Adaptive Web API Recommendation via Matching Service Clusters and Mashup Requirement

Yue Zhu, Guobing Zou(✉), Song Yang, Shengxiang Hu, Pengtao Li, and Chunhua Zeng

School of Computer Engineering and Science, Shanghai University, Shanghai, China
{gbzou,yangsong,shengxianghu,22721507,kriya}@shu.edu.cn

Abstract. With the rapid proliferation of Web services and Web APIs, recommendation systems can effectively address the issue of information overload and alleviate the burden of meaningless filtering. Existing approaches can help filtering appropriate Web services for mashup creation, however, they often fall short of developers' different and personalized needs by recommending only a fixed number of APIs and lack precision in aligning mashup requirements across all categories. To solve the above issue, this paper introduces a novel Web service recommendation framework called AWAR for mashup creation, which focuses on the matching strategy between mashup requirements and Web APIs, and enhances recommendation effectiveness by integrating natural language processing, optimization algorithms, and deep learning. Extensive experiments conducted on large-scale real datasets demonstrate that the proposed approach receives superior recommendation results on multiple evaluation metrics compared to advanced competing baselines.

Keywords: Web Service · Service recommendation · Mashup Creation · Deep Neural Network · 0–1 planning

1 Introduction

With the rapid development of service-oriented architecture, the Internet has witnessed a surge in the proliferation of Web services and Web APIs [1]. In addition, major Internet companies like Yahoo, Google, and Amazon have made their data accessible for integration with other data sources, resulting in the emergence of new tools enabling developers to easily aggregate content from multiple sources to create a new service displayed in a single graphical interface, namely mashup service [2]. Therefore, mashup services have experienced rapid growth on the network by recommending a set of functionally correlated Web API services.

However, the exponential increase in the number of Web services and the homogenization of these services have presented developers with a challenge that in face of the vast sea of service information, they often struggle to quickly locate the services required

H. Gao and X. Wang (Eds.): CollaborateCom 2024, LNICST 626, pp. 102–118, 2026.
https://doi.org/10.1007/978-3-031-93257-1_6

to fulfill their development needs, resulting in information overload [3]. Addressing how to expedite developers' discovery of necessary services for mashup creation is a pivotal problem to be tackled.

The recommendation system can effectively address this issue and alleviate the burden of meaningless screening. Many researchers are committed to investigating how to recommend Web services to mashup developers to help them effectively building mashup services. Web service recommendations for mashup creation are classified based on the recommendation strategy and the information used. Service recommendations based on the quality of service often suggest potential high-quality services to developers once they have determined a set of required services. Collaborative filtering-based service recommendations typically suggest services to users based on service composition and historical service invocation records. Social relationship-based service recommendations often utilize external information, such as developers' social relationships, to recommend appropriate services. The function-based service recommendation algorithm aims to recommend services to mashup developers that can best meet their functional requirements, significantly reducing the burden of service discovery on mashup developers. These service recommendation approaches mentioned above for mashup developers can be applied in real-world Web service sharing platforms.

While existing approaches can help filter appropriate Web services for mashup creation, they still fail to fully meet the needs of mashup developers. The primary reason for this limitation is that existing approaches typically recommend only a fixed number of Web APIs. The size of the recommended API list is often artificially set, making it impossible to dynamically adjust the number of recommended Web APIs based on the required functions for creating the mashup service. Another issue arises from the inadequacy of the existing function-based service recommendation approaches in accurately matching the needs of mashups across all categories. The recommended Web APIs may contain multiple APIs with similar functions, yet these functions may not fully cover the requirements for creating a mashup service.

Aiming at the aforementioned research challenges, this paper proposes a novel service recommendation approach for mashup creation, called Adaptive Web API Recommendation (AWAR). In the process of recommending services based on developers' mashup needs, it usually involves matching mashup requirements with candidate services. We concentrate on refining the matching strategy between mashup requirements and services, leading to enhance the effectiveness of service recommendation through the integration of natural language processing, optimization algorithms, and deep learning. The main contributions of this paper are summarized as follows:

- We propose a novel Web API recommendation framework, which applies functional division of mashup requirements and optimization algorithms to adaptively recommend an indefinite number of Web API services for creating a mashup service.
- The recommended results are further optimized using a Multi-layer Perceptron combined with user-service historical service invocation records and other information from the mashup services, making the functions of the recommended services better cover the functions required for mashup creation.
- Large-scale experiments are conducted on real datasets, and the results demonstrate that the proposed approach receives superior Web API recommendation results.

The remainder of this paper is organized as follows: The problem is formulated in Sect. 2. Section 3 elaborates on our approach for extracting API business execution processes. Section 4 presents extensive experiments and analyzes their performance. Section 5 reviews the related work. Finally, Sect. 6 concludes the paper.

2 Problem Formulation

Definition 1 (Web Service). Web service is a platform-independent, modular, self-contained and composable Web-based application with reusability and programmability [4]. A Web service is denoted as $s = N^{(s)}, D^{(s)}, C^{(s)}$ consists of the name of the service $N^{(s)}$, the textual functional description provided by the publisher $D^{(s)}$ and the category of the service $C^{(s)}$. Moreover, we denote $S = \left\{s_1, s_2, \ldots, s_{|S|}\right\}$ as the set of all the Web services in the service repository, where $|S|$ is the total number of Web services.

