LMSR: LLM-enhanced Multi-perspective Service Feature Learning for Web API Recommendation

Journal:	*Transactions on Services Computing*
Manuscript ID	TSC-2025-04-0497
Manuscript Type:	Regular Paper
Keywords:	M Services Computing, M.3 Web Services < M Services Computing, M.11.3.b Service Mash-up < M.11.3 Software as Services < M.11 Services Delivery Platform and Methodology < M Services Computing

LMSR: LLM-enhanced Multi-perspective Service Feature Learning for Web API Recommendation

Song Yang, Guobing Zou*, Shengxiang Hu, Shengye Pang*, Yanglan Gan, Bofeng Zhang, and Yixin Chen, *Fellow, IEEE*

***Abstract*—Web APIs have become a fundamental paradigm in the Web 4.0 era, with mashup services emerging as a transformative technology that combines multiple APIs to create comprehensive services. However, existing Web API recommendation approaches exhibit two significant limitations: overlooking the quality and completeness of recommendation contexts of new mashup requirements, and failing to effectively extract high-quality collaborative features from multi-perspective service relationships between mashup requirements and APIs. To address these limitations, this paper proposes LMSR, a novel LLM-enhanced Multi-perspective Service Feature Learning framework for Web API Recommendation. LMSR first leverages pre-trained LLMs to refine and encode the original requirement descriptions, and extracts context enhancement features across similar service context prediction tasks, enabling accurate prediction of mashup requirement categories and relevant APIs, thus enhancing the recommendation context of mashup requirements. Furthermore, by integrating the refined mashup descriptions, predicted categories, and relevant APIs into the LLM through fine-tuning, LMSR effectively extracts collaborative features from multi-perspective service relationships between mashup requirements and APIs, ultimately achieving precise Web API recommendation. Comprehensive experiments on real-world datasets demonstrate that LMSR significantly outperforms 11 baseline approaches across precision, recall, F1-score, and NDCG, validating its effectiveness in Web API recommendation.**

***Index Terms*—Web API recommendation, mashup creation, mashup requirement context, collaborative service feature learning.**

I. Introduction

As a fundamental paradigm of the Web 4.0 era, Web APIs have witnessed unprecedented proliferation in both diversity and scale over recent years, establishing themselves as an integral component of the contemporary service-oriented society [1] [2]. However, the increasing sophistication of user requirements has made individual Web API insufficient to address the multifaceted service demands of modern Web users. Moreover, with the continuous advancement of AI multi-agent applications, the integration of diverse Web tools for comprehensive functional execution has attracted significant research attention. Consequently, mashup services, which combine existing Web APIs in specific ways to create new comprehensive services, have emerged as a highly transformative network service technology, and have been widely applied in various sectors of the service industry [3].

Nevertheless, the identification and selection of appropriate Web APIs from an extensive repository for mashup service creation remains a labor-intensive and resource-demanding challenge for service developers. For instance, ProgrammableWeb (PW), the largest online service registry platform, has registered 22,611 Web APIs and 6,419 mashup services by March 2022 [4]. Given the vast scale of available Web APIs, it is impractical for developers to comprehensively understand the functionalities of each service. Consequently, automatically and precisely recommending appropriate Web APIs based on the developers' requirement descriptions, thereby facilitating efficient and effective mashup service construction, has emerged as a critical research direction in service computing [5].

In recent years, Web API recommendation has drawn significant attention from researchers [6]–[17]. Current research in Web API recommendation can be primarily classified into four categories: collaborative filtering (CF)-based approaches [11], [15], content-based approaches [8], [13], deep learning-based approaches [2], [4], [9], and Large language Model (LLM)-based approaches [16], [17]. Both CF-based and content-based approaches often show limited performance in Web API recommendation, as they focus solely on either the mashup requirement description or the historical invocation records. In comparison, deep learning-based approaches can effectively utilize large-scale service data to achieve better recommendation results. However, the limited feature extraction capabilities of deep neural networks constrain the further enhancement of recommendation performance in deep learning-based approaches. Consequently, researchers [16], [17] have leveraged the powerful semantic understanding and feature extraction capabilities of LLMs for Web API recommendation, achieving state-of-the-art (SOTA) recommendation performance.

While substantial advancements have been achieved, existing Web API recommendation approaches face two significant limitations. First, existing approaches primarily focus on historical mashups while overlooking the quality of new mashup requirements and the completeness of their recommendation context. Specifically, due to the cold-start nature of API recommendation, new mashup requirements typically contain only requirement descriptions without categories or

Song Yang, Guobing Zou, Shengxiang Hu and Shengye Pang are with the School of Computer Engineering and Science, Shanghai University, Shanghai, China. Email: yangsong@shu.edu.cn; gbzou@shu.edu.cn; shengxianghu@shu.edu.cn; pangsy@shu.edu.cn.

Yanglan Gan is with the School of Computer Science and Technology, Donghua University, Shanghai 201620, China. Email: ylgan@dhu.edu.cn.

Bofeng Zhang is with the School of Computer and Information Engineering, Shanghai Polytechnic University, Shanghai, China. Email: bfzhang@sspu.edu.cn.

Yixin Chen is with the Department of Computer Science and Engineering, Washington University in St. Louis, St. Louis, MO 63130 USA. E-mail: chen@cse.wustl.edu.

* Corresponding author

invoked APIs, making it challenging for existing approaches to leverage categorical relationships and historical invocation relationships for API recommendation. Furthermore, the lack of standardization in service descriptions results in substantial recommendation-irrelevant redundant information contained in requirement description, severely affecting the extraction of high-quality semantic features. Second, existing approaches fail to effectively extract collaborative features from multi-perspective service relationships between mashup requirements and Web APIs. On one hand, due to the inability to accurately predict mashup categories and relevant APIs, existing approaches struggle to simultaneously consider multi-perspective service relationships, including semantic, categorical, and invocation relationships during recommendation. On the other hand, constrained by the scale and generalization capabilities of feature extraction models, existing approaches fail to extract high-quality collaborative features from these multi-perspective service relationships, thereby significantly compromising the performance of Web API recommendation.

To address the limitations in existing Web API recommendation approaches, we propose **LMSR**, a novel **L**LM-enhanced **M**ulti-perspective **S**ervice Feature Learning framework for Web API **R**ecommendation. LMSR consists of two primary modules: LLM-based mashup requirement recommendation context enhancement, and multi-perspective collaborative service feature extraction and recommendation by fine-tuning LLM. First, we leverage a pre-trained LLM combined with prompt engineering to refine the original requirement descriptions. Subsequently, we encode the refined descriptions using the LLM and propose a neural network based on differential attention [18] and mixture of service experts (MoSE) to extract generic context enhancement features across various similar recommendation contexts. Finally, through external category prediction and relevant API identification layers, we accurately predict the associated categories and relevant APIs for mashup requirements, thereby enhancing and completing multiple contexts necessary for mashup requirement recommendation. Furthermore, we design a specialized prompt that integrates the obtained mashup categories and relevant APIs along with refined descriptions into the LLM, enabling simultaneous consideration of multi-perspective service relationships between mashup and Web APIs for collaborative feature extraction. Additionally, we employ Q-LoRA [19] technique to fine-tune the pre-trained LLM and replace its generative output layer with a Web API recommendation layer, enabling the LLM to effectively and accurately extract collaborative service features from multi-perspective service relationships and achieve precise Web API recommendation.

To validate LMSR's effectiveness, we conduct comprehensive experiments on real-world datasets comprising 8,420 mashups and 21,614 Web APIs. Through extensive comparisons with eleven baseline approaches, including current SOTA approaches, across four standard metrics (precision, recall, MAP, and NDCG), we demonstrate that LMSR consistently achieves superior performance in Web API recommendation, significantly outperforming all existing approaches. In summary, the main contributions of this paper can be outlined as follows:

- We propose LMSR, a novel LLM-enhanced multi-perspective service feature learning framework for Web API recommendation. Initially, LMSR employs the pre-trained LLM and MoSE context enhancement feature extraction network to address the low-quality and missing recommendation contexts of new mashup requirements. Specifically, through requirement description refinement, category prediction, and relevant API identification, LMSR provides comprehensive recommendation contexts for mashup requirements to facilitate subsequent recommendation.
- We integrate refined descriptions, categories and relevant APIs into the LLM through a specially designed prompt. Furthermore, we fine-tune the LLM and replace its output layer with a Web API recommendation layer, enabling effective extraction of collaborative service features from multi-perspective service relationship between mashup requirements and Web APIs, ultimately achieving precise Web API recommendation.
- We conduct comprehensive experiments on real-world large-scale datasets to validate LMSR's effectiveness. Through extensive comparisons with eleven baseline approaches across four standard metrics, we demonstrate that LMSR consistently achieves superior performance in Web API recommendation, significantly outperforming all existing approaches.

The remainder of this paper is organized as follows. Section II is the formulation of the Web API recommendation problem for mashup requirement. Section III illustrates the overall framework of LMSR. Section IV depicts the LMSR approach in detail. Section V presents the experimental results and detailed experiment analysis. Section VI is related work. Finally, Section VII concludes this paper and discusses the future work.

