electronics

MDPI

Article

Similarity and Complementarity Attention-Based Graph Neural Networks for Mashup-Oriented Cloud API Recommendation

Limin Shen [1,2], Yuying Wang [1,2,*], Shuai Zhang [1,2] and Zhen Chen [1,2]

[1] School of Information Science and Engineering, Yanshan University, Qinhuangdao 066004, China; shenllmm@ysu.edu.cn (L.S.); 202222040218@stumail.ysu.edu.cn (S.Z.); zhenchen@ysu.edu.cn (Z.C.)
[2] The Key Laboratory for Software Engineering of Hebei Province, Yanshan University, Qinhuangdao 066004, China
* Correspondence: yuyingwang@stumail.ysu.edu.cn

Abstract: Mashups, which combine various web application programming interfaces (APIs) to implement some complex requirements, have grown to be a popular technique for developing service-oriented software. However, recommending suitable cloud APIs for mashup creation is challenging due to the rapidly increasing number of comparable APIs. Many existing mashup-oriented cloud API recommendations focus on functional similarity and ignore functional complementarity, which significantly impacts the accuracy of the recommendation results. Therefore, this paper proposed a feature representation and recommendation method for cloud APIs that fuses both similarity and complementarity. A heterogeneous information network of the cloud API ecosystem was constructed, and the neighbors, based on metapaths, were aggregated using a self-attention mechanism to generate the features of similarity and complementarity for the cloud APIs. Then, the mashup-related attention was utilized to fuse the two features, taking into consideration the varying preferences of different mashups towards similarity and complementarity features of cloud APIs. This fusion resulted in features that align cloud APIs with mashup requirements, which were employed to predict the probability of the mashup invoking a particular candidate cloud API. The proposed method was evaluated on a real dataset, and the results showed that it outperforms the baseline method and enhances the performance of mashup-oriented cloud API recommendations.

Keywords: cloud API recommendation; mashup; attention mechanism; metapath; functional similarity; functional complementarity

check for updates

Citation: Shen, L.; Wang, Y.; Zhang, S.; Chen, Z. Similarity and Complementarity Attention-Based Graph Neural Networks for Mashup-Oriented Cloud API Recommendation. *Electronics* **2023**, *12*, 4436. https://doi.org/10.3390/electronics12214436

Academic Editor: Manuel Mazzara

Received: 21 September 2023
Revised: 20 October 2023
Accepted: 26 October 2023
Published: 27 October 2023

Copyright: © 2023 by the authors. Licensee MDPI, Basel, Switzerland. This article is an open access article distributed under the terms and conditions of the Creative Commons Attribution (CC BY) license (https://creativecommons.org/licenses/by/4.0/).

1. Introduction

Along with the rapid development of cloud computing, application systems are moving towards openness, and the number of cloud APIs available on the network is rapidly expanding. As the carrier for service delivery, capability replication, and data output, the cloud application programming interface (API) has evolved into the necessary core element for service-oriented software development and operation [1]. As a lightweight application, mashup accelerates the development process and improves application scalability by reusing and combining the functionality of multiple cloud APIs to fulfill complex requirements that cannot be achieved with a single cloud API. By combining cloud APIs, data, and resources from the web to assemble a single application that is more useful to an organization or individual, mashups can meet personalized needs quickly and cost-effectively [2].

However, the continuously growing number of cloud APIs has brought developers challenges in making choices. Finding cloud APIs that meet specific requirements is an indispensable task for successful mashup development. It is essential to rapidly locate one or more cloud APIs that fit the criteria to promote the widespread adoption and advancement of cloud APIs. For example, ProgrammableWeb is a platform that boasts a

vast collection of over 20,000 cloud APIs, which are categorized into 500 different categories. While this abundance of options can be a boon for developers, it can also be overwhelming when it comes to choosing the most suitable APIs for their mashup projects. Therefore, the cloud API recommendation system has become a natural choice for developers to recommend suitable cloud APIs.

In recent years, various researchers have devoted their attention to addressing the challenge of enhancing the efficiency of finding cloud APIs, and have proposed several recommendation methods for assisting mashup developers in selecting suitable cloud APIs expeditiously. Developers' requirements are typically used as inputs for mashup-oriented cloud API recommendations. The recommendation system then calculates a match score for each cloud API in the repository and provides a list of recommended cloud APIs to the developer, sorted based on the match score [3]. The two primary categories of the existing mashup-oriented cloud API recommendation methods are as follows: (1) Recommendation focusing on collaborative filtering (CF) [4–9], which utilizes the implicit mashup–API interaction information to discover mashup interest by similarity computation to complete the recommendation. However, the CF approach usually requires a large number of interaction records that are not available when using a new API or creating a new mashup, and therefore other content data need to be utilized for which the CF approach is not applicable. (2) Recommendation focusing on similar content [10–15], which employs information retrieval technology to recommend high-matching cloud APIs by analyzing and calculating the similarity between requirements and cloud API functions. These methods can recommend cloud APIs that are close to the needs of mashups, but do not pay enough attention to the collaborative relationship between cloud APIs and mashups.

Although existing research has actively investigated cloud API recommendation, there are still some issues that limit the effectiveness of mashup-oriented cloud API recommendation:

1. Many current methods focus on finding similar cloud APIs without taking into account the ability of functional complementarity between them. However, cloud APIs that have very similar functionality are more likely to be replaced than used in combination [16,17]. APIs that are chosen to be used together often need to be functionally complementary to fulfill complex functional requirements. Therefore, to enhance the effectiveness of mashup-oriented cloud API recommendations, not only should we consider APIs with similar functions, but we should also consider whether the selected set of APIs can achieve complementary functions;
2. Different mashups place varying emphasis on the functional similarity and complementarity of cloud APIs. Some mashups require the use of multiple cloud APIs that offer similar functions. For instance, a payment mashup may require the integration of payment cloud APIs from different platforms to provide a more comprehensive payment function. On the other hand, some mashups require the use of multiple cloud APIs that offer different functions. For example, an e-commerce mashup may need to integrate cloud APIs that offer functions such as maps, payments, and socialization to meet complex requirements. Unfortunately, some current cloud API recommendation algorithms fail to consider the different concerns of mashups regarding similarity and complementarity. This oversight can significantly affect the accuracy and comprehensiveness of cloud API recommendations.

To address the challenges faced by the above problems, a mashup-oriented cloud API recommendation method is proposed. The goal of this paper is to recommend a set of cloud APIs for a target mashup on a cloud API service platform based on the mashup requirements presented by the developers. The cloud API ecosystem accumulates a large amount of multi-type knowledge that can be used as side information to enhance recommendation models. For example, cloud API publishing and sharing platforms such as ProgrammableWeb store a large number of objects related to cloud APIs and the relationships between those objects. Based on the side information, a heterogeneous graph is first constructed by taking mashups, cloud APIs, and function tags as nodes, and the

invocation relationship between the mashup and cloud APIs and the labeling relationship between APIs and function tags as the relation.

Cloud API function tags are used to describe the functionality provided by a cloud API, and two cloud APIs with one or more of the same function tags are considered to have functional similarities. Mashup invocations of cloud APIs describe the ability of two cloud APIs to collaborate with each other, and two different types of cloud APIs that have been invoked together by the same mashup are considered more likely to have complementary functionality. Both the similarity and complementarity of cloud APIs can be useful in mashup creation. Some mashups specialize in a specific functionality and often require multiple similar cloud APIs that can provide that functionality. For instance, a mashup can combine multiple music APIs to achieve a more comprehensive service. Some mashups need to invoke complementary cloud APIs for different functionalities that can be used collaboratively to complement each other's functionality. For instance, for about two-thirds of the mashups that invoke "yahoo local search", APIs also invoke the "google maps" API, such as the mashup "bigsity", which provides local services, so the "google maps" API may have some functional complementarity with the "yahoo local search" API. Cloud APIs with similarity and complementarity are detected using the designed similarity metapaths and complementarity metapaths, respectively, in the heterogeneous graph and are called similarity neighbors and complementarity neighbors of each other. Similarity feature and complementarity feature representations of cloud APIs are obtained by aggregating the features of cloud API neighbors based on different metapaths.

Considering the potential preferences of mashups with different requirements for similarity and complementarity, mashup-related attention is utilized to identify the different importance of cloud API similarity features and complementarity features for the given mashup. Then, the cloud API similarity features and complementarity features are fused with the attention to compute a representation of the cloud API features that are aligned with the mashup's requirements. Finally, the probability that each candidate cloud API is expected to be invoked by a mashup is predicted using the mashup requirements and its requirement-related cloud API features, which achieves cloud API recommendations aligned with the mashup's requirements, leading to an improvement in the effectiveness of the mashup-oriented cloud API recommendation.