Definition 2 (Mashup Service). One or more Web services can be aggregated to create new services that meet the comprehensive needs of users and provide additional business value. Generally, the API interfaces of the source services will be used during aggregation. The new service obtained through aggregation is a mashup service [5, 6]. A mashup is denoted as $m = N^{(m)}, D^{(m)}, R^{(m)}$ contains the name of the mashup service N_m, the textual functional description provided by the publisher D_m and a set of APIs invoked by the mashup service from the service repository $R^{(m)} = \left\{s_1^{(m)}, s_2^{(m)}, \ldots, s_{|R^{(m)}|}^{(m)}\right\}$, $\left|R^{(m)}\right|$ is the number of related APIs. As a result, we denote $M = \left\{m_1, m_2, \ldots, m_{|M|}\right\}$ as the set of all the mashup services in the service repository, where $|M|$ is the total number of mashup services.

Definition 3 (Web Service Recommendation for Mashup Creation). The problem to be solved is defined as follows. Suppose that the mashup service to be created is m^*. Given the user requirement $D^{(m^*)} = \left\{w_1^{(m^*)}, w_2^{(m^*)}, \ldots, w_{\left|D^{(m^*)}\right|}^{(m^*)}\right\}$ which consists of a set of words to describe the functional requirement of m^* and $\left|D^{(m^*)}\right|$ is the word count of the requirement, based on the functional descriptions of Web services and mashups, a set of APIs $m' = \left\{s_1^{(m')}, s_2^{(m')}, \ldots, s_{|m'|}^{(m')}\right\}$, where $\left|m'\right|$ is the number of APIs in the set, is needed to be recommended to the user to meet the requirement. The recommended API set is ideal when it includes all the relevant APIs without redundancy and is of an appropriate size. In this study, our focus lies on recommending an API set in which the APIs adequately cover the user's functional requirement without redundancy.

The flow chart of mashup service recommendation is depicted in Fig. 1. Initially, mashup developers, referred to as users, put forward a demand for mashup development. Subsequently, the recommendation system employs certain recommendation strategies to filter several APIs from the service repository that could potentially be invoked to create the mashup, based on the user demand. These APIs are then recommended to the users [7].

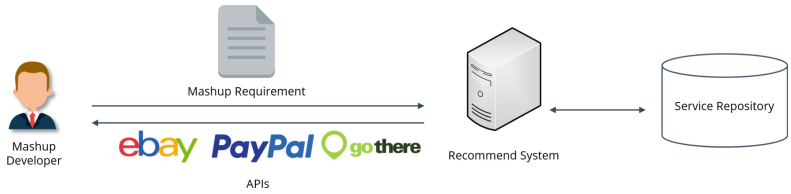

Fig. 1. The Workflow of Web Service Recommendation for Mashup Creation

3 Approach

The purpose is to recommend services based on the functional requirements of mashup developers and the API invocation records of mashups. Following popular service recommendation advancements in recent years, the process first typically involves service clustering and feature extraction. Subsequently, the similarity between demand features and different cluster features is calculated, sorted, and K clusters with the highest similarity are selected. Finally, a certain number of Web services are filtered from each selected cluster to obtain the list of recommended services. In these traditional approaches, the number of clusters selected is often set manually and the corresponding number of APIs cannot be dynamically recommended according to the number of functions required by the mashup service to be created. K is usually a value determined through numerous experiments to improve experimental results.

However, in practical applications, the service repository undergoes constant updates, necessitating dynamic adjustments to K adapting to real demands of personalized mashup requirements. Therefore, setting K as a fixed value potentially lowers the performance of Web API recommendation. Furthermore, there may be multiple APIs with similar functions because they are filtered from the same cluster, leading to inadequate coverage of the functions required to create a mashup service. For instance, a function that occupies a small space in the mashup description is likely to be overlooked when selecting the Top-K similar clusters.

3.1 Overall Framework of AWAR

To address the two disadvantages, a novel approach of Web API recommendation is proposed, focusing on improving the selection and matching of functional clusters with user requirements. It is called Adaptive Web API Recommendation (AWAR) for mashup creation, aiming to improve the recommendation accuracy. The workflow of WAR involves dividing a mashup requirement into multiple subtasks based on existing functional cluster features. It enables the automatic prediction of the number of relevant functional clusters and enhances the coverage of recommended APIs to better meet the mashup requirements. The overall framework of AWAR is illustrated in Fig. 2.

- In the component of mashup requirement feature extraction, the mashup requirement feature is derived from the demands proposed by the mashup developer through feature extraction.
- In the component of Web service cluster feature extraction, the Web services within a service repository are partitioned to a set of service clusters where each cluster

has Web services with similar functions, and then the features of these clusters are extracted to further match the extracted mashup feature.

- In the component of requirement division, m clusters are selected based on the condition that the similarity is the highest between the aggregated features of the clusters and the requirement features. It is equivalent to approximately dividing the requirements into corresponding number of subtasks.
- In the component of Web service recommendation, a specific number of services are chosen from each of these desired clusters.

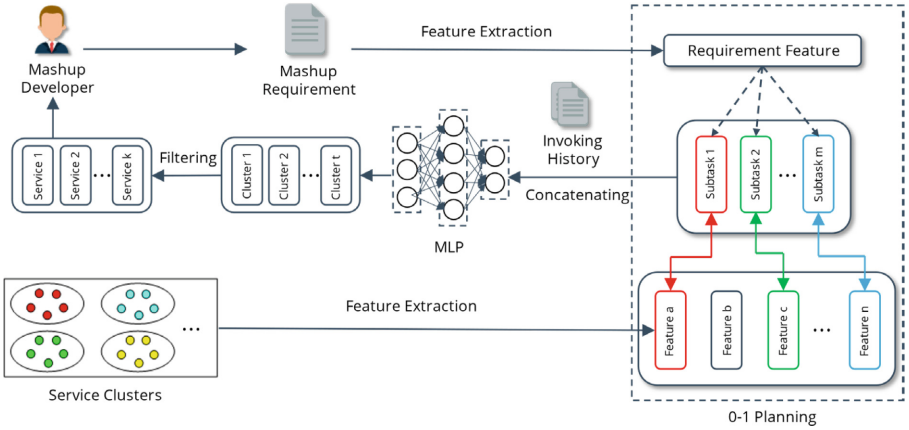

Fig. 2. The Overall Framework of Adaptive Web API Recommendation for Mashup Creation.