II. Problem Formulation

In this section, we introduce the relevant definitions of the Web API recommendation problem addressed in this paper. First, the definition of Web API used in this paper is as follows.

Definition 1 (Web API). *The Web API s can be represented as a triple-tuple $s = \langle D_s, N_s, C_s \rangle$, where D_s, N_s, C_s is the function description, name and category of Web API s, respectively. Furthermore, the Web API repository $\mathbb{S}$ can be represented as a set $\mathbb{S} = \{s_1, s_2, \ldots, s_p\}$, where $s_i \in \mathbb{S}$ is a Web API and p is the scale of the API repository.*

Given the Web API repository $\mathbb{S}$, a mashup service can be created by the Web API within $\mathbb{S}$ and defined as follow.

Definition 2 (Mashup Service). *A mashup service m can be depicted as a quadruple $m = \langle D_m, N_m, C_m, \mathbb{S}_m \rangle$, where D_m, N_m, C_m is the function description, name and category of mashup service m, respectively, $\mathbb{S}_m$ is the set of Web API invoked by m. Additionally, the mashup repository is defined as $\mathbb{M} = \{m_1, m_2, ..., m_n\}$, each $m_i \in \mathbb{M}$ is a mashup service.*

Based on the definition of API and mashup, the invocation relation $\mathbb{I}$ between mashups and APIs can be acquired.

Definition 3 (Invocation Relationship between Mashups and APIs). *The invocation relationship between mashups and*

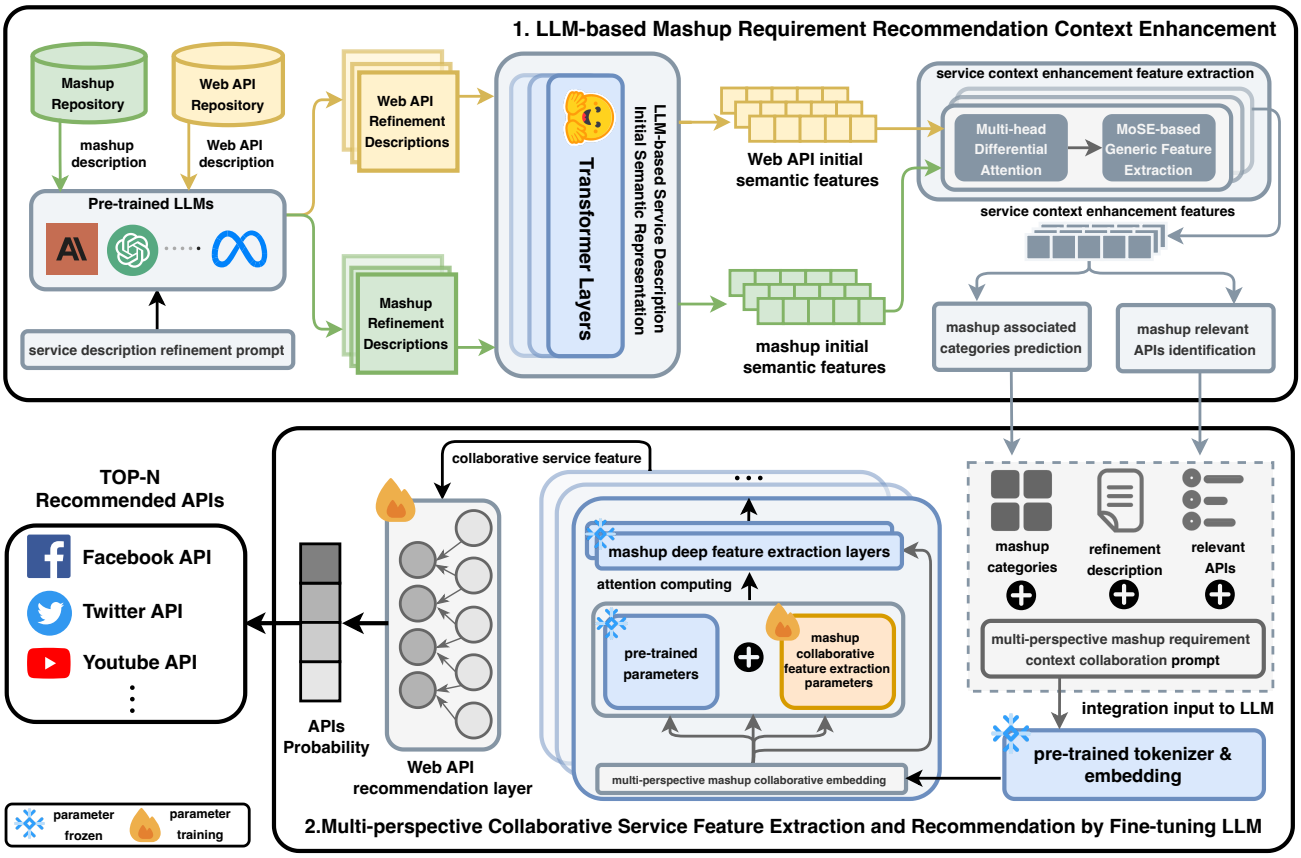


Fig. 1. The overall architecture of LMSR consists of two modules: LLM-based mashup requirement recommendation context enhancement, and multi-perspective collaborative service feature extraction and recommendation by fine-tuning LLM.

APIs can be described as a matrix $\mathbb{I}^{n\times p} = \{I_1^p, I_2^p, \ldots, I_n^p\}$, *where* $I_j^p \in \mathbb{I}^{n\times p}$ *is a row vector with scale of* p, *representing the invocation relationship of mashup* m_j *across all* p *APIs. In this paper,* I_j^p *is defined as a one-hot vector. Specifically, if a certain Web API* s_i *is contained in* $\mathbb{S}_{m_j}$, *then the* i*-th position in* I_j^p *is 1, otherwise it is 0.*

Unlike existing mashups, a mashup requirement refers to a user-submitted mashup with incomplete information.

Definition 4 (Mashup Requirement). *A mashup requirement* m^r *can be defined as a binary tuple* $m^r = \langle D_{m^r}, N_{m^r}\rangle$, *where* D_{m^r} *and* N_{m^r} *denote the mashup requirement description and name provided by the user, respectively.*

To enhance recommendation accuracy, a complete mashup requirement recommendation context should be constructed.

Definition 5 (Mashup Requirement Recommendation Context). *Mashup requirement recommendation context comprises various service context essential for API recommendation, such as requirement description* D_{m^r}, *associated categories* C_{m^r}, *and a list of potentially invoked relevant APIs* $\mathbb{S}^c_{m^r} \subset \mathbb{S}$.

It should be noted that the relevant APIs $\mathbb{S}^c_{m^r}$ only partially capture the mashup's invocation relationship and will not be considered as the final recommended API list. Synthesizing the aforementioned definitions, the Web API recommendation problem for mashup requirement can be defined as follows.

Definition 6 (Web API Recommendation Problem for Mashup Requirement). *The Web API recommendation problem* Ω *can be defined as a six-tuple,* $\Omega = \langle \mathbb{S}, \mathbb{M}, \mathbb{I}, m^r, \mathbb{S}^c_{m^r}, C_{m^r}\rangle$, *where* m^r *is the newly added mashup requirement needed to be recommended Web APIs.* $\mathbb{S}^c_{m^r}$ *is the predicted relevant API set of* m^r, *and* C_{m^r} *is the predicted mashup category of* m^r. *The objective of the Web API recommendation problem is to find a set of Web APIs* $\mathbb{S}_{m^r} \subseteq \mathbb{S}$ *from Web API repository for* m^r, *such that the functional requirements of* m^r *can be satisfied by the Web APIs in* $\mathbb{S}_{m^r}$.

III. The Framework of LMSR

As shown in Fig. 1, LMSR consists of two modules: LLM-based mashup requirement recommendation context enhancement, and multi-perspective collaborative service feature extraction and recommendation by fine-tuning LLM. The process of each modules in LMSR is described as below.

- In the module of LLM-based mashup requirement recommendation context enhancement, we first propose a specially designed prompt, which guides the LLM to perform step-by-step inferring. First, the LLM analyzes the correlation of each sentence in the mashup requirement description to the API recommendation task. Then, it removes statements from the original requirement description not highly related to the recommendation task, ultimately outputting a more concise refined mashup requirement description. Furthermore, we transform the refinement descriptions into initial semantic features through a LLM-based encoder model. After this, we employ a MoSE network with multi-head differential

attention to learn generic context enhancement features across multiple similar service contexts. Finally, we feed the context enhancement features into task-specific layers to obtain mashup categories and relevant APIs for further recommendation.

- In the module of multi-perspective collaborative service feature extraction and recommendation by fine-tuning LLM, we first integrate the previously predicted mashup categories, relevant APIs and refined requirement description through a specially designed prompt as input for the LLM. Subsequently, we utilize the LLM's pre-trained tokenizer and embedding layer to transform the multi-perspective service collaborative context into embeddings. Furthermore, by fine-tuning LLM with Q-LoRA techniques, we efficiently integrate multi-perspective service collaborative features extraction parameters into LLM's pre-trained parameters, enabling the LLM to extract high-quality collaborative service features. Finally, we replace the LLM's original generative output layer to achieve precise Web API recommendation.