In summary, the main contributions of this paper are as follows:

1. We propose a recommendation method that takes into account both the similarity and complementarity of cloud APIs and leverages the rich side information in the cloud API ecosystem to enhance the feature representation of cloud APIs.
2. We construct a heterogeneous graph and use a graph attention neural network to aggregate similarity neighbors and complementarity neighbors based on the metapaths of cloud APIs to obtain similarity features and complementarity features.
3. We consider mashups' different attention to similarity and complementarity, and utilize mashup-related attention to fuse cloud API similarity features and complementarity features with preference to obtain feature representations of cloud APIs that are closely related to mashup, and use them to recommend a set of APIs that satisfy the requirements for a mashup.

The remainder of this paper is organized as follows. In Section 2, the related work is reviewed and synthesized. In Section 3, the motivation and some relevant definitions for the study are given, and then the proposed problem is formally described. In Section 4, the proposed method is described in detail. In Section 5, the experimental evaluation and analysis are given. Finally, Section 6 summarizes the conclusions of this paper.

2. Related Work

Cloud API recommendation technology is essential to service-oriented computing and can significantly raise the quality of service discovery [18]. Many recommendations have been suggested in recent years to assist software developers in swiftly and effectively identifying suitable cloud APIs. Mashup-oriented cloud API recommendations can be

broadly divided into three groups: recommendations based on collaborative filtering, recommendations based on content, and recommendations based on knowledge graphs. This section provides a brief summary of the related work.

The collaborative filtering-based methods make utilization of user-API or mashup-API interaction matrices to discover user preferences through similarity calculations and produce recommendations. A deep matrix factorization framework was introduced by Xue et al. [6] and can directly extract features from the user–API interaction matrix and fully take into account explicit and implicit ratings to achieve top-K recommendations. Fletcher et al. [7] provided a regularized user preference embedding a matrix factorization approach that considers users' explicit and implicit personalized preferences, significantly enhancing the precision and variety of suggestions. Tang et al. [18] suggested a technique that takes advantage of neural graph collaborative filtering technology and high-order connectivity between cloud APIs and users. The importance of similarity calculations in collaborative filtering algorithms is widely established. Some researchers have added further information to improve the accuracy of similarity calculations and increase the impact of the collaborative filtering (CF) model. For instance, Chen et al. [8] took the user's credit and the spatial environment into account. Quality of service (QoS) characterizes the quality information of a cloud API in a certain aspect, such as response time, throughput, availability, and packet loss, which is generally used to describe the user's non-functional requirements for a cloud API. Additionally, Meng et al. [9] took into account temporal impacts by making a distinction between stable QoS measurements based on the traditional neighborhood-based CF model and temporal QoS metrics. However, the methods based on collaborative filtering have a high dependence on the interaction matrix, the quantity and quality of historical interaction information have a serious impact on the recommendation performance, and the cold start problem is serious.

The content-based recommendation mainly focuses on the functional matching of cloud APIs with developers' requirements [12,13]. Xia et al. [17] proposed a category-aware cloud API clustering method that first predicts user-need-based cloud API categories and then scores cloud APIs in each category for recommendation. Shi et al. [14] suggested a text extension and a deep model-based approach to cloud API recommendation. By using a hierarchical probabilistic topic model to enhance sentence-level cloud API descriptions, this approach then proposes cloud APIs based on semantic similarity. Zhong et al. [15] employ author topic models to model mashup descriptions and component cloud APIs, and they rebuild cloud API profiles by using mashup descriptions and structures to find significant lexical features of cloud APIs. A new cloud API recommendation algorithm is suggested using the word features in the cloud API profile that was rebuilt. By combining textual features, closest neighbor features, API-specific characteristics, tag features from mashups, and cloud APIs, Zhao et al. [13] suggested a method for recommending cloud APIs that used features ensemble and learning-to-rank. These content-based approaches recommend cloud APIs that are more closely aligned with the developer's requirements but often ignore whether these cloud APIs are suitable for combining into mashups.

The knowledge graph-based approach makes extensive use of attribute and interaction data from the cloud API ecosystem for cloud API recommendation [19,20]. Geng et al. [21] built a knowledge graph of cloud APIs using additional data such as cloud API categories and developers. They then used Doc2Vec to mine the semantics of cloud APIs describing documents and graph convolution networks to mine higher-order relationships between cloud APIs and mashup preferences to predict the cloud APIs necessary for mashup development. In order to improve recommendation models, Wang et al. [22] made use of the wealth of knowledge in the cloud API ecosystem, created a clear knowledge graph schema to encode mashup-specific context, modeled mashup requirements with graph entities, and used random walks with restarts. Knowledge graphs are used to look for possible correlations between mashup requirements and cloud APIs. Based on this, Wang et al. [23] went on to further utilize the mashup-API common call and cloud API category attributes to construct a refined knowledge graph and used the deep random walk of the knowledge

graph for unsupervised cloud API recommendation. Gao et al. [24] propose a multi-relational graph neural network model that merges functional and labeling relationships between mashups and services into combination relationships and merges features of higher-order neighbors through graph convolution.

Inspired by the above work and considering the specificity of the mashup-oriented recommendation cloud API, we propose a new cloud API recommendation method that constructs heterogeneous information networks (HINs) with cloud API attribute information and interaction information to enhance cloud API feature representation. The similarity features and complementarity features of the cloud API are individually represented and fused to make the recommendation results more suitable for mashup requirements.

3. Background

3.1. Motivation

The problem to be solved is how to recommend the most suitable set of cloud APIs on a cloud API service platform (e.g., ProgrammableWeb) for the target mashup based on the mashup requirements presented by the developer. By observation of cloud API ecosystem data, we find that different mashups tend to have different concerns about cloud API functional similarity and functional complementarity.

In detail, some mashups are more focused on a specific function and are usually more likely to use multiple cloud APIs for that specific function concurrently. For example, according to the real cloud API data based on ProgrammableWeb, some mashups in the map category may need to call multiple map APIs, such as Google Maps, Google Earth, Bing Maps, etc., and integrate them. The actual data show that 96% of the mashups in the category of maps call cloud APIs with map tags, of which 47% call multiple map cloud APIs simultaneously. Therefore, the functional similarity of the APIs needs to be given more attention when creating such mashups. While some other mashups need to call multiple different categories of cloud APIs that can be used collaboratively, for example, a mashup that provides travel services needs photo, map, social, and other categories of APIs to complement each other to realize mutual supplementation of functionality. The complementary performance of the multiple cloud APIs is critical in creating this type of mashup, and the data show that, among the mashups that belong to this category and call more than two APIs, only 10% of mashups called only single category cloud APIs, while 41% of mashups called both the photo and map categories, and 28% of mashups called both the social and map categories. In addition, some mashups with more complex and diverse functions, such as e-commerce mashups, not only need multi-functional APIs for online payment, transportation, data storage, etc., but also need to integrate APIs provided by multiple different payment platforms and transportation service platforms to realize comprehensive services. Obviously, for such mashups, neither the similarity nor complementarity of the cloud API functions can be ignored.

Hence, it can be seen that equal treatment of cloud API similarity and complementarity features indicated by different types of side information will lead to the neglect of some potential preference information of mashups. To address this problem, this study introduces an attention mechanism to learn the preference scores of different mashups for cloud API similarity and complementarity and adds the mashup's preference information to the cloud API's feature representation. This enables the feature representation of cloud APIs to better fit the requirements of the mashups to be developed and improves the performance of mashup-oriented recommendations.

3.2. Definition

In order to better state the proposed problem and its solution in this paper, some definitions related to heterogeneous graphs are given below.

Definition 1 (Heterogeneous graph)**.** *A heterogeneous graph is defined as a graph* $\mathcal{G} = (\mathcal{V}, \mathcal{E})$ *associated with a node-type mapping function* $\phi : \mathcal{V} \rightarrow \mathcal{T}^{\mathcal{V}}$ *and an relation-type mapping function*

$\psi : \mathcal{E} \rightarrow \mathcal{T}^{\mathcal{E}}$. $\mathcal{T}^{\mathcal{V}}$ *and* $\mathcal{T}^{\mathcal{E}}$ *denote the sets of node types and relation types, respectively, with* $|\mathcal{T}^{\mathcal{V}}| + |\mathcal{T}^{\mathcal{E}}| \geq 2$.