3.2 Matching Service Clusters and Mashup Requirement by 0–1 Planning Model

When the requirement is divided into m subtasks, it can be modeled as a 0–1 planning problem. From the existing n service clusters, m clusters are selected to be screened out so that the aggregated feature vector of these m clusters has the highest similarity with the mashup demand feature vector. As a result, the objective function is:

$$\max\left(Sim\left(MR, MR'\right)\right) \tag{1}$$

where MR is the feature vector of mashup requirement, and $MR\prime = a_1C_1 + a_2C_2 + \ldots + a_nC_n$ is the service feature vector obtained from the aggregation of m subtasks and $C_i(0 < i \leq n)$ is the feature vector of cluster i. The constraints are as follow:

$$s.t.a_1, a_2, \ldots, a_n = \begin{cases} 0 unselectthecluster \\ 1 selectthecluster \end{cases} \tag{2}$$

while m is the number of selected clusters equaling to the number with the value of 1 in $a_1, a_2, \ldots, a_n$.

Here, the simple genetic algorithm (SGA) is proposed to tackle the 0–1 programming problem because it can compute and search for the approximately optimal solution by

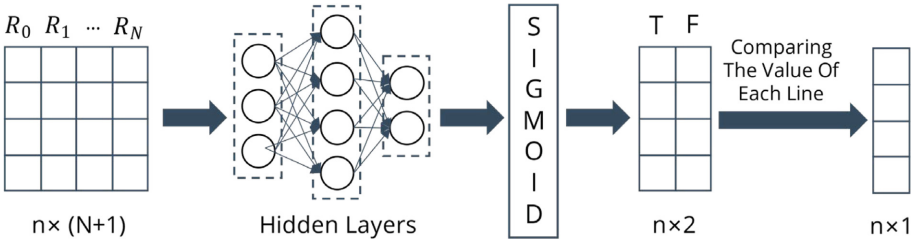

Fig. 3. The MLP Model.

simulating the principles of biological evolution, and demonstrates robust global optimization capabilities [8]. SGA simulates the phenomena occurring in natural selection and heredity through 3 fundamental operations: selection $\Phi(x)$, crossover $\Gamma(x)$ and mutation $\psi(x)$, collectively known as genetic operators.

First of all, the randomly initialized population P_0 is encoded into a fixed-length binary string by applying specific encoding methods. Starting from the encoded population P_0, the next generation P_1, which is more suited to the environment, can be generated through random selection, crossover and mutation operations. Thus, the population can evolve towards increasingly favorable regions within the search space. In this way, through successive generations of reproduction and evolution convergence occurs towards a group of individuals P_t which are best adapted to the environment, indicating obtain a high-quality solution to the problem.

As a result, SGA can be described as $SGA = (C, E, P_0, M, \Phi, \Gamma, \Psi, \mathrm{T})$ where C represents the individual coding method, E denotes the fitness function which describes individual performance and determines whether the individual is eliminated, P_0 is the initialized population, M is the population size, Φ, Γ and Ψ are 3 genetic operators, and T is the termination condition dictating when the evolution cease. When implementing SGA, parameters such as M, T, crossover probability p_c and mutation probability p_m should be initially set for generating a suboptimal to an originally adaptive Web API recommendation problem for mashup creation.

3.3 Recommending Adaptive Web APIs by Multi-layer Perceptron

The Multi-layer Perceptron (MLP) is also known as the Artificial Neural Network (ANN) that is a basic type of deep neural network [9, 10]. MLP is capable of detecting complex, non-linear interactions among features and evaluating their different levels of importance [11]. In our adaptive Web API recommendation framework, the MLP model in step 2 is used for capturing the interactions among the invocation records and target mashup service and it is depicted in Fig. 3. The output in step 1 signifies the APIs predicted for the creation of a mashup service. Relatively, invocation history comprises the record of the APIs utilized in crafting existing mashup services, thus the invocation history of similar mashup services as the target one serves as a practical reference for mashup creation. In the step of MLP, the result from step 1 and the invocation history of the Top-N similar mashups $\{R_1, \ldots, R_N\}$ are combined to forecast the APIs for mashup creation. Hence, the input of the MLP consists of the outcome of the 0–1 planning $R_0 =$

$\{a_1^0, a_2^0, \ldots, a_n^0\}$ concatenated with the invocation history matrix of similar mashup services $R_x = \{R_0, R_1, \ldots, R_N\}$, where R_x has the dimensions of $n \times (N+1)$, with n denoting the number of service clusters and N indicating the number of selected similar mashups.