IV. Approach

A. LLM-based Mashup Requirement Recommendation Context Enhancement

1) Service Description Refinement and Initial Semantic Representation: Service description is the most fundamental semantic information for Web API recommendation. However, it often contains substantial statements irrelevant to functionality or requirements. These supplementary contents make service descriptions verbose and introduces additional noise, making it challenging to extract high-quality semantic features. Therefore, we employ pre-trained LLMs combined with prompt engineering [20] to refine the service descriptions.

Specifically, given the service description with n words $\mathcal{D} = \{w_1, w_2, \ldots, w_n\}$ and the prompt $\mathcal{T}$, the input of LLM can be represented as follow:

$$X = \text{Splice}(\mathcal{T}, \mathcal{D}) \tag{1}$$

where Splice$(\cdot)$ is operation of text concatenation. Given an LLM $\mathcal{M}$, after incorporating the input X with a task prompt, $\mathcal{M}$ generates a word sequence $Y = \{y_1, y_2, \ldots, y_m\}$ of length m. Each $y_i \in Y$ is generated as follow:

$$\begin{aligned} Y &= \mathcal{M}(X) = \{y_1, y_2, \ldots, y_m\} \\ y_i &= \arg\max_{w_i} P(w_i | X, y_1, y_2, ..., y_{i-1}), w_i \in \mathbb{W} \end{aligned} \tag{2}$$

where $\mathbb{W}$ is the vocabulary of $\mathcal{M}$.

Therefore, to refine service descriptions, we design a Chain-of-Thought (CoT)-based Semantic Refinement Prompt (CSRP) for service descriptions, which guides the LLM through step-by-step reasoning and ultimately generates concise and well-structured service descriptions

In detail, Fig. 2 presents a case of using CSRP to refine a mashup description. In the system prompt of CSRP, we first designate the LLM's role as a service description reconstruction system. Subsequently, we employ a CoT-based prompt chaining technique to decompose the complex service description refinement problem into three sequential subtasks.

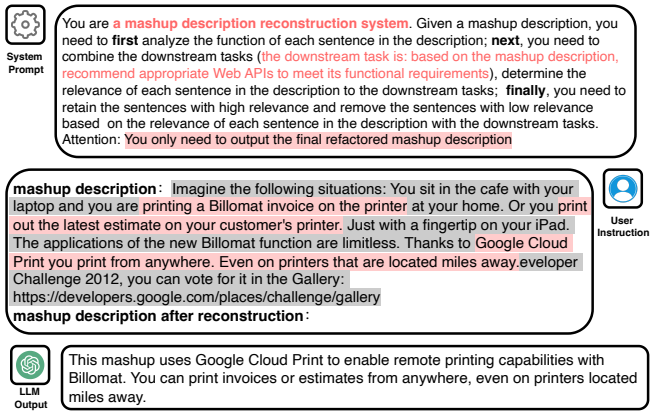


Fig. 2. A case study of mashup description refinement with CSRP.

First, CSRP guides the LLM to analyze the function of each sentence in the description. Then, based on the sentence function analysis results, CSRP directs the LLM to evaluate the relevance of each sentence to the downstream Web API recommendation task. Finally, based on this relevance analysis, CSRP instructs the LLM to retain highly relevant sentences, remove irrelevant or low-relevance sentences, and reconstruct the service description, ultimately producing the semantically refined service description.

Observing the output of LLM in Fig. 2, it can be noted that, with the instruction of CSRP, the LLM is able to reconstruct and remove redundant information irrelevant to Web API recommendation, thereby generating high-quality and more concise service descriptions. Generally, the mashup description D'_m semantically refined by the LLM in conjunction with CSRP is obtained as follows:

$$D'_m = \mathcal{M}(\text{Splice}(\mathcal{T}_{CSRP}, D_m)) \tag{3}$$

where $\mathcal{T}_{CSRP}$ denotes the specific content of the CSRP, and D_m is the original mashup description. Similarly, given a Web API description D_s, the semantically refined description D'_s can be obtained as follows:

$$D'_s = \mathcal{M}(\text{Splice}(\mathcal{T}_{CSRP}, D_s)) \tag{4}$$

Following the acquisition of service refined descriptions, we leverage pre-trained LLM [21] $\mathcal{M}$ to obtain high-quality service initial semantic features. Taking mashup descriptions as an instance, for a refined description D'_{m_i} of mashup m_i in mashup repository $\mathbb{M}$, each token in D'_{m_i} are converted into corresponding token embeddings through tokenizer and embedding components of $\mathcal{M}$, as shown in equation (5).

$$\mathbf{V}_{m_i} = [\mathbf{v}_{[\text{PAD}]}, \ldots, \mathbf{v}_{[\text{PAD}]}, \mathbf{v}_1, \mathbf{v}_2, \ldots, \mathbf{v}_n] \tag{5}$$

Then, the initial embedding matrix $\mathbf{V_{m_i}}$ is input into $\mathcal{M}$. Ultimately, the output of the final hidden layer in $\mathcal{M}$, will be used as the initial semantic representation for the service description, as illustrated in Equation (6).

$$\begin{aligned} \mathbf{F}_{m_i} &= \mathbf{H}_{(-1)}(\mathbf{V}_{m_i}) \\ \mathbf{H}_{(k)}(V) &= \mathbf{W}^{\mathbf{k}}{}_{\mathcal{M}(\mathbf{V})} \end{aligned} \tag{6}$$

Similarly, we obtain the service initial semantic feature $\mathbf{F}_{s_j}$

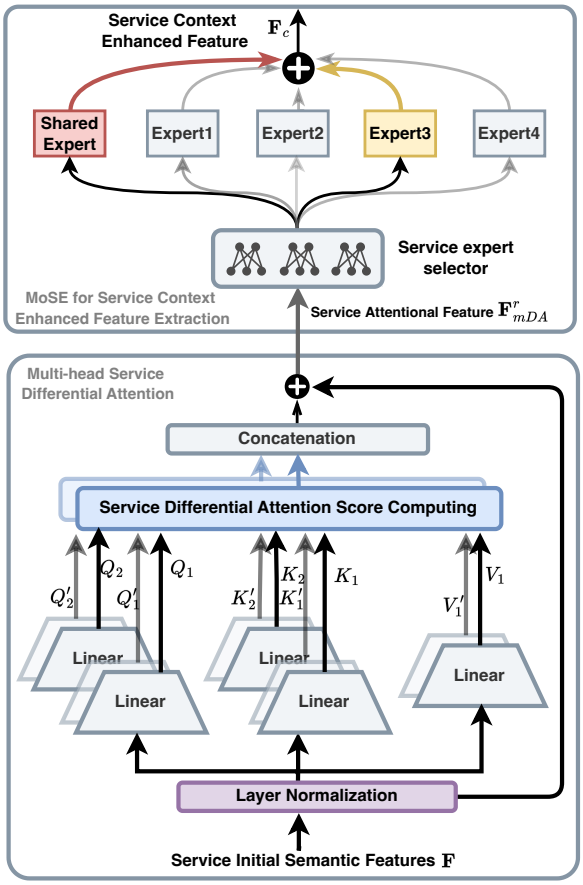


Fig. 3. Illustration of MoSE, including multi-head service differential attention and MoSE-based service context enhancement feature extraction.

of each existing APIs s_j in API repository $\mathbb{S}$.

2) Service Context Enhancement Feature Extraction: To leverage the category and historical invocation relationships between mashups and APIs for recommendation, and to further enhance the recommendation context for the mashup requirement, we predict its categories and potentially invoked relevant APIs. Furthermore, we observe that commonalities and similarities exist among certain different context prediction tasks. For instance, both the mashup and API category prediction tasks are multi-class classification problems within the same category space. Similarly, predicting relevant APIs for a target mashup and the task of identifying existing mashups with shared API invocations can both be framed as regression tasks following a similar paradigm. Consequently, we can extract common features from these related context prediction tasks. To this end, we propose MoSE to extract context enhancement features between similar service context prediction tasks, the architecture of MoSE is illustrated in Fig. 3.