Definition 2 (Metapath). *A metapath P is defined as a path in the form of* $\mathcal{T}_1^{\mathcal{V}} \xrightarrow{\psi_1} \mathcal{T}_2^{\mathcal{V}} \xrightarrow{\psi_2} \cdots \xrightarrow{\psi_{l-1}} \mathcal{T}_l^{\mathcal{V}}$, *which describes a composite relation* $\psi_P = \psi_1 \circ \psi_2 \circ \cdots \circ \psi_{l-1}$ *between node types* $\mathcal{T}_1^{\mathcal{V}}$ *and* $\mathcal{T}_l^{\mathcal{V}}$, *where* $\circ$ *denotes the composition operator on relations.*

Definition 3 (Metapath-based neighbor). *Given a node v and a metapath P of a heterogeneous graph* $\mathcal{G}$, *the metapath-based neighbor* $\mathcal{N}_v^P$ *is defined as the set of nodes that connect with node v through metapath P. Note that* $\mathcal{N}_v^P$ *includes v itself if P is symmetric.*

3.3. Problem Formulation

The core task of recommending cloud APIs for developers based on their requirements is to solve the problem of predicting the probability that a mashup may call cloud APIs. Therefore, a heterogeneous cloud API ecosystem graph is constructed, such as $\mathcal{G} = (\mathcal{V}, \mathcal{E})$, where the node set $\mathcal{V}$ contains the cloud API node set $A = \{a_1, a_2, \cdots, a_{|A|}\}$, the mashup node set $M = \{m_1, m_2, \cdots, m_{|M|}\}$, and the function tag node set $T = \{t_1, t_2, \cdots, t_{|T|}\}$ and the relationship set $\mathcal{E}$ is a set of multiple relationships between the different types of nodes listed above. The invocation relationship between mashups and cloud APIs can be defined as matrix $Y \in {}^{|M| \times |A|}$, where $y_{m,a}$ represents whether the cloud API is invoked by mashup in binary form. If cloud API a is invoked by mashup m, then $y_{m,a}$ equals 1; otherwise, it equals 0. The set $Treq_m = \{treq_1, treq_2, \cdots, treq_n\} \in T$ consisting of multiple function tags given by the developer for describing the requirements of the mashup is used to represent the features of the mashup m to be developed.

The probability that a given mashup m with specific requirements may invoke cloud API a is predicted using the above mashup–cloud API interaction matrix Y and heterogeneous cloud API ecosystem graph $\mathcal{G}$:

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

where $\hat{y}_{m,a}$ denotes the probability that mashup m will invoke cloud API a that has not been noticed, and Θ is the parameters of the model. On this basis, the top-N cloud APIs with the highest probability of being invoked by mashup m are selected as the recommendation results of the mashup.

4. Method

It is crucial to comprehensively and accurately describe the feature representation of cloud APIs in order to solve the problem of how to recommend the most appropriate cloud APIs to develop a mashup with a given requirement. In this study, a graph neural network algorithm based on a metapath and attention mechanism is used that comprehensively takes into account the functional similarities and functional complementarity of cloud APIs as well as the requirements of mashups to obtain a mashup requirements-aligned representation of cloud APIs that can be used to predict the probability of cloud API invocation by the target mashup. The overall framework of the proposed mashup-oriented cloud API recommendation method is given in Figure 1. In the figure, $a_1, a_2 \ldots a_6$ are the API nodes, m_1, m_2 are the mashup nodes, and t_1, t_2 are the tag nodes used to describe the functions of APIs. α^{sim} and α^{comp} denote the aggregation attention coefficients in the similarity metapath and complementarity metapath, respectively, and β_m denotes the inter-metapath fusion attention coefficient. $\hat{y}_{m,a}$ is the probability that mashup m will invoke cloud API a_1. The method proposed consists of three parts:

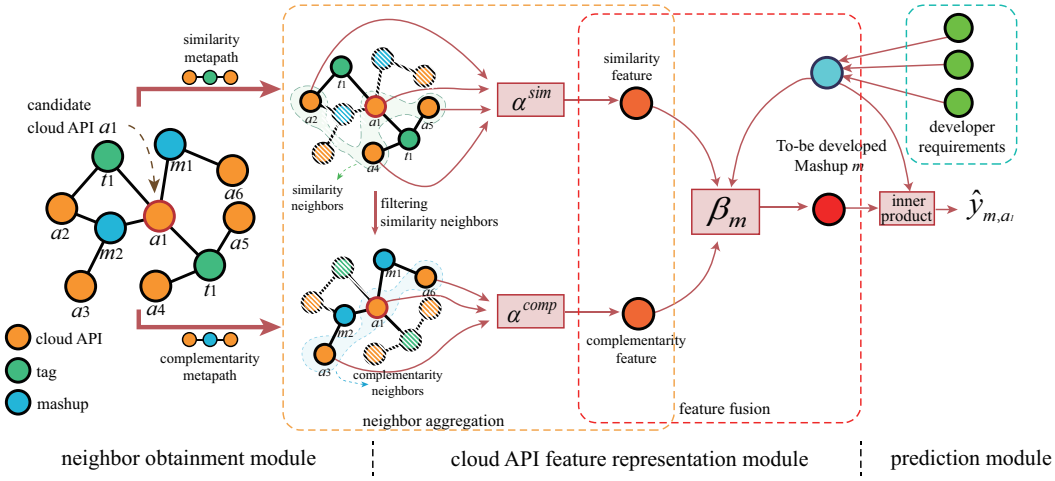


Figure 1. Overall framework of the proposed mashup-oriented cloud API recommendation method.

1. In the neighbor obtainment module, similarity neighbors and complementarity neighbors of the cloud API are obtained by given metapaths, and two independent subgraphs are generated accordingly;
2. In the cloud API feature formation module, cloud API features are first initialized by utilizing the functional description information. Then, similarity neighbor features and complementarity neighbor features of cloud APIs are aggregated in two subgraphs. After that, the similarity features and complementarity features are fused by using the attention related to the mashup requirements provided by the developer to obtain a cloud API feature representation related to the to-be-developed mashup;
3. The prediction and recommendation module calculates the probability of candidate cloud APIs being invoked by the mashup with a given requirement, and the top-N cloud APIs with the highest probability are recommended to developers for mashup construction.

4.1. Neighbors Obtainment

The cloud API ecosystem accumulates a large amount of functional description information and interaction information about cloud APIs, mashups, etc. Making full use of this rich side information can effectively improve the accuracy and comprehensiveness of recommendations.

Therefore, a heterogeneous cloud API ecosystem graph, as in Definition 1, is constructed using data from the cloud API ecosystem. As shown in Figure 1, the heterogeneous graph contains three types of nodes: cloud API, tag, and mashup. The labeling relationship between cloud API nodes and tag nodes, and the invocation relationship between cloud API nodes and mashup nodes are introduced to characterize the cloud APIs.

In heterogeneous graphs, metapaths can be used to analyze complex information, and different metapaths reveal different semantic features. In order to characterize cloud API similarity features and complementary features, respectively, the following two meta-paths are designed. And, according to Definition 3, the neighbors of cloud APIs with different semantic features under different meta-paths are obtained.

The similarity metapath 'API $\overset{\text{be labeled}}{\rightarrow}$ tag $\overset{\text{label}}{\rightarrow}$ API' connects two APIs with certain similar functionality because both APIs have the same functional tag. Given a candidate cloud API, all the nodes in the graph that are connected to this node through instances of this metapath via intermediate nodes of type 'tag' are called similarity neighbors of the given cloud API node. For example, 'yahoo-weather(API)' is a similarity neighbor of 'weather-channel(API)', and they connect via the metapath instance 'weather-channel $\rightarrow$ weather $\rightarrow$ yahoo-weather'.

The complementarity metapath 'API $\overset{\text{be invoked}}{\rightarrow}$ mashup $\overset{\text{invoke}}{\rightarrow}$ API' is used to indicate that two different categories of APIs have been invoked by the same mashup and that

the two APIs may have certain complementary functions. The nodes that are connected to the candidate cloud API node through this metapath instance via the 'mashup' type intermediate nodes are called the complementarity neighbors of the candidate cloud API node. It should be noted that multiple cloud APIs invoked by the same mashup may also be similar cloud APIs used for refining a single functional requirement, which is obviously not suitable for describing complementarity features. Therefore, the complementarity neighbors' obtainment needs to filter out the nodes that are identified as similarity neighbors other than itself beforehand.

Taking the node in Figure 1 as an example, the similarity neighbors of the candidate node based on the similarity metapath 'cloud API → tags → cloud API' include a_1 (itself), a_2, a_4, and a_5, and the complementarity neighbors of candidate node A1 based on the complementarity metapath 'cloud API → mashup → cloud API' include a_1 (itself), a_3, and a_6. Obviously, neighbors based on different metapaths can utilize different structural information in the heterogeneous graph. Neighbors based on metapaths can be obtained by multiplication of sequences of adjacency matrices.

4.2. Feature Formation

The accurate, comprehensive, and purposeful characterization of cloud API features is the basis for achieving mashup-oriented cloud API recommendations, and an illustration of mashup requirements-aligned cloud API feature formation based on the graph neural network is given in Figure 2. First, the cloud API feature representation vector is initialized using the functional description information of the cloud API, and a set of feature tags for describing the requirements given by the developer is used to vectorize the features of the mashup to be developed. Then, the neighbor features of cloud APIs are aggregated independently on different metapaths. Since neighbor nodes do not have exactly the same importance as candidate cloud API nodes, the self-attention mechanism is used to discover the differences between neighbors and learn the weights. Considering that different mashups have different preferences for similarity and complementarity, as mentioned earlier, attention related to the mashup is used to model the preference information of a given mashup to be developed. The biased fusion of cloud API similarity features and complementarity features achieves the integration of different semantic information in the graph and obtains cloud API features that align with the requirements proposed by mashup developers.