Through several hidden layers, the sigmoid function is used to map the result between 0 and 1, which can be expressed as:

$$Sigmoid(x) = \frac{1}{1+e^{-x}} \tag{3}$$

Here, the activation function of the other layers is the Rectified Linear Unit (ReLU) function:

$$Relu(x) = \max(0, x) \tag{4}$$

The output is $R_y = Sigmoid(WR_x + b)$, which can also be represented as $\{R_y = R_T, R_F\}$, where w is the weight and b is the bias of MLP and R_y has the shape of $n \times 2$. Each element in the output denotes a probability that the cluster is or is not chosen, which ranges from 0 to 1. The element a_i^T in $R_T = \{a_1^T, a_2^T, \ldots, a_n^T\}$ indicates the probability that the service cluster i is selected, and the element a_i^F in $R_T = \{a_1^F, a_2^F, \ldots, a_n^F\}$ shows the probability that the cluster is not selected. Then, if the value of a_i^T is larger than the value of a_i^F, the i^{th} element of the final result is 1, otherwise, the value is 0, the function of which is:

$$a_i^{Final} = \begin{cases} 0 a_i^T < a_i^F \\ 1 a_i^T > a_i^F \end{cases} \tag{5}$$

Finally, the result $R_{Final} = \{a_1^{Final}, a_2^{Final}, \ldots, a_n^{Final}\}$ is obtained with the shape of $n \times 1$, where $a_i^{Final} = 1 or 0$ indicates whether to select the service cluster i or not. A particular quantity of services can be selected from each of these desired clusters afterwards.

3.4 Model Training and Parameter Optimization

In our Adaptive Web API recommendation framework, it is crucial to select an appropriate loss function which quantifies the dispersity between the predicted value $\hat{y}$ and the ground truth value y. In this paper, the binary cross entropy loss function, which is widely utilized in multi-label classification, is employed to train our recommendation model:

$$L = -\frac{1}{n}\sum_{q=1}^{n} y_q \log(p_q) + (1-y_q)\log(1-p_q) \tag{6}$$

where n is the number of clusters, p_q denotes the predicted possibility that a service cluster is selected, y_q indicates whether the service cluster is selected or not. That is, when $y_q = 1$ means the cluster is selected and $y_q = 0$ indicates that it is not selected for mashup creation.

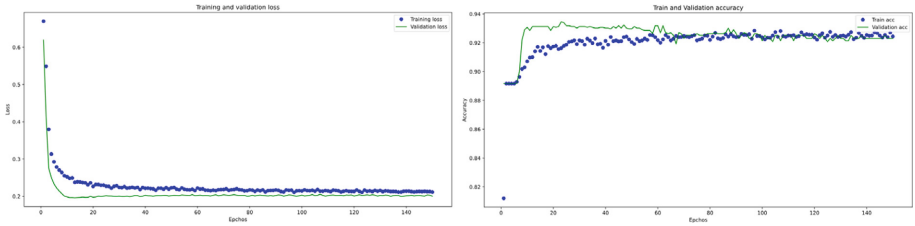

Fig. 4. Training Loss and Accuracy Curve.

The values of weights and bias are randomly initialized at first. After forward propagation, the error between the predicted results and the ground truth values is calculated. Then the error of each neurons in each layer is calculated backward during back propagation. As previously mentioned, for each layer of the MLP, the output is $y = F(w, b)$. Given the errors of each cell, gradient descent algorithm is applied for updating weights and bias in order to minimize errors:

$$w' = w - \eta \frac{\partial \mathrm{C}}{\partial \mathrm{w}} \tag{7}$$

$$b' = b - \eta \frac{\partial \mathrm{C}}{\partial \mathrm{b}} \tag{8}$$

where η represents the learning rate which may influence the convergence efficiency and performance.

Additionally, the selection of an optimizer is also an important part of deep learning model [10]. The mini-batch adaptive moment estimation (Adam) optimization algorithm is used to optimize the parameters of our adaptive Web API recommendation model. The Adam has the superiority of high computing effectiveness, minimal memory specifications, and robust interpretability, etc. Furthermore, Adam effectively accepts the estimation of the first and the second moments of the gradient to calculate the step size and efficiently integrates the strengths of the previously given two optimization algorithms RMSProp and AdaGrad. The function of training accuracy is:

$$Accuracy = \frac{TP + TN}{TP + FP + TN + FN} \tag{9}$$

The training accuracy and loss curve are illustrated in Fig. 4.

Table 1. Distribution of selected APIs.

Number of	Max	Min	Average
APIs per Category	850	229	38.3
APIs per Mashup	10	1	1.3

4 Experiments

4.1 Experimental Datasets and Setup

To evaluate the performance of AWAR, a series of experiments are conducted on an equipment configured with an Intel(R) Xeon(R) Gold 6130 CPU@2 and a 192GB RAM in the environment of python3.7. A Web information crawler has been developed and deployed to obtain Web services and related data up to July 1st, 2018 from ProgrammableWeb. The dataset crawled comprises 3,989 mashup services and 16,243 Web services across 403 categories. From these, 7,666 Web services from the top 20 categories and 2,724 mashup services containing only these Web services are selected for the experiments. The distribution of the selected services is presented in Table 1.

Following data preprocessing, APIs are clustered into several groups. Moreover, the language model BERT, introduced by Google in 2018, is implemented for word embedding in the experiments. BERT, based on the transformer architecture, is a pre-trained model known for its strong performance in language representation and feature extraction. In consideration of the relatively limited word count in most service descriptions and the computational cost of model training, we implement the BERT-base model (with parameters L = 12, H = 768, A = 12) for word embedding of both Web and mashup service descriptions.