Specifically, service initial semantic features $\mathbf{F}$ are input into a multi-head service differential attention-based residual network. And the computation of service query features and service key features in service differential attention is performed through two parallel sets of linear transformations. Specifically, the first set of linear layers $\mathbf{W}_Q$ and $\mathbf{W}_K$ projects $\mathbf{F}_A^{m_t,m_i}$ into the primary query-key feature space, while the second set $\mathbf{W}_{Q'}$ and $\mathbf{W}_{K'}$ projects it into a complementary feature space. This dual-space projection enables more comprehensive feature extraction and bias calibration in the subsequent attention computation process, as shown in Equation (7).

$$\begin{aligned}[f_Q \in \mathbb{Q}, f_{Q'} \in \mathbb{Q}'] &= [\mathbf{W}_Q \cdot \mathbf{F}_A, \mathbf{W}_{Q'} \cdot \mathbf{F}] \\ [f_K \in \mathbb{K}, f_{K'} \in \mathbb{K}'] &= [\mathbf{W}_K \cdot \mathbf{F}_A, \mathbf{W}_{K'} \cdot \mathbf{F}]\end{aligned} \tag{7}$$

where $[f_Q, f_{Q'}], [f_K, f_{K'}]$ are the service query feature pairs and key feature pairs obtained from the service initial semantic representation $\mathbf{F}$, respectively. For the service value feature set $\mathbb{V}$, we map it using a linear layer $\mathbf{W}_V$, as shown in Equation (8).

$$f_V \in \mathbb{V} = \mathbf{W}_V \cdot \mathbf{F} \tag{8}$$

Next, we use the service query feature pairs $[f_Q, f_{Q'}]$, service key feature pairs $[f_K, f_{K'}]$, service value feature f_V to compute service differential attention score and attain the single-head service attentional feature f_{DA}, as shown in Equation (9).

$$\begin{aligned}f_{DA} &= \mathbf{a} \cdot f_V \\ &= (\text{Softmax}(\frac{f_Q \cdot f_K}{\sqrt{k}}) - \lambda \cdot \text{Softmax}(\frac{f_Q \cdot f_K}{\sqrt{k}})) \cdot f_V\end{aligned} \tag{9}$$

where a is the service differential attention score, k is the embedding dimension of f_{DA} and λ is a learning parameter. Finally, we concatenate f_{DA} obtained from each attention head with residual connection to derive the multi-head service attentional feature $\mathbf{F}^r_{mDA}$, as shown in Equation (10).

$$\mathbf{F}^r_{mDA} = \text{Concat}(\mathbf{f_{DA_1}}, \mathbf{f_{DA_2}}, \ldots, \mathbf{f_{DA_h}}) + \mathbf{F} \tag{10}$$

The service differential attention residual network adaptively adjusts attention allocation based on the distinctions between different service context prediction tasks. Subsequently, we design a MoSE network that learns generic context enhancement features across similar service context prediction tasks through multiple parallel linear layers and a gating network. The architecture of MoSE network comprises three primary components: service expert selector, service experts pool, and expert feature aggregation. In particular, $\mathbf{F}^r_{mDA}$ is initially fed into the service expert selector to derive the service expert relevance distribution $\mathcal{P}_{SE}$, as shown in Equation (11).

$$\mathcal{P}_{SE} = \text{Softmax}(\mathbf{W}_{SES} \cdot \mathbf{F}^r_{mDA} + b_{SES}) \tag{11}$$

Furthermore, we input $\mathbf{F}^r_{mDA}$ into individual service expert networks, each comprising a single-layer linear network with dropout. Additionally, to learn enhancement features across similar service context prediction tasks, we combine and train similar tasks simultaneously. Moreover, we introduce a task-shared service expert alongside task-specific experts, resulting in $n_s = n_{\text{task}} + 1$, where n_s and n_{task} denote the number of service experts and service tasks, respectively. Consequently, we obtain a service expert feature set $\mathbb{F}_{SE} = \{\mathbf{F}_{SE_1}, \mathbf{F}_{SE_2}, \ldots, \mathbf{F}_{SE_{n_s}}\}$ of length n_s. Finally, we aggregate the service expert feature set based on $\mathcal{P}_{SE}$ to obtain the service generic feature $\mathbf{F}_c$, as shown in Equation (12).

$$\mathbf{F}_c = \sum_{i=1}^{n_s} \mathcal{P}_{SE}[i] \cdot \mathbf{F}_{SE_i} \tag{12}$$

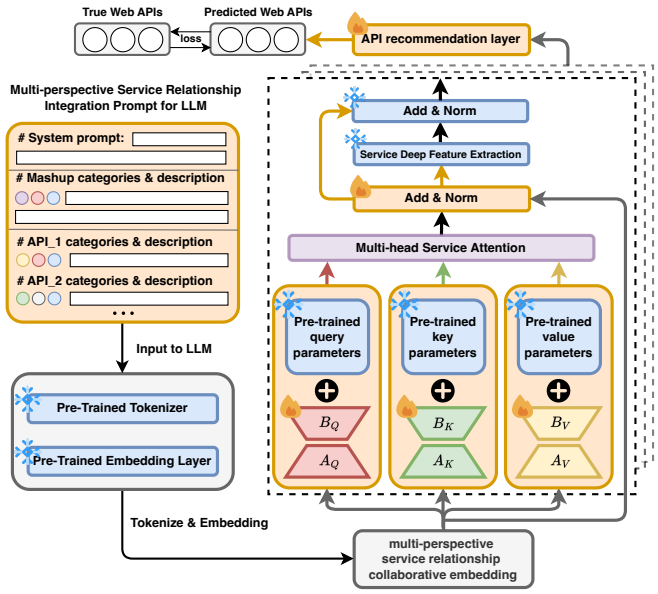


Fig. 4. The process of collaborative service feature extraction by fine-tuning LLM for Web API recommendation.

3) Mashup Associated Categories Prediction and Relevant APIs Identification: The service context enhancement feature $\mathbf{F}_c$ account for both the distinctions and interconnections between similar service context prediction tasks. Furthermore, to enable the utilization of $\mathbf{F}_c$ for both the mashup category prediction and relevant API identification tasks, we feed $\mathbf{F}_c$ into distinct task-specific layers. Specifically, for the mashup category prediction task, since it shares the same category space as the similar API category prediction task, the task-specific layer is a fully connected layer mapping from $\mathbb{R}^n$ to $\mathbb{R}^{|\mathbb{C}|}$, where n represents the feature dimension of $\mathbf{F}_c$, and $|\mathbb{C}|$ denotes the total number of categories, as shown in Equation (13).

$$p = \text{Softmax}(\mathbf{W}^{n\times|\mathbb{C}|} \cdot \mathbf{F}_c + \mathbf{b}) \tag{13}$$

where $\mathbf{W}^{n\times|\mathbb{C}|}, \mathbf{b}$ are the weight and bias of categories prediction layer.

However, for the relevant API identification task, its output space differs from that of the common invocation mashup identification task. To address this, we first conduct training for the similar mashup identification task and preserve the MoSE network parameters. Subsequently, we replace the task-specific layer and proceed with training for the relevant API identification task, the task-specific layer for relevant API and mashup identification is as follows:

$$p_a = \text{Softmax}({\mathbf{W_t}}^{n\times d_t} \cdot \mathbf{F}_c + \mathbf{b}_t), d_t \in \{|\mathbb{S}|, |\mathbb{M}|\} \tag{14}$$

where ${\mathbf{W_t}}^{n\times d_t}, \mathbf{b}_t$ are the task-specific weights and bias, d_t is task-dependent, taking values equal to either the total number of APIs or the count of existing mashups.

B. Multi-perspective Collaborative Service Feature Extraction and Recommendation by Fine-tuning LLM

Existing approaches that independently extract and subsequently fuse different service relationship features tend to result in the loss of cross-perspective collaborative features. To address this limitation, we integrate multiple service enhanced context, including refined requirement descriptions, categories and relevant APIs of mashup requirement, through a specially designed prompt as input of the LLM, thereby facilitating comprehensive collaborative feature extraction across multi-perspective service relationship between mashup and API, as shown in Fig. 4.

1) Multi-perspective Mashup Requirement Context Collaboration: Taking Web API recommendation for mashup m_i as an instance, we incorporate the refined mashup description D'_{m_i}, the predicted service categories C_{m_i} and the descriptions of relevant Web APIs $\mathbb{S}^i_R$ that have high potential invocation relevance with m_i, the process is as follows:

$$\begin{aligned} D^i = \text{Splice}(D^{i'}_m, \mathbb{S}^i_R) = [&\{D^{i'}_m, C_{m_i}\}, \{D^{1'}_s, C_{s_1}\}, \\ &\{D^{2'}_s, C_{s_2}\}, \ldots, \{D^{N'}_s, C_{s_N}\}] \end{aligned} \tag{15}$$

Next, we concatenate a concise system prompt with D_i to generate the final LLM input prompt P_i. The system prompt designates the LLM's role as a Web API recommendation system, specifies the mashup input format combining categories and descriptions, and acknowledges the presence of relevant Web APIs in the input. Furthermore, it explicitly instructs the LLM that recommendations need not be strictly confined to the relevant Web APIs and consider the multi-perspective service relationship collaboration between mashup and API. Then, we tokenize P^i using the pre-trained tokenizer and map the resulting token set into embedding space using the pre-trained embedding model $\mathcal{M}_e$. The process is as follows:

$$\mathbb{E}^i = \mathcal{M}_e(\text{tokenizer}(D^i)) = [\mathbf{e}_1, \mathbf{e}_2, ..., \mathbf{e}_q] \tag{16}$$

where $\mathbf{e}_p \in \mathbb{E}^i$ is the embedding of p-th token. Consequently, multi-perspective service context is effectively input into the LLM for further collaborative service feature extraction.