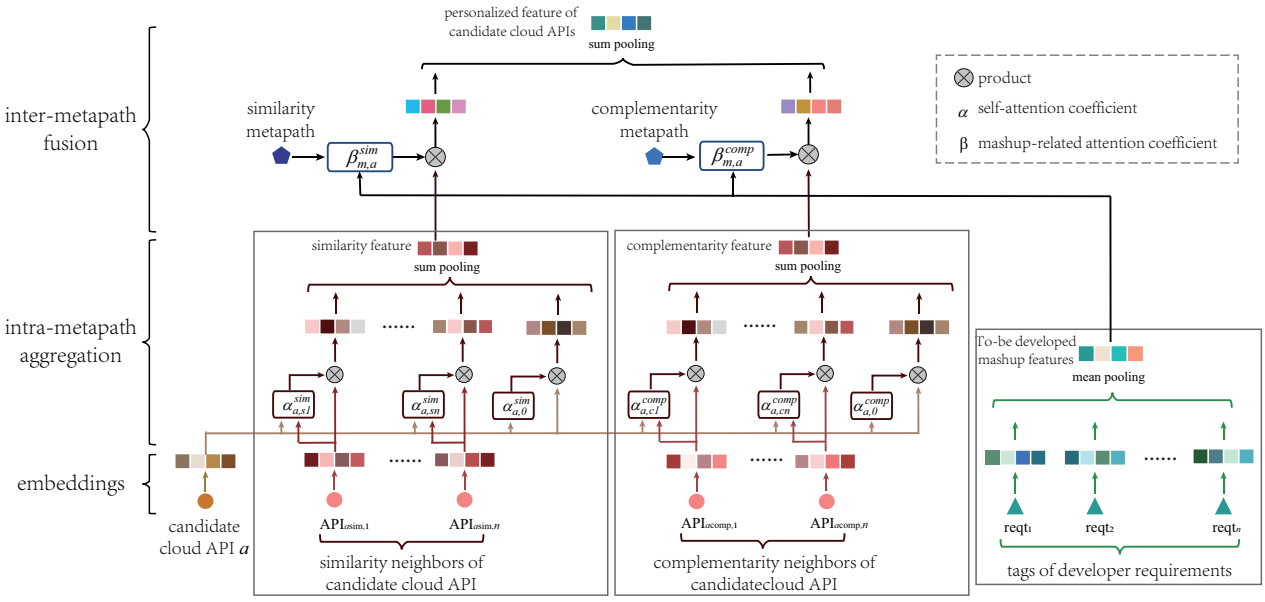


Figure 2. Illustration of mashup requirements-aligned cloud API feature formation based on the graph neural network.

4.2.1. Mashup Requirements Representation

In this study, the features of a mashup are represented by a set of feature tags given by the developer for describing the requirements, e.g., the tags (social, e-commerce, reference) are used as inputs to the cloud API recommendation system to characterize the mashup of the relevant combination functions to be developed. Utilizing tags to describe items has been considered successful in numerous information systems, and this also applies to the cloud API ecosystem [22]. In the cloud API ecosystem, the combination of multiple different tags can uniquely characterize mashup requirements, enabling the characterization of individual mashup requirements with fewer parameters. It also avoids the problem of vocabulary variance caused by developers' free use of vocabulary and text describing requirements without predetermined standards [14].

Specifically, according to the tags set T in the cloud API ecosystem, the query information consisting of several tags that characterize the mashup function is given by the developer. For example, in order to find suitable sets of candidate cloud APIs to develop mashups m_1 and m_2, the required feature set $Treq_{m_1} = \left\{t_2, t_{|T|-1}, t_{|T|}\right\} \in T$ and $Treq_{m_2} = \left\{t_1, t_{|T|}\right\} \in T$ should be given. The multi-hot embedding is utilized to represent mashups m_1 and m_2 as follow:

$$\begin{array}{rccccccc} & \{ & t_1, & t_2, & \cdots, & t_{|T|-1}, & t_{|T|} & \} \\ m_1 = & (& 0, & 1, & \cdots, & 1, & 1 &) \\ m_2 = & (& 1, & 0, & \cdots, & 0, & 1 &). \end{array} \tag{2}$$

Then, the embedding technique $f : T \to \mathbb{R}^d$ is used to map all tags to a low dimensional dense space, which is suitable for deep neural network training and learning, where d is the embedding dimension. Finally, to fully describe the requirement information, the mashup features are represented by aggregated tag embeddings, such as follows:

$$\begin{aligned} e_{m_1} &= g(e_{t_2}, e_{t_{|T|-1}}, e_{t_{|T|}}) \\ e_{m_2} &= g(e_{t_1}, e_{t_{|T|}}) \end{aligned} \tag{3}$$

where e_{t_i} is the embedding of tag t_i, and $g(\cdot)$ is the operation function used to aggregate tag embeddings. In our experiments, we assume that each tag is equally important to the mashup requirements; therefore, $g(\cdot)$ is chosen as the averaging operation.

4.2.2. Intra-Metapath Aggregation

In the neighbor aggregation phase, the heterogeneous cloud API ecosystem graph is split into two homogeneous subgraphs according to the similarity and complementarity neighbors of the candidate cloud API nodes based on the metapaths. A graph attention neural network (GAT) [25] is used to aggregate similarity neighbor features and complementarity neighbor features in different subgraphs, respectively. The GAT introduces an attention mechanism to achieve better neighbor aggregation. By learning the weights of neighbors, GAT can achieve weighted aggregation of neighbors. Therefore, GAT is not only more robust to noisy neighbors, but the attention mechanism also endows the model with a certain degree of interpretability, so that we can understand the contribution of each neighbor. To be specific, assume that the metapath-based subgraph has n cloud API nodes, and the initial feature set of nodes is $\mathbf{h} = \{\mathbf{h_1}, \mathbf{h_2}, \ldots, \mathbf{h_n}\}, \mathbf{h_i} \in \mathbb{R}^d$. Given a metapath P, a candidate cloud API node a, and metapath-based neighbor node $j \in \mathcal{N}_a^P$, the importance of node j to node a can be obtained by the attention coefficient $\omega_{a,j}^P$. The calculation formula is as follows:

$$\omega_{a,j}^P = \sigma_1(\mathbf{q}_P^T(\mathbf{W}\mathbf{h}_a \| \mathbf{W}\mathbf{h}_j)) \tag{4}$$

where $\|$ means concatenation operation, $\mathbf{W} \in \mathbb{R}^{\mathbf{d}*\mathbf{d}}$ is the shared linear transformation weight matrix applied at each node, $\mathbf{q}_P \in \mathbb{R}^{2d}$ is the attention vector for metapath P, and $\sigma_1(\cdot)$ denotes the activation function, such as LeakyReLU.

According to the graph structure information, the mask attention mechanism is used to filter out nodes that are not neighbors of the target cloud API, and only calculate the attention coefficient $\omega_{a,j}^{P}$ of the metapath-based neighbor $j \in \mathcal{N}_a^P$ to the target cloud API node a. In order to better distribute weights among different nodes, we need to normalize the attention coefficients via the softmax function:

$$\alpha_{a,j}^{P} = Softmax_j(\omega_{a,j}^{P}) = \frac{\exp(\omega_{a,j}^{P})}{\sum_{k \in \mathcal{N}_a^P} \exp(\omega_{a,k}^{P})} \tag{5}$$

It should be noted that $\alpha_{a,j} \neq \alpha_{j,a}$, since they have different orders when using the masked attention mechanism to calculate the attention score. Finally, the normalized attention coefficients are linearly combined with their corresponding features, and the final aggregated output of the cloud API node on the metapath-based subgraph is obtained as follows:

$$\mathbf{z}_a^P = \sigma_2(\sum_{j \in \mathcal{N}_a^P} \alpha_{a,j}^{P} \mathbf{W}\mathbf{h}_a) \tag{6}$$

To sum up, given the cloud API initial feature vector and metapath set $\Phi = \{P_{sim}, P_{comp}\}$, where P_{sim} denotes similarity metapath and P_{comp} denotes complementarity metapath, aggregating the information in each metapath separately and independently, the feature representations set $\{\mathbf{z}_a^{sim}, \mathbf{z}_a^{comp}\}$ of cloud API node a under each metapath can be obtained.

4.2.3. Inter-Metapath Fusion

After obtaining similarity features and complementarity features of cloud API nodes on different metapaths, these two features need to be fused to integrate different semantic information in the heterogeneous cloud API ecosystem graph and achieve a comprehensive representation of cloud APIs. One of the most straightforward metapath fusion methods is to take the element-wise mean of these semantic feature vectors. However, we observed that different mashups pay different attention to similarity and complementarity; that is to say, for different mashups, different metapaths have different importance in the heterogeneous graph of the cloud API ecosystem.

