IEEE Int. Conf. Serv. Comput. (SCC)*, Milan, Italy, Jul. 2019, pp. 8–13.
[40] Y. Zhang, H. Yang, and L. Kuang, "A web API recommendation method with composition relationship based on GCN," in *Proc. ISPA*, Exeter, U.K., 2020, pp. 17–19.
[41] H. Wang, F. Zhang, M. Zhang, J. Leskovec, M. Zhao, W. Li, and Z. Wang, "Knowledge-aware graph neural networks with label smoothness regularization for recommender systems," in *Proc. KDD*, 2019, pp. 968–977.
[42] Z. Chen, T. Zhang, and X. Peng, "A novel API recommendation approach by using graph attention network," in *Proc. QRS*, Hainan, China, 2021, pp. 6–10.

LIMIN SHEN received the B.S. and Ph.D. degrees in computer science and technology from Yanshan University, China. He is currently a Professor and the Ph.D. Supervisor with the College of Computer Science and Engineering, Yanshan University. His research interests include service computing, collaborative computing, and cooperative defense.

YUYING WANG received the M.S. degree from the Institute of Electrical Engineering, Yanshan University, Qinhuangdao, China, in 2018, where she is currently pursuing the Ph.D. degree in information science and engineering. Her research interests include cloud API recommendation and complementary recommender systems.

CHENGYU LI is currently pursuing the master's degree with the College of Information Science and Engineering, Yanshan University. His current research interests include recommenders and service computing.

ZHEN CHEN received the B.S. and Ph.D. degrees in computer science and technology from Yanshan University, China, in 2010 and 2017, respectively. He is currently an Associate Professor with the College of Computer Science and Engineering, Yanshan University. He is also a Postdoctoral Researcher with the College of Information Science and Engineering, Yanshan University. He is also working on service computing, cloud computing, and collaborative computing.