2) Collaborative Service Feature Learning by Fine-tuning LLM: Although the LLM is provided with the necessary data conditions to extract collaborative features by inputting the complete mashup requirement recommendation context, it cannot be directly utilized for collaborative service feature extraction due to the differences in pre-training objectives. To address this limitation, we employ the Q-LoRA technique, representing the parameter changes introduced by collaborative service features extraction through the multiplication of two low-rank matrices. The modified parameters are then added to the pre-trained parameters, the process is as follows:

$$\mathbf{W}_{\text{ft}} = \mathbf{W}_{\text{pt}} + \Delta\mathbf{W}_{\text{LoRA}} = \mathbf{W}_{\text{pt}} + \mathbf{B}_{\text{LoRA}} \cdot \mathbf{A}_{\text{LoRA}} \tag{17}$$

where $\mathbf{W}_{\text{ft}} \in \mathbb{R}^{d\times d}$ is the parameters of LLM which contains the ability of collaborative service features extraction, $\mathbf{W}_{\text{pt}} \in \mathbb{R}^{d\times d}$ is the pre-trained parameters, $\mathbf{W}_{\text{LoRA}} \in \mathbb{R}^{d\times d}$ is the newly added parameters, $\mathbf{B}_{\text{LoRA}} \in \mathbb{R}^{d\times l}$ and $\mathbf{A}_{\text{LoRA}} \in \mathbb{R}^{l\times d}$ are the low-rank parameters matrix, where l significantly smaller than d.

In each hidden layer of a pre-trained LLM, the decoder component includes four trainable parameters, denoted as $\mathbf{W}_q$, $\mathbf{W}_k$, $\mathbf{W}_v$, and $\mathbf{W}_o$. Consequently, for a given input $\mathbb{E}^i$, the

output of each LLM hidden layer becomes:

$$H_i = \mathbf{W}_o \cdot ((\text{softmax}(\frac{1}{\sqrt{d}}\mathbf{W}'_q \cdot \mathbf{W}'_k) \cdot \mathbf{W}'_v) \cdot \mathbb{E}^i) + \mathbf{b_o} \quad (18)$$

where $\mathbf{W}'_q, \mathbf{W}'_k, \mathbf{W}'_v$ is the parameters of LLM after fine-tuning, as shown in Equation (19)

$$\begin{aligned} \mathbf{W}'_q &= \mathbf{W}_q + \Delta\mathbf{W}_q = \mathbf{W}_q + \mathbf{B}_q \cdot \mathbf{A}_q \\ \mathbf{W}'_k &= \mathbf{W}_k + \Delta\mathbf{W}_k = \mathbf{W}_k + \mathbf{B}_k \cdot \mathbf{A}_k \\ \mathbf{W}'_v &= \mathbf{W}_v + \Delta\mathbf{W}_v = \mathbf{W}_v + \mathbf{B}_v \cdot \mathbf{A}_v \end{aligned} \quad (19)$$

where $\mathbf{W}_q, \mathbf{W}_k, \mathbf{W}_v$ are the pre-trained parameters, $\Delta\mathbf{W}_q$, $\Delta\mathbf{W}_k$ and $\Delta\mathbf{W}_v$ are the fine-tuning parameters by incorporating deep collaborative features extraction.

3) Web API Recommendation and Model Training: To enable the LLM to make recommendations across all available Web APIs rather than being confined to candidate APIs, LMSR replaces the original generative output layer of the LLM with a trainable multi-class classification layer whose output dimension equals the size of the Web API repository. Specifically, we task the last token feature of the final hidden layer state $H^{-1}_{(-1)}$ of the fine-tuned LLM as collaborative service feature into the Web API recommendation layer, as shown below:

$$\begin{aligned} \hat{\mathbf{y}}_{m_i} = \text{sigmoid}(&\mathcal{A}(\mathbf{W}_n(\cdots \\ &\mathcal{A}(\mathbf{W}_1 H^{-1}_{(-1)} + \mathbf{b}_1)\cdots) + \mathbf{b}_n)) \end{aligned} \quad (20)$$

where $\mathbf{W}_n, \mathbf{b}_n$ is the weight and bias parameter of the n-th hidden state of the classification layer, respectively, and $\mathcal{A}$ represents the activation function, $\hat{y}_{m_i}$ is the prediction probability that each Web API belongs to the mashup m_i. Then, we utilize the Binary Cross-Entropy Loss (BCELoss) function $\mathcal{L}_{\text{BCE}}(\cdot)$ to calculate loss, as shown in Equation (21):

$$\begin{aligned} \mathcal{L}_{\text{BCE}}(\hat{\mathbf{y}}_{m_i}, \mathbf{y}_{m_i}) = -\frac{1}{|\mathbb{S}|}\sum_{s\in\mathbb{S}} &(\mathbf{y}_{m_i}[s]\log(\hat{\mathbf{y}}_{m_i}[s]) \\ &+ (1-\mathbf{y}_{m_i}[s])\log(1-\hat{\mathbf{y}}_{m_i}[s])) \end{aligned} \quad (21)$$

where $\mathbf{y}_{m_i}[s]$ indicates whether Web API s truly contained by mashup m_i ($\mathbf{y}_{m_i}[s] = 1$ if m_i contains s; otherwise, $\mathbf{y}_{m_i}[s] = 0$), and $\hat{\mathbf{y}}_{m_i}[s]$ is the prediction of model. Finally, we use the predicted probability $\hat{\mathbf{y}}_{m_i} \subseteq \{0,1\}^{|\mathbb{S}|}$ to rank all Web APIs, thereby generating a recommended Web APIs list for mashup m_i.

Besides, we apply a regularization term into BCE loss function to prevent model overfitting, the BCE loss function in this paper is further defined as:

$$\mathcal{L}_{\text{BCE}}(\theta) = \min\left(\frac{1}{|\mathbb{M}|}\sum_{m_i\in\mathbb{M}}\mathcal{L}_{\text{BCE}}(\hat{\mathbf{y}}_{m_i}, \mathbf{y}_{m_i}) + \gamma\|\theta\|_2^2\right) \quad (22)$$

where $\theta = \{\mathbf{W}'_q, \mathbf{W}'_k, \mathbf{W}'_v, \mathbf{W}_1, \mathbf{b}_1, \ldots, \mathbf{W}_n, \mathbf{b}_n\}$ is the set of all trainable parameters, and $\gamma||\theta||_2^2$ is the L2 regularization term.

V. Experiments

A. Experimental Setup and Dataset

Experiments in this paper were conducted on a workstation equipped with dual NVIDIA RTX 4090 GPUs, dual Intel (R) Xeon (R) Silver 4210R @2.40 GHz CPUs, and 1TB RAM. The experimental environment consists of Python 3.12.5, CUDA 12.4, torch 2.5.0, and transformers 4.44.1.

TABLE I
Statistics of the experimental dataset

Statistics	Value
Mashups	7739
Web APIs	1342
Categories	443
Web APIs per mashup	1.944
Categories per mashup	2.997
Categories per Web API	2.887
Words in description per mashup	31.092
Words in description per Web API	68.972

To evaluate the performance of LMSR, we use a widely-used real-world dataset crawled from ProgrammableWeb, which contains 8,420 mashup services and 21,614 Web APIs. Without loss of generality, only the Web APIs that had been invoked at least once could be recommended in the experiment. After excluding mashup and Web APIs without description, final experimental dataset contains 7,739 mashups and 1,342 Web APIs. Detailed statistics of the dataset is presented in Table I. Besides, the experimental dataset was randomly dividing into training dataset contained 6,193 mashups, validation dataset contained 773 mashups and test dataset contained 773 mashups in the ratio of 8:1:1.

In LMSR, we utilize open-source LLMs from the LLaMA[1] family as base models for service initial semantic representation and fine-tuning optimization, specifically incorporating three LLMs: LLaMA3-8B, LLaMA3-3B, and LLaMA2-7B. To reduce memory usage during fine-tuning, we apply 4-bit quantization with the quantization type set to "nf4" and computation type set to "bfloat16". Regarding LoRA parameter settings, the rank of the parameter change matrix is set to 16, the LoRA scaling factor is set to 32, and the LoRA dropout is set to 0.1, the target parameter modules for LoRA fine-tuning are the $\mathbf{W}_q, \mathbf{W}_k, \mathbf{W}_v, \mathbf{W}_o$ within the attention module of the baseline LLM. The batch size was set to 1 for reducing memory usage, and we used AdamW optimizer with initial learning rate 2e-5 and weight decay of 0.01. Without loss of generality, similar to most approaches in the domain of Web API recommendation, the objective of the Web API recommendation task is to recommend the Top-N relevant Web APIs for a mashup service, where N is set to 5, 10, 15, and 20 in LMSR and all comparative approaches.

B. Evaluation Metrics

In this paper, we apply four widely-used evaluation metrics to evaluate the performance of approaches: Precision@N, Recall@N, Mean Average Precision (MAP@N), and Normalized Discounted Cumulative Gain (NDCG@N).

[1]https://www.llama.com

Precision@N is used to measure the proportion of correctly recommended Web APIs within the top-N recommended Web APIs list.

$$\text{Precision@N} = \frac{|realAPIs \cap topNAPIs|}{|topNAPIs|} \tag{23}$$

where $|realAPIs|$ represents the number of correctly recommended Web APIs, and $|topNAPIs|$ is the number of Web APIs in the top-N recommended list.