Therefore, in order to achieve a mashup-oriented cloud API recommendation, mashup-metapath importance scores are learned in the feature fusion phase and are used to biasedly fuse cloud API features from different metapaths. Given a metapath P, its importance to mashup m can be calculated by the following equation:

$$\mu_m^P = f(\mathbf{e}_m, \mathbf{e}_P) \tag{7}$$

where $\mathbf{e}_m$, $\mathbf{e}_P$ are the representations of mashup and metapath, respectively. Function $f(\cdot)$ is used to calculate the score, and the inner product is used in our experiments.

After obtaining the importance score of each metapath to mashup m, the attention β_P^m of the metapath P to mashup m can be obtained by normalizing the above importance of all metapaths using the softmax function:

$$\beta_m^P = Softmax_p(\mu_m^P) = \frac{\exp(\mu_m^P)}{\sum_{\rho \in \Phi} \exp(\mu_m^{\rho})} \tag{8}$$

In order to obtain feature embedding for cloud API node a aligned with mashup m requirements, the above attention scores on metapaths are used as filters when computing the feature representation of candidate cloud API nodes. The final feature embedding of the candidate cloud API node a about a specific given requirement mashup m is obtained by linear combination:

$$\mathbf{e}_a^m = \sum_{P \in \Phi} \beta_m^P \mathbf{z}_a^P \tag{9}$$

4.3. Prediction and Recommendation

According to the above section, the cloud API personalized embedding $\mathbf{e}_a^m$, which is related to the mashup m, is obtained. This representation embedding contains cloud API functional similarity features and complementary features, which are fused according to the attention of the target mashup. Feeding the cloud API personality embedding $\mathbf{e}_a^m$ together with the corresponding mashup representation embedding $\mathbf{e}_m$ into the prediction function, the probability that the mashup m may invoke the cloud API a is predicted as follows:

$$\hat{y}_{m,a} = \sigma_3(\mathbf{e}_a^m \cdot \mathbf{e}_m) \tag{10}$$

Since the mashup-oriented cloud API recommendation is an implicit feedback problem, the target value $y_{m,a}$ is a binary 1 or 0, which indicates whether mashup m will invoke cloud API a. So, in the above equation, we use the sigmoid function for the output. The loss function defined by cross entropy can better focus on the duality of implicit data [6]. Therefore, the loss function is defined as:

$$Loss = -\sum_{(m,a)\in Y} (y_{m,a} \log(\hat{y}_{m,a}) + (1 - y_{m,a}) \log(1 - \hat{y}_{m,a})) \tag{11}$$

where Y is a training set of size n, including mashup–API interaction data and random negative samples.

5. Experiments

In this section, we first detail the experimental setup, including the dataset, evaluation metrics, and parameter settings. Then, we outline a series of experiments that were conducted to answer the following research questions:

RQ 1: How does our proposed algorithm perform compared to state-of-the-art cloud API recommendation methods?

RQ 2: Does fusing similarity features and complementarity features to enhance cloud API feature representation improve the performance of recommendation algorithms?

RQ 3: Does biased fusion of similarity and complementarity using mashup-related attention benefit recommendation performance improvement?

RQ 4: Does splitting metapaths and using the attention mechanism to aggregate cloud API features improve recommendation performance?

RQ 5: How can our proposed approach be applied in reality?

5.1. Preparation

5.1.1. Datasets and Experiment Settings

In the experiments, the real data information was crawled from the ProgrammableWeb platform by the crawler program to obtain reliable experimental results. After data extraction and cleaning, the final experimental dataset was obtained, as shown in Table 1. The experimental data of this article can be downloaded and obtained from our shared link on Github: https://github.com/jiaojianz/CloudAPIdata, accessed on 22 November 2021.

Table 1. Statistics for cloud API datasets.

Item	Value
Number of mashups	6019
Number of cloud APIs	23,520
Number of invocations	12,564
Number of cloud APIs invoked	1509
Average number of cloud APIs per mashup	2.09
Total number of tags	440
Number of tags for cloud APIs	373
Number of tags for mashups	415
Sparsity of mashup–API interaction matrix	99.86%

Since there are no mashup–API rating data in this dataset, mashup-oriented cloud API recommendation is identified as an implicit feedback problem. So, we use their invoked data as the rates. For example, if mashup m_i has invoked cloud API a_j, the rating data are assigned a value of 1, and we use this value as the rating for the mashup–API pair. Furthermore, we randomly select some cloud APIs that are not used by the mashup and mark them as 0, as negative samples.

We randomly select 80% of existing mashups and their associated cloud APIs and categories for training, and the remaining 20% are divided equally for validation and testing. Our experiments were carried out at embedding size 64, learning rate 1×10^{-4}, and dropout rate 0.5. An early stopping strategy was implemented in the experiments, and we considered the algorithm to converge if the loss increases on the training dataset or the evaluation metric recall decreases on the validation set for 10 consecutive epochs.

5.1.2. Evaluation Metrics

The performance of our model and baseline models are evaluated on the ProgrammableWeb dataset introduced above. To measure the performance of top-N recommendation sets, two commonly used metrics were chosen in our experiment: recall and normalized discounted cumulative gain (NDCG).

Given a full ranking with all cloud API candidates for target mashups, recall is a measure to calculate the proportion of the top-N-related cloud APIs out of all true related cloud APIs.

$$Recall@N = \frac{|\{True\ APIs\} \cap \{Top - N\ APIs\}|}{|\{True\ APIs\}|} \tag{12}$$

where $\{True\ APIs\}$ is the true cloud API set that a mashup invoked, $\{Top - N\ APIs\}$ is the top-N recommended cloud API set.

NDCG is a measure of ranking quality in which positions are log-discounted. The location of correct recommendations is accounted for by introducing a location influence factor that assigns higher scores to top-ranked correct recommendations.

$$NDCG@N = \frac{DCG@N}{IDCG@N} \tag{13}$$

where $DCG@N = \sum_{i=1}^{N} \frac{2^{rel(i)}-1}{\log_2(i+1)}$ is discounted cumulative gain, $rel(i)$ represents the relevance score of the recommendation result at position i, and $IDCG@N$ is the ideal DCG of the ranking lists of candidates.

5.2. Experimental Results

5.2.1. Performance Comparison

Since there is currently no universally applied baseline model for mashup-oriented cloud API recommendation, some widely used methods and some recent state-of-the-art methods are selected to compare the performance of our proposed method. Some of them have been shown to be applicable to cloud API recommendations. The baseline recommendation methods chosen for comparison covered three categories, including historical interaction-based recommendations (POP, BPR, NGCF, DMF), context-aware recommendations (DeepFM, AFM), and knowledge-based recommendations (KGCN, MKR, KGAT). Brief descriptions of the baseline methods are as follows:

- POP method discovers and records the popularity of cloud APIs in the dataset and recommends the most popular cloud APIs.
- BPR generates recommendations by ranking cloud APIs with maximum posterior probability obtained from Bayesian analysis [26].
- NGCF uses graph neural networks to capture higher-order collaborative signals from API calls and use them for collaborative filtering [27].
- DMF implements recommendations based on a deep matrix decomposition model of neural networks [28].

- DeepFM combines the recommendation capabilities of a factorization machine with the feature representation capabilities of deep learning [29].
- AFM improves FM by distinguishing the importance of different feature interactions, which are learned from the data through a neural network [30].
- KGCN enriches the embedded representation of items by aggregating neighbor information, catching local neighborhood structures, and obtaining the personalized preferences of users [31].
- MKR automatically learns higher-order feature interactions between items and entities in the knowledge graph through cross-compression units [32].
- KGAT introduces knowledge graphs and user behavior data through graph neural network technology to reveal higher-order information as side information [33].

Tables 2 and 3 show the performance comparison results of our method and several baseline recommendation methods introduced above. A larger recall and NDCG indicate better performance.

Table 2. Recall comparison of baseline methods and proposed method.

Method	Recall					
	Top 2	**Top 3**	**Top 5**	**Top 10**	**Top 20**	**Top 30**
POP	0.0075	0.0207	0.0368	0.0554	0.2339	0.4771
BPR	0.0725	0.0873	0.1205	0.1784	0.2437	0.2834
NGCF	0.1233	0.1655	0.2373	0.3364	0.4110	0.4666
DMF	0.2136	0.2479	0.3172	0.3927	0.4945	0.5440
DeepFM	0.1706	0.2149	0.3073	0.4094	0.5184	0.5742
AFM	0.2491	0.2928	0.3563	0.4122	0.5265	0.5640
KGCN	0.2386	0.2836	0.3500	0.4319	0.5388	0.5866
MKR	0.2121	0.2732	0.2778	0.3282	0.4308	0.5159
KGAT	0.1936	0.2512	0.3353	0.4644	0.5662	0.6234
Ours	0.3063	0.3578	0.3997	0.4848	0.5719	0.6364

Table 3. NDCG comparison of baseline methods and proposed method.