4.2 Evaluation Metrics

The recommendation performance of AWAR is evaluated by 4 metrics: Precision, Recall, F1-score and Hit Ratio. TP is the number of services recommended which meet the requirement; FP is the number of services recommended which do not fit the requirement; FN is the number of services not recommended but required; HitNumber is the number of services recommended are required; GT is the number of Web services the mashup service actually required for creation. They are calculated as below.

$$Precision = \frac{TP}{TP + FP} \tag{10}$$

$$Recall = \frac{TP}{TP + FN} \tag{11}$$

$$F1 = \frac{2 \times Precision \times Recall}{Precision + Recall} \tag{12}$$

$$HR = \frac{HitNumber}{GT} \tag{13}$$

The values of these 4 evaluation metrics are in the range of [0,1] and have a direct ratio with the recommendation performance.

Table 2. Experimental Results of Web API Recommendation Among Competing Approaches.

Method	Precision	Recall	F1	Hit Ratio
SingleBPR	0.1691	0.5655	0.2603	0.5655
LSTM	0.2661	0.4207	0.3260	0.4207
SPR	0.4722	0.6501	0.5471	0.6501
Pop	0.3887	**0.7079**	0.5019	**0.7079**
MTFM	0.6164	0.5452	0.5786	0.5452
AWAR	**0.7368**	0.6260	**0.6769**	0.6260

4.3 Competing Methods

To evaluate the performance of AWAR, it is compared with 5 representative approaches, which are listed as follows.

- SingleBPR[12]: BPR is effectively employed in Web API recommendation by predicting user preferences. It learns user behavior through pairwise comparisons of API interactions. This enables the model to recommend APIs that match user interests closely, ensuring high preference scores and improving recommendation accuracy.
- LSTM[13]: LSTM is commonly used to track sequential features when recommending Web APIs. By capturing long-term dependencies, LSTM delivers precise and relevant Web API recommendations that adapt to changing user preferences, improving the personalization and efficiency of web service discovery.
- SPR[14]: The SPR model applies the author-topic model to learn how mashup descriptions relate to APIs. By analyzing the topical structure of these descriptions, SPR produces Web API recommendations that are more relevant and accord with user's requirement, enhancing the precision of results.
- Pop[15]: The Pop model recommends services based on their popularity to users. In mashup creation, this popularity is quantified by the number of category-aware invocations. APIs with higher usage within these categories are prioritized, ensuring that the most frequently used APIs are recommended, thereby enhancing the relevance and reliability of the recommendations.
- MTFM[16]: MTFM model integrates multi-model fusion and multi-task learning to enhance prediction accuracy. The inclusion of a semantic component to represent service requirements improves the API recommendation results.

4.4 Experiment Results and Analyses

The performance of AWAR is validated by comparing it with five competing baselines. Since AWAR can adaptively adjust the recommended number of Web APIs instead of a predefined one, we initially compare its performance with the best performance of each competing approach, regardless of the recommended number K.

Experimental results of Web API recommendation among competing approaches are provided in Table 2.

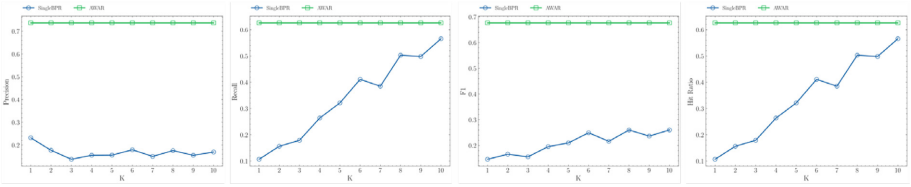

Fig. 5. Web API Recommendation Performance of SingBPR with Different K and AWAR.

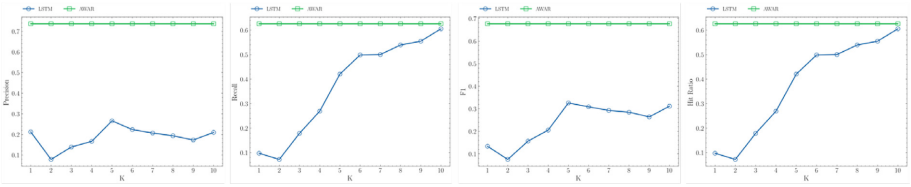

Fig. 6. Web API Recommendation Performance of LSTM with Different K and AWAR.

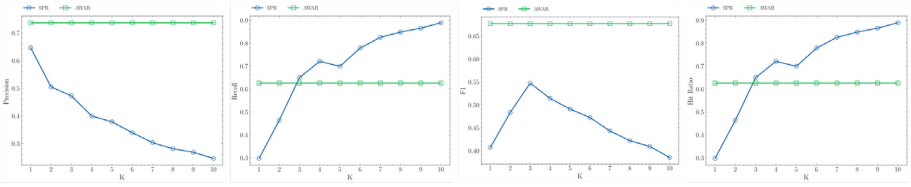

Fig. 7. Web API Recommendation Performance of SPR with Different K and AWAR.

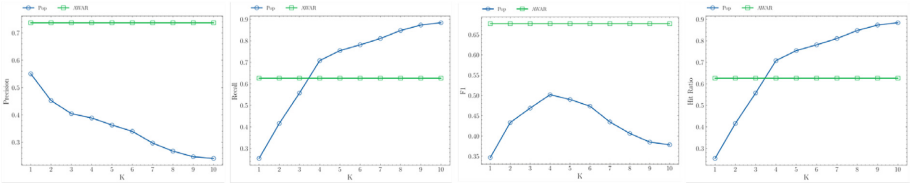

Fig. 8. Web API Recommendation Performance of Pop with Different K and AWAR.