Recall@N is the proportion of the correctly recommended Web APIs to all actual used Web APIs in mashup service, as shown in below:

$$\text{Recall@N} = \frac{|realAPIs \cap topNAPIs|}{|realAPIs|} \tag{24}$$

NDCG@N takes into account the order of Web APIs in the recommendation list; the higher the position of a correctly recommended Web API in the list, the higher the value of this metric, as shown follows:

$$\text{NDCG@N} = \frac{\text{DCG@N}}{\text{IDCG@N}} \tag{25}$$

where DCG@N measures the position of correctly recommended Web APIs within the list, as shown below:

$$\text{DCG@N} = \sum_{i=1}^{N} \frac{real_i}{\log_2(i+1)} \tag{26}$$

IDCG@N is the theoretical maximum value of DCG@N, used to normalize DCG@N.

MAP@N is used to measure the number of correctly recommended Web APIs in the top-N recommended service list with the positional information, the higher position the correctly recommended Web APIs are in the list, the higher the value of this metric:

$$\text{MAP@N} = \frac{1}{|\mathbb{M}|} \sum_{m \in \mathbb{M}} AP_m@N \tag{27}$$

where $AP_m@N$ is the average precision of all correctly recommended positions in the TOP-N Web API list for mashup service m:

$$\text{AP@N} = \frac{\sum_{i=1}^{N} real_i * \text{Precision@i}}{\sum_{i=1}^{N} real_i} \tag{28}$$

C. Competing Approaches

To evaluate the effectiveness of LMSR, we compare it with timely representative Web API recommendation approaches, including three CF and content-based approaches [22], [23], [24], three deep learning-based approaches [25], [26], [10], three LLM-based approach [16], GPT-4o[2], LLaMA3-8B and two self-developed variants of LMSR, as described below.

- **NCF** [22]: It integrates neural networks into matrix factorization to obtain the service feature matrix.
- **SPR** [23]: It applies a user topic model to analyze service descriptions and recommends Web APIs by calculating the relevance between mashups and Web APIs.
- **RWR** [24]: It models the mashup context as a graph, then employs random walk and restart to extract mashup and Web API features.
- **MTFM** [25]: It utilizes the combination of invocation relationship and semantic descriptions for recommending Web APIs. To leverage the categorical relationships between APIs and mashups for recommendation, it performs category prediction for new mashup requirements.
- **FSFM** [26]: It integrates service functionality and structural information by enhancing the service semantic features with a heterogeneous network.
- **SEHCN** [10]: It proposes a multi-semantic aggregator to capture semantic associations between features and introduces a semantic service embedding component to learn global and local semantic information.
- **LLaMA3-8B**: It extracts semantic features from service descriptions using the pre-trained LLaMA3-8B LLM and performs recommendations by combining these semantic features with a Web API recommendation layer.
- **GPT-4o**: It utilizes the LLM-generated APIs as recommendation results by inputting service descriptions to the pre-trained GPT-4o model.
- **LLMAR** [16]: It proposes a multi-task instruction fine-tuning framework to acquire domain-specific knowledge for precise Web API recommendation.
- **LMSR (LLaMA3-3B)**: It another of the variants of our proposed approach LMSR, which takes LLaMA3-3B as the base LLM.
- **LMSR (LLaMA3-8B)**: It one of the variants of our proposed approach LMSR, which takes LLaMA3-8B as the base LLM.
- **LMSR (LLaMA2-7B)**: It is our main approach, containing mashup categories and candidate APIs acquisition, and deep collaborative feature extraction by fine-tuning LLM for Web API recommendation. It utilizes LLaMA2-7B as the base LLM.

D. Experiment Results and Analyses

Table II presents the experimental results of LMSR compared to 11 approaches under different settings for the number of recommended Web APIs, and we separate the experimental results of traditional, deep learning-based, and LLM-based Web API recommendation approaches from LMSR using horizontal lines in the table. Specifically, the approach performs the best on one metric is highlighted with a gray background and bold text, while the second-best approach is indicated with a light gray background. Additionally, the "Gains" column shows the performance improvement of LMSR over the current SOTA Web API recommendation approach, LLMAR.

By observing Table II, we observe that traditional Web API recommendation approaches such as NCF, SPR, and RWR fail to achieve satisfactory recommendation performance. This indicates that approaches based on collaborative filtering or insufficient semantic feature extraction are inadequate for complex Web API recommendation tasks. Taking N=5 as an example, LMSR outperforms the best traditional approach RWR by 51.51%, 41.35%, 62.39%, and 52.47% in terms of

[2]https://openai.com/

TABLE II
WEB API RECOMMENDATION PERFORMANCE COMPARISON OF DIFFERENT APPROACHES WHEN N IS 5, 10, 15 AND 20.

Approaches	N=5				N=10			
	Precision	Recall	MAP	NDCG	Precision	Recall	MAP	NDCG
NCF	0.1350	0.3871	0.3783	0.4191	0.0860	0.4859	0.3859	0.4500
SPR	0.1648	0.5314	0.4357	0.5017	0.1043	0.6476	0.4459	0.5286
RWR	0.1802	0.5719	0.4677	0.5266	0.1078	0.6829	0.4862	0.5718
MTFM	0.2069	0.6429	0.5571	0.6099	0.1164	0.7014	0.5698	0.6310
FSFM	0.2209	0.6729	0.6199	0.6630	0.1225	0.7203	0.6332	0.6794
SEHCN	0.2303	0.7464	0.6345	0.6924	0.1299	0.7996	0.6424	0.7095
LLaMA3-8B	0.0677	0.2454	0.2641	0.2441	0.0339	0.2457	0.2632	0.2443
GPT-4o	0.2539	0.7650	0.7179	0.7636	0.1398	0.8049	0.7303	0.7560
LLMAR	0.2436	0.7370	0.6708	0.7233	0.1365	0.7888	0.6853	0.7404
LMSR (LLaMA3-3B)	0.2687	0.8006	0.7488	0.7955	0.1479	0.8412	0.7628	0.8082
LMSR (LLaMA3-8B)	0.2700	0.8033	0.7570	0.8014	0.1480	0.8423	0.7707	0.8135
LMSR (LLaMA2-7B)	**0.2730**	**0.8084**	**0.7595**	**0.8029**	**0.1493**	**0.8468**	**0.7727**	**0.8147**
Gains on LLMAR	12.08%	9.69%	13.22%	11.01%	9.39%	7.35%	12.75%	10.03%
Approaches	**N=15**				**N=20**			
	Precision	**Recall**	**MAP**	**NDCG**	**Precision**	**Recall**	**MAP**	**NDCG**
NCF	0.0624	0.5346	0.3867	0.4606	0.0496	0.5638	0.3824	0.4642
SPR	0.0825	0.7393	0.4417	0.5519	0.0672	0.8034	0.4463	0.5475
RWR	0.0810	0.7466	0.4705	0.5870	0.0628	0.8092	0.4727	0.5822
MTFM	0.0831	0.7380	0.5747	0.6426	0.0648	0.7555	0.5767	0.6480
FSFM	0.0857	0.7481	0.6360	0.6881	0.0665	0.7677	0.6378	0.6938
SEHCN	0.0909	0.8203	0.6467	0.7157	0.0705	0.8313	0.6476	0.7189
LLaMA3-8B	0.0229	0.2471	0.2636	0.2443	0.0174	0.2494	0.2642	0.2445
GPT-4o	0.0970	0.8228	0.7338	0.7819	0.0745	0.8363	0.7353	0.7859
LLMAR	0.0967	0.8218	0.6906	0.7510	0.0749	0.8371	0.6926	0.7557
LMSR (LLaMA3-3B)	0.1024	0.8591	0.7663	0.8140	0.0788	**0.8740**	0.7682	0.8184
LMSR (LLaMA3-8B)	0.1024	0.8591	0.7741	0.8191	0.0783	0.8692	0.7753	0.8221
LMSR (LLaMA2-7B)	**0.1032**	**0.8611**	**0.7759**	**0.8197**	**0.0790**	0.8710	**0.7773**	**0.8228**
Gains on LLMAR	6.73%	4.79%	12.35%	9.15%	5.50%	4.04%	12.23%	8.88%

Precision, Recall, MAP, and NDCG, respectively. Therefore, it is essential to consider multi-perspective service relationship and employ deep feature extraction models to obtain high-quality deep service features, which is crucial for achieving superior Web API recommendation results.

Moreover, Observing deep learning-based Web API recommendation approaches, we find that their performance depends on both the quality of deep service features extracted by the employed deep neural networks and the number of service relationships considered between mashups and APIs. Consequently, SEHCN achieves the best performance among deep learning-based approaches by utilizing lightGCN networks to combine service category relationships, semantic relationships, and structural relationships. However, SEHCN assumes that mashup requirements possess predefined categories and partially known API invocations, which is incompatible with real-world application scenarios where such prior knowledge is typically unavailable. Moreover, the performance of SEHCN is constrained by the scale and limited feature extraction capability of lightGCN. Therefore, LLM-based approaches LLMAR, leveraging the powerful semantic understanding and relationship feature extraction capabilities of LLMs, achieve the current SOTA performance in Web API recommendation.