Method	NDCG					
	Top 2	**Top 3**	**Top 5**	**Top 10**	**Top 20**	**Top 30**
POP	0.0069	0.0136	0.0203	0.0263	0.0685	0.1205
BPR	0.0622	0.0696	0.0832	0.1019	0.1184	0.1268
NGCF	0.1025	0.1236	0.1531	0.1853	0.204	0.2159
DMF	0.1916	0.2087	0.2375	0.2625	0.2882	0.2987
DeepFM	0.1620	0.1842	0.2225	0.2553	0.2826	0.2945
AFM	0.2214	0.2432	0.2689	0.2868	0.3159	0.3240
KGCN	0.2153	0.2378	0.2652	0.2918	0.3189	0.3291
MKR	0.1898	0.2226	0.2433	0.2768	0.2981	0.3101
KGAT	0.1664	0.1952	0.2299	0.2719	0.2978	0.3101
Ours	0.3140	0.3408	0.3584	0.3841	0.4055	0.4190

The experimental results show that the method proposed in this paper achieves the best performance as it achieves higher recall and NDCG in each top-N recommendation list, which indicates that our method is effective.

In the above comparison experiments, the methods POP, BPR, NGCF, and DMF only exploit the implicit interaction relationship between mashup and cloud API for recommendation, while DeepFM and AFM, two methods based on factorized machines, incorporate tag features to enhance the vector representation of mashup and cloud API. KGCN, MKR, and MKR are implemented based on graph structures. Both recall and NDCG methods that consider tag information and graph features perform significantly better than the recommended methods that only consider interaction information. It can be seen that

introducing information other than interaction information in mashup-oriented cloud API recommendation can effectively improve recommendation performance.

5.2.2. Ablation Experiments

In the previous section, we pointed out that, in addition to cloud API functional similarity features, historical common invocation information of multiple cloud APIs is beneficial for improving the accuracy of mashup-oriented cloud API recommendations. Therefore, it is necessary to verify whether introducing common invoked information characterizing cloud API complementary in cloud API recommendation systems can lead to improved recommendation performance.

Figure 3a,b gives the ablation experiments of our proposed algorithm, giving a comparison of experiments considering only cloud API similarity features and cloud API complementary features, and considering both features based on attention mechanisms. In addition, we also utilize other graph-based recommendation models, KGCN and KGAT, for validating the impact of introducing cloud API functional similarity and complementary on recommendation performance and comparing them with our algorithm; the validation results are also given in Figure 3. In the experiments, KGCN-S, KGAT-S, and ours-S indicate that only the functional similarity of cloud APIs is considered, and KGCN-C, KGAT-C, and ours-C indicate that only the complementary feature of cloud APIs is considered.

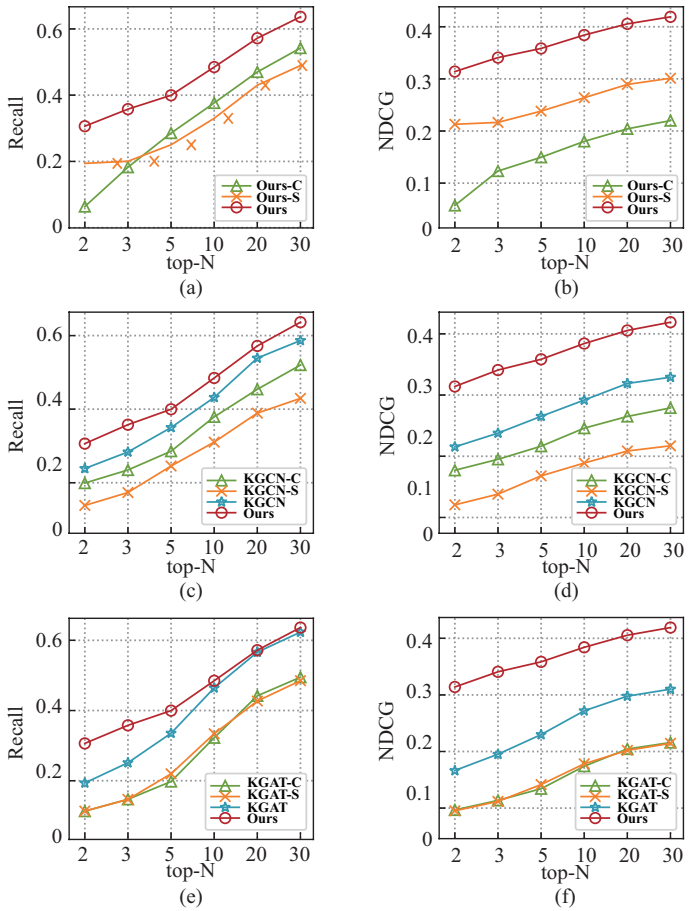


Figure 3. Recommendation performance of similarity and complementary considered separately and simultaneously. (**a**) Recall comparison of our method. (**b**) NDCG comparison of our method. (**c**) Recall comparison of KGCN. (**d**) NDCG comparison of KGCN. (**e**) Recall comparison of KGAT. (**f**) NDCG comparison of KGAT.

The experimental results show a significant improvement in improving the recommendation performance by considering the functional similarity and complementary of cloud

API together. We observe that both Recall and NDCG, which consider only functional similarity, are significantly lower than combining functional similarity and complementary at different values of top-N. This indicates that adding auxiliary information characterizing cloud API complementary feature to mashup-oriented cloud API recommendations will indeed greatly improve the recommendation performance.

5.2.3. Impact of Mashup-Related Attention Fusion

In metapath-based graph neural network algorithms, such as HAN [34] and MAGNN [35], the fusion of metapaths usually employs a global-based self-attention mechanism. We considered that different mashups may have different preferences for cloud API functional similarity and complementarity, and designed a mashup-related attention for the fusion of different paths.

We compared the experimental results using equal attention, self-attention, and mashup-related attention in the metapath fusion module, as shown in Tables 4 and 5.

Table 4. Recall comparison of different fusion mechanisms.

Fusion Mechanisms	**Recall**					
	Top 2	**Top 3**	**Top 5**	**Top 10**	**Top 20**	**Top 30**
Equal attention	0.2579	0.2863	0.3559	0.4075	0.4926	0.5532
Self attention	0.2373	0.2895	0.3630	0.4300	0.5093	0.5603
Mashup-related attention [1]	**0.3063**	**0.3578**	**0.3997**	**0.4848**	**0.5719**	**0.6364**
%Improv. [2]	0.1877	0.2360	0.1011	0.1273	0.1228	0.1359

[1] Bolded indicates the methods proposed in this paper.[2] Percentage improvement in the proposed method, i.e., mashup-related attention fusion, over the strongest fusion mechanism method.

Table 5. NDCG comparison of different fusion mechanisms.

Fusion Mechanisms	**NDCG**					
	Top 2	**Top 3**	**Top 5**	**Top 10**	**Top 20**	**Top 30**
Equal attention	0.2622	0.2771	0.3064	0.3230	0.3446	0.3565
Self attention	0.2454	0.2731	0.3047	0.3265	0.3453	0.3558
Mashup-related attention [1]	**0.3140**	**0.3408**	**0.3584**	**0.3841**	**0.4055**	**0.4190**
%Improv. [2]	0.1976	0.2299	0.1699	0.1766	0.1744	0.1754

[1] Bolded indicates the methods proposed in this paper.[2] Percentage improvement of the proposed method, i.e., mashup-related attention fusion, over the strongest fusion mechanism method.

It can be seen that the method of mashup-related attention fusion in this paper obtains the best results at any value of top-N, which greatly improves the recommendation performance. In addition, it can be seen that the performance of recommendations based on the fusion of the equal attention mechanism and the self-attention mechanism does not differ much. The self-attention scores are given subsequently in our case study experimental section, and the experiment results illustrate that similarity features and complementarity features are almost equally important for mashup-oriented cloud API recommendation in the whole cloud API ecosystem without considering the mashup's different requirements. This clearly leads to the mashup's requirements being ignored, and thus the recommendation is less effective than the proposed method of recommending cloud APIs that align with the mashup's requirements.

5.2.4. Impact of Aggregation Algorithms and Introduction of Metapath

In the metapath aggregation module, graph attention neural networks are used to compute metapath-based cloud API representations on each subgraph of metapath-based neighbors in this paper. To verify that the introduction of attention mechanisms can better implement mashup-oriented cloud API recommendations, we replaced the metapath

internal aggregation algorithm for comparison. Two representative graph neural networks, GCN and GraghSAGE, were chosen for comparison.

In Figure 4, the GAT-path indicates that the functional similarity and complementarity of cloud APIs are computed separately by different metapaths, and the graphical attention algorithm is used in the internal aggregation module of the metapaths, i.e., the method proposed in this paper. The GCN-path and SAGE-path indicate the replacement of the internal aggregation module with GCN and GraphSAGE, respectively. As we can see in the figure, the aggregation of metapath-based neighbors using the attention mechanisms achieved better results: recall is higher than the aggregation using the GCN and GraphSAGE algorithms, and NDCG has a significant improvement.