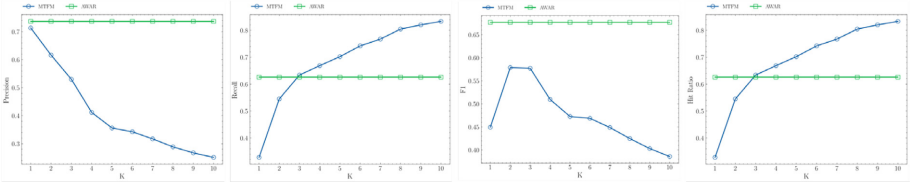

Fig. 9. Web API Recommendation Performance of MTFM with Different K and AWAR.

From the results, it is observed that Pop achieves the highest recall and hit ratio, followed by SPR, but their precisions and F1-scores are not competitive. AWAR exhibits the highest precision and F1-score, while its recall and hit ratio rank are not the best but also quite high. Overall, AWAR outperforms the five comparative approaches significantly on precision and F1-score as well as competitive in recall and hit ratio.

Table 3. Comparison Results of Web API Recommendation in Ablation Experiments.

Method	Precision	Recall	F1	Hit Ratio
0–1	0.3621	0.3262	0.3432	0.3270
MLP	0.7111	0.5289	0.6066	0.5289
AWAR	**0.7368**	**0.6260**	**0.6769**	**0.6260**

Specifically, the recommendation performance among competing baselines with different numbers of K are shown in Figs. 5, 6, 7, 8 and 9. SingleBPR and LSTM share similar results compared to AWAR. When $K = 10$ and $K = 5$, SingleBPR and LSTM are observed to have the best performance because their F1 score reaches the maximum values. For SingleBPR, LSTM and SPR, they achieve comparable accuracy of Web API recommendation. The advantage of SingleBPR lies in direct ranking optimization, while LSTM lies in the superior encoding structure of bidirectional LSTM with an attention mechanism. The advantage of SPR is that the hidden topic layer can overcome the disadvantages of the bag-of-words model and semantically connect the requirements with the APIs.

Additionally, the results of SPR, Pop, and MTFM are similar to each other compared to AWAR. Their recall and hit ratio outperform AWAR, but precision and F1 score are lower. Compared with all the competing baselines, MTFM has the advantages of both CF-based and content-based approaches. AWAR outperforms them significantly on all the accuracy indicators.

Since other 3 comparison methods are not self-adaptive, their recall and hit ratio may outperform AWAR when K is set to a large number because the more candidate services there are, the more correct services will be recommended. However, most mashup services generally apply a small number of Web services, as our proposed AWAR adaptively recommends a small number of Web APIs. In addition, Ks of the comparison methods are artificially set after a large number of experiments for optimal results, but in real application scenarios, setting K in advance is not feasible because the services in the repository are not fixed. A conclusion can be drawn that from the perspective of self-adaptation, AWAR can make recommendations more stably by better meeting users' requirements and is more effective in practical use.

4.5 Performance Impact of Parameters

Ablation experiments are conducted to validate the effectiveness of the two parts of AWAR. In the first part, service clusters the combination of which is most similar to the

target mashup service are preliminary filtered out. Then in the second part, the result is improved according to invocation history. The results are shown in Table 3. From the results, it can be seen that the 0–1 planning and MLP can recommend Web services for mashup creation separately, and the performance of MLP is better than that of 0–1 planning. Additionally, the combination of the two parts has the best performance of all four evaluation metrics. As a result, it indicates that the two components in the AWAR framework can effectively make recommendations for mashup creation.

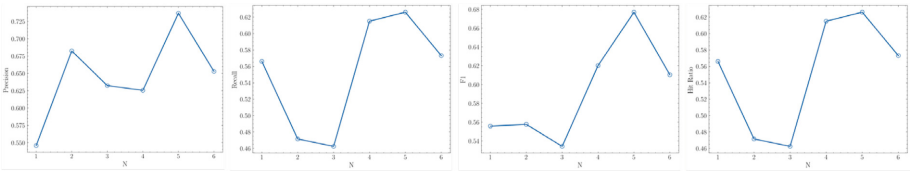

Fig. 10. The Impact of the Number of Similar Mashup Services N.

Table 4. Web API Recommendation Performance with Different Learning Rates.

Method	Precision	Recall	F1	Hit Ratio
LR = 0.01	**0.8916**	0.3058	0.4554	0.3058
LR = 0.001	0.7798	0.4920	0.6034	0.4921
LR = 0.0001	0.7368	**0.6260**	**0.6769**	**0.6260**

Table 5. Web API Recommendation Performance with Different Gradient Descent Algorithms.

Method	Precision	Recall	F1	Hit Ratio
SGD	**0.8421**	0.1882	0.3077	0.1882
Adam(LR = 0.0001)	0.7368	**0.6260**	**0.6769**	**0.6260**
Adam(LR = 0.001)	0.8346	0.4398	0.5761	0.4398
Adam(LR = 0.01)	0.6704	0.4878	0.5647	0.4878

In the experiments, as the number of similar mashup services changes for the input of MLP, the comparison results are shown in Fig. 10. According to the results, the value of the four evaluation metrics reaches its maximum when the input of MLP contains the five similar mashup services. Thus, AWAR has the best Web API recommendation performance when the number of similar mashup services is set as 5.

Simultaneously, the learning rate also affects the performance of Web API recommendation. Table 4 shows the recommendation results with different learning rates. Although the model achieves the highest precision when learning rate is set as 0.01, its recall and F1-score are lower than the others. From the overall consideration, learning rate is set as 0.0001 because it has the highest values of recall, F1-score, and hit ratio.