However, although LLMAR can learn service category and invocation relationships through multi-task instruction fine-tuning, it fails to explicitly utilize these learned relationships in the mashup recommendation task due to the absence of mashup categories and potential API invocations in the prompt. This limitation prevents the LLM from effectively extracting collaborative features from the semantic, category, and invocation relationships between mashups and APIs, thereby constraining further improvements in Web API recommendation performance. In contrast, LMSR first employs the LLM-based MoSE network to refine requirement descriptions and accurately predict mashup categories and relevant API. Subsequently, it integrates the refined requirement description, categories, relevant API descriptions, and categories through prompts as integration input to the LLM. It enables the LLM to directly extract collaborative features from the semantic, category, and invocation relationships between mashups and APIs. These improvements enable LMSR to surpasses LLMAR, by 12.08%, 9.69%, 13.22%, and 11.01% in Precision, Recall, MAP, and NDCG, respectively when $N = 5$, this significant improvement in Web API recommendation performance demonstrates the effectiveness of the LMSR.

Furthermore, compared to the current representative general

TABLE III
PERFORMANCE COMPARISON OF ABLATION APPROACHES.

Approaches	Precision@5	Recall@5	MAP@5	NDCG@5	Precision@10	Recall@10	MAP@10	NDCG@10
LMSR *w/o* FT	0.2051	0.6006	0.5127	0.5672	0.1191	0.6770	0.5295	0.5923
LMSR *w/o* CF	0.2390	0.6729	0.6080	0.6510	0.1349	0.7339	0.6244	0.6700
LMSR *w/o* MC	0.2312	0.6488	0.5834	0.6260	0.1313	0.7123	0.5999	0.6461
LMSR	**0.2730**	**0.8084**	**0.7595**	**0.8029**	**0.1493**	**0.8468**	**0.7727**	**0.8147**
Approaches	**Precision@15**	**Recall@15**	**MAP@15**	**NDCG@15**	**Precision@20**	**Recall@20**	**MAP@20**	**NDCG@20**
LMSR *w/o* FT	0.0844	0.7094	0.5341	0.6020	0.0663	0.7300	0.5367	0.6083
LMSR *w/o* CF	0.0943	0.7603	0.6283	0.6780	0.0727	0.7754	0.6301	0.6825
LMSR *w/o* MC	0.0918	0.7375	0.6037	0.6538	0.0711	0.7542	0.6056	0.6586
LMSR	**0.1032**	**0.8611**	**0.7759**	**0.8197**	**0.0790**	**0.8710**	**0.7773**	**0.8228**

TABLE IV
PERFORMANCE COMPARISON ON DIFFERENT SERVICE CONTEXT PREDICTION TASKS FOR MASHUP REQUIREMENTS.

Approaches	associated categories prediction				relevant APIs identification			
	Recall@5	NDCG@5	Recall@10	NDCG@10	Recall@5	NDCG@5	Recall@10	NDCG@10
MTFM	0.6267	**0.7757**	0.7064	0.7691	-	-	-	-
LMSR-Linear	0.7109	0.7197	0.7946	0.7536	0.6488	0.6260	0.7123	0.6461
LMSR-MoSE	**0.7340**	0.7362	**0.8240**	**0.7763**	**0.6729**	**0.6510**	**0.7339**	**0.6700**

pre-trained LLM, GPT-4o, LMSR significantly outperforms it across all evaluation metrics, even with substantially fewer model parameters. This superiority primarily stems from LMSR's ability to fully leverage collaborative features from multiple service relationships between mashups and APIs, while reducing hallucinations through fine-tuning and output layer replacement, thereby making the LLM more suitable for Web API recommendation tasks. Additionally, compared to the baseline LLM LLaMA3-8B used in LMSR, LMSR achieves substantial performance improvements after fine-tuning. Taking N=5 as an example, LMSR outperforms the pre-trained LLaMA3-8B by 303.2%, 229.4%, 187.6%, and 228.9% in terms of Precision, Recall, MAP, and NDCG, respectively, further demonstrating the necessity and effectiveness of the LMSR approach.

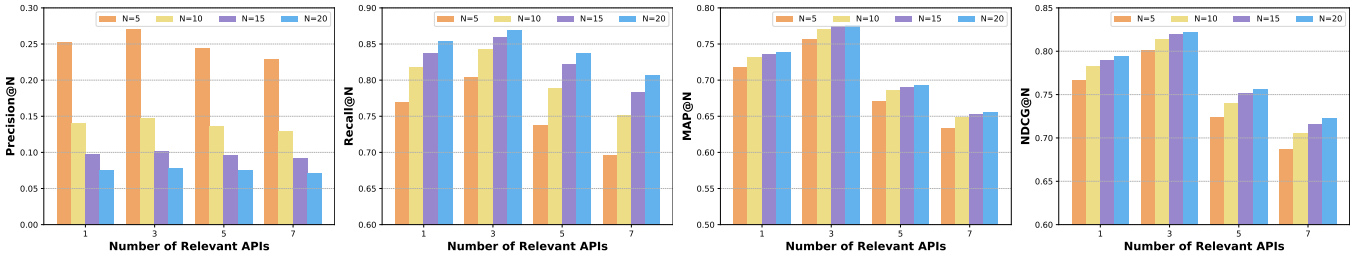
To investigate the impact of different parameter sizes of LLMs on LMSR's performance, we conducted experiments using LLMs with varying parameter sizes: LLaMA3-3B, LLaMA2-7B, and LLaMA3-8B as base models for service initial embedding representation and collaborative service feature extraction. As observed in Table II, with the scale of model parameters increases, the quality of collaborative service features improves accordingly, the model achieves optimal performance when the parameter size reaches 7B. However, further increasing the parameter size not only fails to improve the performance of Web API recommendation, but also leads to a significant increase in the computational cost. Based on these findings, LMSR adopts LLaMA2-7B as the primary base LLM for Web API recommendation.

E. Ablation Study

1) Analysis contributions of component in LMSR: To investigate the impact of each module within the LMSR framework on Web API recommendation performance, we proposed three self-developed variants based on the LMSR framework: LMSR *w/o* FT, LMSR *w/o* CF and LMSR *w/o* MC. Among these, LMSR *w/o* FT is a variant of LMSR that does not involve fine-tuning of LLM, LMSR *w/o* CF is a variant of LMSR that directly utilizes the relevant APIs output by the MoSE network as recommendation results without inputting to the LLM, and LMSR *w/o* MC is a variant of LMSR that does not incorporate mashup categories and relevant APIs predicted by the MoSE network, instead directly combining original service descriptions and raw invocation records to fine-tune the LLM.

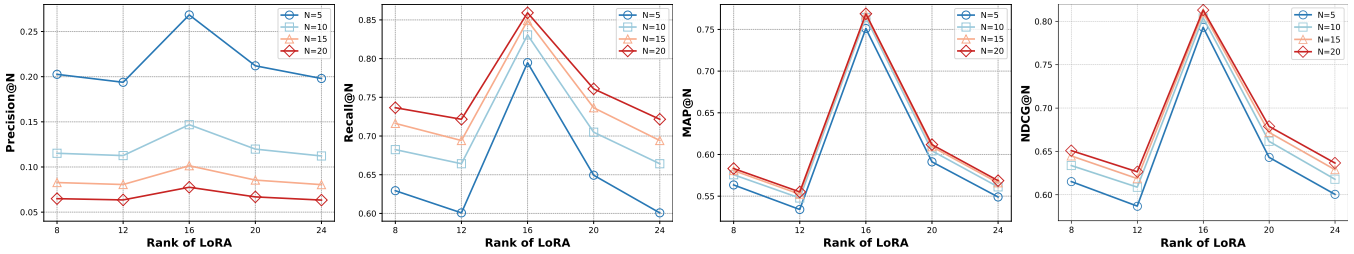
Tables III presents a comparison of the experimental results between LMSR and its three variant approaches. Examining the experimental results of LMSR *w/o* FT, we observe a significant performance degradation when feeding mashup requirement enhanced recommendation context into the non-fine-tuned LLM. This deterioration can be attributed to the LLM's lack of ability for collaborative service feature extraction from multi-perspective service relationship. In this scenario, LLM merely performs semantic matching between relevant APIs and the mashup requirement, with recommendation accuracy being constrained by the precision of relevant APIs. This observation is further corroborated by the fact that LMSR *w/o* FT exhibits inferior recommendation performance compared to LMSR *w/o* CF. The experimental results of LMSR *w/o* FT demonstrate that fine-tuning LLM for collaborative service feature extraction from multi-perspective service relationship is crucial for enhancing the LLM's feature extraction capabilities and consequently improving Web API recommendation performance.

Furthermore, examining the experimental results of LMSR *w/o* CF, we observe that while the LLM-based MoSE network successfully learns context enhancement features across similar service context prediction task, the absence of fine-tuned LLM for further extraction of collaborative service features from multi-perspective service relationship prevents the model from achieving higher recommendation accuracy. Moreover, the experimental results of LMSR *w/o* MC demonstrate that while fine-tuning LLM is crucial for extracting collaborative



(a) Precision@N (b) Recall@N (c) MAP@N (d) NDCG@N

Fig. 5. Performance impact of the different number of relevant Web APIs contained in the input of the LLM.