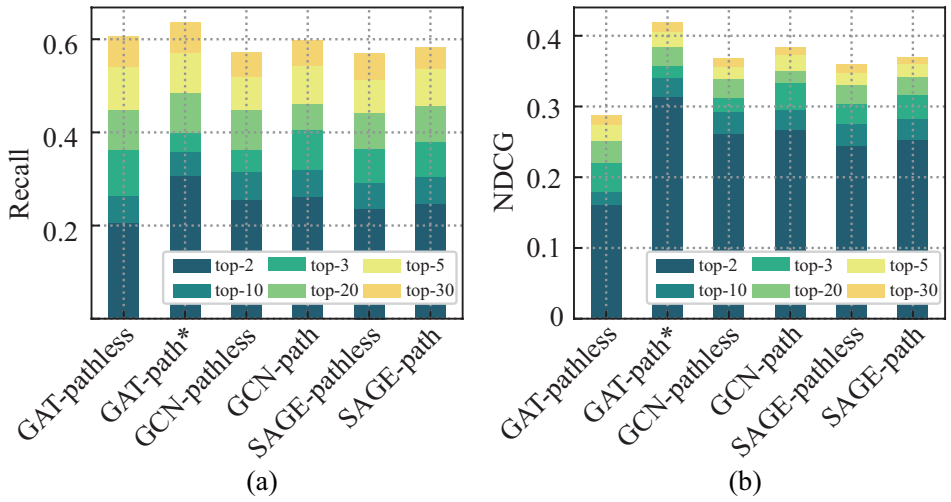


Figure 4. Recommendation performance of utilizing different internal aggregation algorithms and introducing metapaths or not. GAT-path is the method proposed in this paper. (**a**) Recall comparison. (**b**) NDCG comparison.

Furthermore, in order to prove that the calculation of functional similarity features and complementary features of cloud APIs based on metapaths is helpful for mashup-oriented cloud API recommendation after the reasonable division of metapaths, we conducted comparative experiments to verify whether introducing metapaths can improve the recommendation.

In Figure 4, GAT-pathless, GCN-pathless, and SAGE-pathless indicate the aggregation of web ecosystem graphs directly using GAT, GCN, and GraphSAGE algorithms, respectively, without considering the different concerns of cloud API functional similarity and complementary based on metapaths.

As shown in Figure 4, regardless of which aggregation algorithm is used, adding metapaths to consider the functional similarity and complementarity of cloud APIs can improve the recommendation performance to some extent. This result proves that, as proposed in this paper, it is necessary to consider that different mashups pay different attention to cloud API similarity features and relevance in mashup recommendations. After distinguishing cloud API similarity features and complementary features and calculating them separately, weighted fusion of feature vectors from different paths based on mashup preferences can help improve recommendation performance.

5.2.5. Case Study

In order to demonstrate the availability and usefulness of the proposed method more intuitively, actual mashups were selected as examples. Tags indicate the functions or services offered by the mashup. Therefore, developer requirements, which are used as inputs to characterize the mashup, are represented by tags. The actual tags of the example mashups are selected as follows:

- Mashup1: Social, e-commerce, reference;
- Mashup2: Mapping.

Based on the given requirements, the similarity and complementarity features of the candidate cloud APIs are learned separately using attention graph neural networks. Based on the given requirements, the similarity and complementarity features of the candidate cloud APIs are learned separately using attention graph neural networks. Then, the attention scores of the to-be-developed mashup for the similarity and complementarity features are computed, and the results are given in Table 6. For comparison, the global self-attention that is not associated with the mashup is given at the same time.

Table 6. Mashup attention score.

Item	Attention to Similarity	Attention to Complementarity
Mashup1	0.3024	0.6976
Mashup2	0.8533	0.1467
Self-attention	0.5101	0.4899

The above attention scores are used to fuse the similarity and complementarity features of the candidate cloud APIs, and these features are used to predict the probabilities that the given mashup may invoke each cloud API. Candidate cloud APIs with high probability are recommended for the mashup. Table 7 gives the list of the top five cloud API recommendations for mashup instances under four scenarios: considering only similarity, considering only complementarity, fusion by self-attention, and fusion by mashup-related attention.

Table 7. List of cloud API top five recommendations.

Mashup	Cloud API (First Tag)			
	Only Similarity	**Only Complementarity**	**Self-Attention**	**Mashup-Related Attention**
Mashup1	1. yahoo-maps (mapping) 2. Twitter (social) 3. google-maps (mapping) 4. google-earth (mapping) **5. Youtube (video)**	1. reuters-spotlight (news) 2. Ribbit (telephony) 3. google-maps (mapping) 4. Flickr (photos) 5. Twitter (social)	1. google-maps (mapping) 2. Flickr (photos) **3. Youtube (video)** 4. Twitter (social) **5. Facebook (social)**	1. Twitter (social) **2. Youtube (video)** **3. amazon-product-advertising (e-commerce)** **4. Facebook (socials)** 5. Flickr (photos)
Mashup2	1. microsoft-bing-maps (mapping) **2. google-maps (mapping)** 3. google-earth (mapping) **4. yahoo-maps (mapping)** 5. Youtube (video)	1. reuters-spotlight (news) **2. google-maps (mapping)** 3. Flickr (photos) 4. Twitter (social) 5. Facebook (social)	**1. google-maps (mapping)** 2. Flickr (photos) 3. Youtube (video) 4. Twitter (social) 5. Facebook (social)	**1. google-maps (mapping)** 2. microsoft-bing-maps (mapping) 3. google-earth (mapping) **4. yahoo-geocoding (mapping)** 5. google-maps-data (mapping)

Bolding indicates that the cloud API is actually called by the to-be-developed mashup.

In actuality, mashup1 calls six cloud APIs, namely, amazon-product-advertising (advertising, e-commerce), amazon-a9-opensearch (search), Youtube (video, media), and Facebook (social, webhooks), amazon-simpledb (database), amazon-marketplace-web-service (e-commerce), and mashup2 calls three cloud APIs, namely, google-maps (mapping, viewer), yahoo-maps (mapping, viewer), and yahoo-geocoding (mapping, addresses).

From the recommendation results in Table 7, it can be seen that the recommendation results that only consider similarity often contain multiple similar cloud APIs, which are not applicable to all types of mashups, whereas in the recommendation that only considers complementarity, multiple types of APIs are recommended, but functional similarity is ignored, which means that the recommended APIs may not be able to satisfy the functional requirements of mashups. The self-attention scores of different features are almost equal without considering alignment with mashup requirements, indicating that similarity features and complementarity features are almost equally important in the global consideration towards the cloud API ecosystem, but this obviously leads to mashup's requirements

being neglected. Our method, on the other hand, is based on the mashup-related attention mechanism, which fully considers the needs of different mashups and can more effectively give a recommended list of cloud APIs that are aligned with the mashup's requirements.

6. Conclusions

In order to improve the accuracy of mashup-oriented cloud API recommendation, a cloud API recommendation method that fuses cloud API similarity features and complementarity features is proposed. For more accurate and comprehensive cloud API feature representation, a heterogeneous graph neural network based on metapath and attention mechanisms is utilized to introduce rich side information into the cloud API ecosystem. The cloud API similarity feature and complementarity feature representations are obtained by aggregating similarity neighbor features and complementarity neighbor features in different metapaths by using the self-attention mechanism. In addition, considering that different mashups have different preferences for similarity and complementarity, cloud API similarity features and complementarity features are biasedly fused by using attention related to the given mashup's requirements. Requirement-aligned feature representations of candidate cloud APIs that are tightly related to the target mashup are obtained and used to recommend cloud APIs for mashup developers. Experimental results on a real cloud API dataset show that requirement-aligned representation of cloud APIs using mashup-related attention considering both similarity and complementarity improves recommendation performance, effectively recalling more cloud APIs compared to the baseline method. We plan to explore more useful auxiliary information in the future to improve the performance of the method.

Author Contributions: Conceptualization, L.S.; methodology, Y.W.; software, Z.C. and Y.W.; validation, Y.W.; writing—original draft preparation, Y.W.; writing—review and editing, L.S., S.Z., and Z.C.; supervision, L.S. All authors have read and agreed to the published version of the manuscript.

Funding: This research was funded by the National Natural Science Foundation of China (grant No. 62102348 and grant No. 61772450), Natural Science Foundation of Hebei Province China (grant No. F2022203012), Science and Technology Research Project of Hebei University (grant No. QN2020183), and Project of Hebei Key Laboratory of Software Engineering (grant No. 22567637H).

Data Availability Statement: The data used to support the findings of this study are available from the corresponding author upon request.

Conflicts of Interest: The authors declare no conflict of interest.