Although the precision is the lowest compared with the other learning rates, it is still relatively high compared to the competing baselines.

In addition to learning rate, different gradient algorithms used in MLP are evaluated in the experiments. From the results of Table 5, the SGD gradient algorithm has the highest precision among the four algorithms, while the values of the other three evaluation metrics are much lower. In contrast, Adam has the highest recall, F1-score, and hit ratio, while its precision is low. Therefore, the Adam gradient algorithm is chosen, when it is applied to perform adaptive Web APIs for its superior performance on multiple evaluation metrics.

5 Related Work

In recent years, with the continuous development of Web services, API recommendation for mashup creation have gradually become a research hotspot. Many researchers are committed to studying how to recommend Web services to mashup developers, aiding them in quickly building their desired mashup services. Currently, service recommendations for mashup creation can be divided into four categories: quality-based service recommendations, collaborative filtering-based service recommendations, social relationship-based service recommendations, and function-based service recommendations.

Quality-based service recommendation often involves recommending potentially high-quality services to developers after they have identified a set of required services [17]. Reference [18] utilized temporal information to refine similarity measurements in neighborhood-based CF (Collaborative Filtering). Location information was also integrated to cluster services and users, enabling personalized recommendations based on clustering results [19]. Paper [20] proposed a hidden Markov model-based approach to assist users in locating services with optimal response times for their requests. Additionally, Luo et al. [21] introduced the BNLFTs model regarding the issue of QoS data fluctuating over time. Biased non-negative latent factorization of tensors were utilized for predicting Quality of Service (QoS) in a temporal context. Wang et al. [22] introduced a novel method extending the original Graphplan form and incorporating branch structures into composite solutions to address uncertainty in the service composition process.

Collaborative filtering-based service recommendation typically recommends services to users based on service composition and call records. Jiang et al. [23] introduced an effective personalized collaborative filtering method for Web service recommendation. When calculating user similarity, this method considers the personalized impact of services rather than solely relying on the Pearson correlation coefficient (PCC). Liang et al. [24] proposed a recommendation algorithm for secure collaborative filtering services. The algorithm integrated deep neural networks and content similarity modules to alleviate the problem of data sparsity in Mashup Web service matrix using traditional collaborative filtering algorithms. In [25], the authors focused on the problem of data sparsity and cold start and proposed a service recommendation model. It combines knowledge graph for uncovering potential relationships and collaborative filtering for API recommendations.

Social relationship-based service recommendation often uses external information, such as the social connections among developers, to recommend services. In [26], the authors studied service usage patterns in an evolving service repository called myExperiment, proposing a recommendation algorithm based on service correlations within the network. Wei et al. [27] proposed the Social-powered Graph Hierarchical Attention Network (SGHAN) using a service-level attentional encoder to identify significant services in friends' mashups and a friend-level graph attention network to prioritize and propagate the social influences of friends. Additionally, [28] outlined a service recommendation approach grounded in link prediction within a dynamic service co-occurrence network. Recent work has applied association mining techniques over service networks to discern positive and negative collaboration patterns among services.

Function-based service recommendation aims to recommend services that best fulfill the functional requirements of mashup developers, thereby alleviating the burden of service discovery. Wu et al. [29] introduced a probabilistic model for suggesting services to mashup developers, applying a topic model to represent mashup requirements, APIs, and their relationships. Xia et al. [30] proposed a distributed recommendation method comprising two steps: service clustering and distributed service recommendation, which help narrow the search scope of mashup developers' service discovery. Its notable feature is the recommendation of multiple sorting lists to users, with each list representing a category of service sets, rather than recommending all services to developers at once. Gao et al. [3] presented a service recommendation framework to suggest sets of services to developers, where each service set can be viewed as a potential mashup rather than a single service. The framework consists of two stages: service clustering and service set sorting. Gu et.al. [31] build a semantic service package repository where the service packages are denoted with compositional semantics by mining mashup and then recommend Web APIs based on the repository.

6 Conclusion and Future Work

In this paper, we are dedicated to adaptively recommending Web API into service recommendation tailored for mashup creation. The proposed SASSBR focuses on the matching process between mashup requirements and candidate API services, improving recommendation effectiveness through the integration of natural language processing, optimization algorithms, and deep learning, where 0–1 planning and multi-layer perceptron are applied for accurately performing adaptive Web API recommendation. We conduct a series with competing baselines to assess the performance of AWAR, including comparative analyses of methods and the impact of parameters within our proposed recommendation approach.

In the future work, we aim to further explore the latest techniques to advance Web API recommendations by refining the alignment between the descriptions of Web services and mashup requirements. We also focus on improving the way of calculating similarity between mashup services and aggregated services by utilizing neural network. Moreover, we intend to further optimize the feature extraction method for aggregated service and API services.

Acknowledgments. This work was supported by National Natural Science Foundation of China (No. 62272290, 62172088).