References

1. Rojas, H.; Arias, K.A.; Renteria, R. Service-oriented architecture design for small and medium enterprises with infrastructure and cost optimization. *Procedia Comput. Sci.* **2021**, *179*, 488–497. [CrossRef]
2. Jiang, B.; Liu, P.; Wang, Y.; Chen, Y. HyOASAM: A Hybrid Open API Selection Approach for mashup Development. *Math. Probl. Eng.* **2020**, *2*, 2–16. [CrossRef]
3. Zhou, J.; Jiang, B.; Yang, J.; Yang, J.; Li, H.; Wang, N.; Wang, J. Service Discovery Method Based on Knowledge Graph and Word2vec. *Electronics* **2022**, *11*, 2500. [CrossRef]
4. Botangen, K.A.; Yu, J.; Sheng, Q.Z.; Han, Y.; Yongchareon, S. Geographic-aware collaborative filtering for web service recommendation. *Expert Syst. Appl.* **2020**, *151*, 113347. [CrossRef]
5. Jiang, B.; Yang, J.; Qin, Y.; Wang, T.; Wang, M.; Pan,W. A Service Recommendation Algorithm Based on Knowledge Graph and Collaborative Filtering. *IEEE Access* **2021**, *9*, 50880–50892. [CrossRef]
6. Xue, H.; Dai, X.; Zhang, J.; Huang, S.; Chen, J. Deep Matrix Factorization Models for Recommender Systems. In Proceedings of the 26th International Joint Conference on Artificial Intelligence, Melbourne, Australia, 19–25 August 2017.
7. Fletcher, K. Regularizing matrix factorization with implicit user preference embeddings for web API recommendation. In Proceedings of the IEEE International Conference on Services Computing, Milan, Italy, 8–13 July 2019.
8. Chen, K.; Mao, H.; Shi, X.; Xu, Y.; Liu, A. Trust-aware and location-based collaborative filtering for web service QoS prediction. In Proceedings of the IEEE 41st Annual Computer Software and Applications Conference, Turin, Italy, 4–8 July 2017.
9. Meng, S.; Li, Q.; Chen, S.; Yu, S.; Qi, L.; Lin, W.; Xu, X.; Dou, W. Temporal-sparsity aware service recommendation method via hybrid collaborative filtering techniques. In Proceedings of the International Conference on Service-Oriented Computing, Hangzhou, China, 12–15 November 2018.

10. Nguyen, M.; Yu, J.; Nguyen, T.; Han, Y. Attentional matrix factorization with context and co-invocation for service recommendation. *Expert Syst. Appl.* **2021**, *186*, 115698. [CrossRef]
11. Gu, Q.; Cao J.; Liu, Y. CSBR: A Compositional Semantics-Based Service Bundle Recommendation Approach for mashup Development. *IEEE Trans. Serv. Comput.* **2022**, *15*, 3170–3183. [CrossRef]
12. Cao, B.; Liu, X.F.; Rahman, M.M.; Li, B.; Liu, J.; Tang, M. Integrated content and network-based service clustering and web APIs recommendation for mashup development. *IEEE Trans. Serv. Comput.* **2020**, *13*, 99–113. [CrossRef]
13. Zhao, H.; Wang, J.; Zhou, Q.; Wang, X.; Wu, H. Web API recommendation with features ensemble and learning-to-rank. In Proceedings of the CCF Conference on Big Data, Wuhan, China, 26–28 September 2019.
14. Shi, M.; Tang, Y.; Liu, J. Functional and contextual attention-based LSTM for service recommendation in mashup creation. *IEEE Trans. Parallel Distrib. Syst.* **2019**, *30*, 1077–1090. [CrossRef]
15. Zhong, Y.; Fan, Y.; Tan, W.; Zhang, J. Web service recommendation with reconstructed profile from mashup descriptions. *IEEE Trans. Autom. Sci.* **2018**, *15*, 468–478. [CrossRef]
16. Yao, L.; Wang, X.; Sheng, Q.Z.; Benatallah, B.; Huang, C. Mashup Recommendation by Regularizing Matrix Factorization with API Co-Invocations. *IEEE Trans. Serv. Comput.* **2021**, *14*, 502–515. [CrossRef]
17. Xia, B.; Fan, Y.; Tan, W.; Huang, K.; Zhang, J.; Wu, C. Category-Aware API Clustering and Distributed Recommendation for Automatic mashup Creation. *IEEE Trans. Serv. Comput.* **2015**, *8*, 674–687. [CrossRef]
18. Lian, S.; Tang, M. API recommendation for mashup creation based on neural graph collaborative filtering. *Connect. Sci.* **2022**, *34*, 124–138. [CrossRef]
19. Kwapong, B.; Fletcher, K. A knowledge graph based framework for web API recommendation. In Proceedings of the IEEE World Congress on Services, Milan, Italy, 8–13 July 2019.
20. Xie, F.; Zhang, Y.; Przystupa, K.; Kochan, O.A. Knowledge Graph Embedding Based Service Recommendation Method for Service-Based System Development. *Electronics* **2023**, *12*, 2935. [CrossRef]
21. Geng, J.; Cao, B.; Ye, H.; Chen, J.; Peng, M.; Liu, J. Web service recommendation based on knowledge graph convolutional network and doc2vec. In Proceedings of the IEEE World Congress on Services, Beijing, China, 18–23 October 2020.
22. Wang, X.; Wu, H.; Hsu, C.H. Mashup-oriented API recommendation via random walk on knowledge graph. *IEEE Access* **2019**, *7*, 7651–7662. [CrossRef]
23. Wang, X.; Liu, X.; Liu, J.; Chen, X.; Wu, H. A novel knowledge graph embedding based API recommendation method for mashup development. *World Wide Web* **2021**, *24*, 869–894. [CrossRef]
24. Gao,W.; Wu, J. Multi-Relational Graph Convolution Network for Service Recommendation in mashup Development. *Appl. Sci.* **2022**, *12*, 924. [CrossRef]
25. Veličković, P.; Cucurull, G.; Casanova, A.; Romero, A.; Liò, P.; Bengio, Y. Graph attention networks. In Proceedings of the International Conference on Learning Representations, Vancouver, Canada, 30 April–3 May 2018.
26. Rendle, S.; Freudenthaler, C.; Gantner, Z.; Schmidt-Thieme, L. BPR:Bayesian personalized ranking from implicit feedback. In Proceedings of the Conference on Uncertainty in Artificial Intelligence, Montreal, Canada, 18–21 June 2009.
27. Ling, C.; Zou, Y.; Xie, B. Graph neural network based collaborative filtering for API usage recommendation. In Proceedings of the IEEE International Conference on Software Analysis, Evolution and Reengineering, Honolulu, HI, USA, 9–12 March 2021.
28. Ramathulasi, T.; Babu, M.R. Enhanced PMF model to predict user interest for web API recommendation. In *Handbook of Research on Advances in Data Analytics and Complex Communication Networks*; Krishna, V., Ed.; IGI Global: Hershey, PA, USA, 2022; pp. 131–139.
29. Cao, B.; Peng, M.; Qing, Y.; Liu, J.; Kang, G.; Li, B.; Fletcher, K.K. Web API recommendation via combining graph attention representation and deep factorization machines quality prediction. *Concurr. Comput. Pract. Exp.* **2022**, *34*, e7069. [CrossRef]
30. Cao, Y.; Liu, J.; Shi, M.; Cao, B.; Chen, T.; Wen, Y. Service recommendation based on attentional factorization machine. In Proceedings of the IEEE International Conference on Services Computing, Milan, Italy, 8–13 July 2019.
31. Zhang, Y.; Yang, H.; Kuang, L. A web API recommendation method with composition relationship based on GCN. In Proceedings of the IEEE Intl Conf on Parallel & Distributed Processing with Applications, Big Data & Cloud Computing, Sustainable Computing & Communications, Social Computing & Networking, Exeter, UK, 17–19 December 2020.
32. Wang, H.; Zhang, F.; Zhao, M.; Li, W.; Xie, X.; Guo, M. Multi-task feature learning for knowledge graph enhanced recommendation.In Proceedings of the World Wide Web Conference, San Francisco, CA, USA, 13–17 May 2019.
33. Chen, Z.; Zhang, T.; Peng, X. A novel API recommendation approach by using graph attention network.In Proceedings of the IEEE International Conference on Software Quality, Reliability and Security, Hainan, China, 6–10 December 2021.
34. Wang, X.; Ji, H.; Shi, C.; Wang, B.; Ye, Y.; Cui, P.; Yu, P.S. Heterogeneous graph attention network. In Proceedings of the World Wide Web Conference, San Francisco, CA, USA, 13–17 May 2019.
35. Fu, X.; Zhang, J.; Meng, Z.; King, I. MAGNN: Metapath aggregated graph neural network for heterogeneous graph embedding. In Proceedings of the Web Conference 2020, Taipei, Taiwan, 20–24 April 2020.

Disclaimer/Publisher's Note: The statements, opinions and data contained in all publications are solely those of the individual author(s) and contributor(s) and not of MDPI and/or the editor(s). MDPI and/or the editor(s) disclaim responsibility for any injury to people or property resulting from any ideas, methods, instructions or products referred to in the content.