References

1. Al-Masri, E., Mahmoud, Q.H.: Investigating Web services on the world wide Web. In: International Conference on World Wide Web, pp.795–804 (2008)
2. Fichter, D.: What Is a Mashup, http://books.infotoday.com/books/Engard, last accessed 2013/12/8
3. Gao, W., Wu, J.: A novel framework for service set recommendation in mashup creation. In: 2017 IEEE International Conference on Web Services (ICWS), pp. 65–72. (2017)
4. David, B., Hugo, H., Francis, M., et al.: Web Services Architecture, https://www.w3.org/TR/ws-arch/, last accessed 2024/7/30
5. Gao, W., Chen, L., Wu, J., et. al.: Manifold-learning based API recommendation for mashup creation. In: 2015 IEEE International Conference on Web services, pp.432–439 (2015)
6. Zhong, Y., Fan, Y., Huang, K., et al.: Time-aware service recommendation for mashup creation. IEEE Trans. Serv. Comp. **8**(3), 356–368 (2014)
7. Yao, L., Wang, X., Sheng, Q.Z., et al.: Mashup recommendation by regularizing matrix factorization with API co-invocations. IEEE Trans. Serv. Comp. **14**(2), 502–515 (2018)
8. Man, K.F., Tang, K.S., Kwong, S.: Genetic algorithms: concepts and applications in engineering design. IEEE Trans. Ind. Electron. **43**(5), 519–534 (1996)
9. Nielsen, M.A.: Neural networks and deep learning. Determination press, San Francisco, CA, USA (2015)
10. Goodfellow, I., Bengio, Y., Courville, A.: Deep learning. MIT press, US (2016)
11. Ma, Y., Geng, X., Wang, J., He, K., Athanasopoulos, D.: Deep learning framework for multi-round service bundle recommendation in iterative mashup development. CAAI Trans. Intell. Technol. **8**(3), 914–930 (2023)
12. Rendle, S., Freudenthaler, C., Gantner, Z., Schmidt-Thieme, L.: BPR: Bayesian personalized ranking from implicit feedback. In: 25th Conference on Uncertainty in Artificial Intelligence, UAI 2009, pp. 452–461 (2009)
13. Shi, M., Tang, Y., Liu, J.: Functional and contextual attention-based LSTM for service recommendation in mashup creation. IEEE Trans. Parall. Distributed Syst. **3**(5), 1077–1090 (2019)
14. Zhong, Y., Fan, Y., Tan, W., et al.: Web service recommendation with reconstructed profile from mashup descriptions. IEEE Trans. Autom. Sci. Eng. **15**(2), 468–478 (2016)
15. Rocío, C., Pablo C.: Should I Follow the Crowd? A Probabilistic Analysis of the Effectiveness of Popularity in Recommender Systems. In: The 41st International ACM SIGIR Conference on Research & Development in Information Retrieval, pp.415–424 (2018)
16. Wu, H., Duan, Y., Yue, K., Zhang, L.: Mashup-oriented web API recommendation via multi-model fusion and multi-task learning. IEEE Trans. Serv. Comp. **15**(6), 3330–3343 (2021)
17. Zheng, Z., Li, X., Tang, M., Xie, F., Lyu, M.R.: Web service QoS prediction via collaborative filtering: A survey. IEEE Trans. Serv. Comp. **15**(4), 2455–2472 (2020)
18. Zhong, Y., Fan, Y., Tan, W., Zhang, J.: Web service recommendation with reconstructed profile from mashup descriptions. IEEE Trans. Autom. Sci. Eng. **15**(2), 468–478 (2016)
19. Liu, Y., Cao, J.: API-Prefer: An API Package Recommender System Based on Composition Feature Learning. In: International Conference on Service-Oriented Computing, pp. 500–507. Springer, Cham (2020)
20. Li, C., Zhang, R., Huai, J., Sun, H.: A novel approach for API recommendation in mashup development. In: 2014 IEEE International Conference on Web Services, pp. 289–296 (2014)

21. Luo, X., Wu, H., Yuan, H., Zhou, M.: Temporal Pattern-Aware QoS Prediction via Biased Non-Negative Latent Factorization of Tensors. IEEE Trans. Cybernet. **50**(5), 1798–1809 (2020)
22. Wang, P., Ding, Z., Jiang, C., Zhou, M., Zheng, Y.: Automatic Web service composition based on uncertainty execution effects. IEEE Trans. Serv. Comp. **9**(4), 551–565 (2015)
23. Jiang, Y., Liu, J., Tang, M., Liu, X.: An effective Web service recommendation method based on personalized collaborative filtering. In: 2011 IEEE International Conference on Web Services, pp. 211–218 (2015)
24. Liang, W., Xie, S., Cai, J., Xu, J., Hu, Y., Xu, Y., Qiu, M.: Deep neural network security collaborative filtering scheme for service recommendation in intelligent cyber–physical systems. IEEE Internet of Things Journal **9**(22), 22123–22132 (2021)
25. 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 **9**, 50880–50892 (2021)
26. Li, H., Liu, J., Cao, B., Tang, M., Liu, X., Li B.: Integrating Tag, Topic, Co-Occurrence, and Popularity to Recommend Web APIs for Mashup Creation. In: 2017 IEEE International Conference on Services Computing (SCC), pp. 84–91. Honolulu, HI (2017)
27. Wei, C., Fan, Y., Zhang, J.: Time-aware service recommendation with social-powered graph hierarchical attention network. IEEE Trans. Serv. Comp. **16**(3), 2229–2240 (2022)
28. Gu, Q., Cao, J., Peng, Q.: Service package recommendation for mashup creation via mashup textual description mining. In: 2016 IEEE International Conference on Web Services (ICWS), pp. 452–459 (2016)
29. Wu, H., Duan, Y., Yue, K., Zhang, L.: Mashup-oriented web API recommendation via multi-model fusion and multi-task learning. IEEE Trans. Serv. Comp. **15**(6), 3330–3343 (2021)
30. 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. Comp. **8**(5), 674–687 (2014)
31. Gu, Q., Cao, J., Liu, Y.: CSBR: A compositional semantics-based service bundle recommendation approach for mashup development. IEEE Trans. Serv. Comp. **15**(6), 3170–3183 (2021)