(a) Precision@N (b) Recall@N (c) MAP@N (d) NDCG@N

Fig. 6. Performance impact of the different rank r of the LoRA matrix

service features and improving Web API recommendation performance, without the preliminary enhancement of recommendation context by MoSE, merely mining invocation relationships from raw service invocation records prevents the LLM from capturing categorical relationships between mashups and APIs. Furthermore, since mashup requirements lack invocation relationships, this limitation prevents the model from utilizing invocation relationships during the recommendation phase, thereby degrading recommendation performance. Additionally, the refined service descriptions enable the LLM to extract high-quality semantic features, further enhancing recommendation accuracy. These experimental results comprehensively validate both the accuracy and necessity of LLM-based MoSE.

2) Analysis of the contribution of MoSE in Mashup Categories and Relevant APIs Prediction: Table IV presents a performance comparison between LMSR, its variants, and the baseline approach MTFM [25] across mashup category prediction and relevant API identification. LMSR-Linear represents a variant that bypasses the MoSE network and directly extracts task-specific features through multiple multilayer perceptrons. LMSR-MoSE represents our primary approach, which employs the MoSE network to extract context enhancement features across multiple similar service context prediction tasks.

The experimental results demonstrate that LMSR achieves substantial improvements over the category-aware Web API recommendation approach MTFM in mashup category prediction, primarily due to its ability to extract high-quality semantic features based on LLM from refined service descriptions. Furthermore, the experimental results between LMSR-Linear and LMSR-MoSE reveals that, in contrast to linear layers, MoSE's shared service experts across multiple context prediction tasks enable the extraction of generic context enhancement features among similar tasks, thereby further enhancing the accuracy of service context prediction. These findings validate both the necessity and effectiveness of the MoSE network architecture.

F. Performance Impact of Parameters

1) Impact of Number of Relevant Web APIs: The number of relevant Web APIs $|\mathbb{S}_R|$ included in the input of the LLM affects the performance of Web API recommendation. Fig. 5 (a)-(d) illustrate the variations in different evaluation metrics with different values of $|\mathbb{S}_R|$. In each subplot, a bar represents the performance of the corresponding evaluation metric for different Top-N values.

Observing Fig. 5 (a)-(d), it can be noted that as the number of relevant Web APIs increases, all four performance metrics for evaluating Web API recommendation initially rise and then fall, reaching their peak at $|\mathbb{S}_R| = 3$ (3 is close to the average number of APIs invoked in the mashup dataset we used). Upon thoroughly analysis of the experimental results, we believe this phenomenon is caused by the limited number of Web APIs truly invoked by a mashup service. As the number of relevant Web APIs increases, while the LLM acquires richer contextual information, it may also be subject to more noise. The experiments show that when $|\mathbb{S}_R| \leq 3$, the model gains more relevant contextual information than noise, positively impacting performance. Conversely, beyond this point, the LLM receives more noise, leading to a decline in Web API recommendation performance. Therefore, in the experiments, proper number of relevant Web APIs are input to the LLM which not only ensures an improvement in the performance of Web API recommendation, but also avoids the computational overhead associated with excessive contextual information during training and inference.

2) Impact of Rank of LoRA Matrix: LMSR employs the Q-LoRA technique for fine-tuning, which transforms the fine-tuning parameters into the product of two low-rank matrices. Consequently, the rank r of the LoRA matrices significantly impacts the performance of Web API recommendation. Fig. 6 illustrates the variations in different evaluation metrics for Web API recommendation performance across various r values. Specifically, each subplot in Fig. 6 represents an evaluation metric, and each line within a subplot depicts the performance changes for a specific Top-N recommendation, where $N \in \{5, 10, 15, 20\}$.

Analyzing subplots (a)-(d) in Fig. 6, it can be observed that for each recommendation performance evaluation metric, the performance initially increases and then decreases as the value of r rises. Specifically, taking Fig. 6 (b) as an instance, when $r \leq 16$, the value of the evaluation metric *Recall@N* exhibits a fluctuating upward trend. However, when $r > 16$, the value of *Recall@N* declines significantly. Similar conclusions can be drawn from the other subplots in Fig. 6. We attribute this experimental phenomenon to the following reason: as the value of r increases, the fitting capacity of the fine-tuning parameters improves. Consequently, the performance of Web API recommendations initially rises with increasing r. However, after reaching a peak, the limited size of the fine-tuning dataset combined with the growing number of fine-tuning parameters leads to increasingly severe overfitting, resulting in a significant decline in Web API recommendation performance. To mitigate the risk of exacerbating overfitting, we chose to set r to 16 in the experiments, thereby maximizing the performance of the fine-tuned model in Web API recommendation.

VI. Related Work

A. CF and Content-based Web API Recommendation

CF-based approaches identify services similar to the target mashup and recommend relevant Web APIs based on the historical invocation records of similar services. Alinia et al. [11] considered location features of the services to find similar services, and proposed a Web API recommendation framework combining self-attention and collaborative filtering, which ultimately recommended Web APIs based on the predicted service quality. Kang et al. [15] introduced the concept of domain intersection to reduce computational complexity and employed an implicit attention mechanism to differentiate the importance of various intersecting features. The intersecting features are ultimately applied for Web API recommendations through deep neural decomposition. Yao et al. [27] proposed a probabilistic matrix factorization-based approach for Web API recommendation, incorporating implicit correlation regularization to enhance recommendation diversity. Meng et al. [28] proposed a time-aware scalable collaborative filtering model with trust-clustering for real-time edge services recommendation.

Content-based Web API recommendation approaches achieve Web API recommendations by designing various feature extraction models to uncover hidden semantic features within multidimensional service content information, such as service descriptions, service names, and service tags. Cao et al. [29] used a two-level topic model to generate implicit cluster features and then predicted service invocation records based on co-invocation relationship between Web APIs. Pan et al. [30] proposed a Web API recommendation approach with keyword semantic enhancement and label co-occurrence to extract important words and categories in service descriptions. Ren et al. [31] applied distribution-aware SVM for services recommendation by identifying similar users, refining dataset and decreasing unfaithful rating.

Nevertheless, constrained by the limited service relationships and the limitations of feature extraction models, CF and content-based approaches struggle to achieve satisfactory Web API recommendation performance.

B. Deep Learning-based Web API Recommendation

In recent years, with the rapid development of deep learning technologies, an increasing number of researchers have been utilizing deep neural networks to address Web API recommendation problems. Wang et al. [32] proposed a weight adaptive multi-task deep neural network for Web API recommendation by fully applying the correlation information between multiple subtasks. Boulakbech et al. [33] introduce an attentional deep learning model for service recommendation, they use two attention-based neural network to discover services from both functional and structural perspectives.

Additionally, many works are currently applying graph neural networks to explore the structural relationships between mashup services and Web APIs. Xiao et al. [34] capture structural relation and attribute information of Web APIs by a structural reinforcing and attribute weakening network. Yan et al. [35] utilize collaborative attention graph convolutional network to acquire structural features from service co-invocation graph. Wang et al. [36] propose a Motif-based Graph Attention Network for Web API recommendation by capturing high-order information of different motifs.

However, existing deep learning-based approaches demonstrate limitations in effectively utilizing categories and invocation relationship of new mashup requirements. And they also struggle to extract deep collaborative features from multi-perspective service relationships.

C. LLM-based Web API Recommendation

Given LLM's [37] [38] powerful semantic understanding and feature extraction capabilities, researchers have begun exploring the application of LLM to Web API recommendation. Zheng et al. [17] proposed a LLM-driven and motif-informed linearizing graph transformer for Web API recommendation. Qin et al. [16] proposed a LLM-based Web API recommendation approach by leveraging the LLM's semantic understanding and feature extraction capabilities to enhance the recommendation performance. Although LLM-based approaches have achieved excellent Web API recommendation performance, these approaches overlook the difficulty in obtaining categories and relevant APIs for mashup requirements in real-world scenarios, and ignore the collaborative features among multi-perspective service relationships, resulting in

both limited improvement in Web API recommendation performance and reduced approach practicality.

To address these limitations, we propose LMSR, a LLM-enhanced multi-perspective service feature learning for Web API recommendation. LMSR employs specially designed prompts with LLM for semantic refinement of service descriptions, while leveraging an LLM-based MoSE network to extract context refinement features across similar service context prediction tasks for precise mashup categories and relevant APIs extraction. Through fine-tuning the LLM, LMSR enhances the LLM's capability to extract collaborative features from multi-perspective service relationships, ultimately achieving superior Web API recommendation performance.

VII. Conclusion and Future Work

This paper presents LMSR, a novel LLM-enhanced multi-perspective service feature learning framework for Web API recommendation. To address the challenges of Web API recommendation, LMSR first employs a pre-trained LLM to refine and encode the original requirement descriptions, while leveraging the MoSE network to extract context enhancement features for accurate predictions of categories and relevant APIs, which ultimately enhances the recommendation context of mashup requirements. Furthermore, LMSR inputs the refined requirement descriptions, categories, and relevant APIs into the LLM. By fine-tuning the LLM, LMSR extracts collaborative features from multi-perspective service relationships between mashup requirements and APIs, ultimately achieving precise Web API recommendations. Extensive experiments conducted on large-scale real-world datasets demonstrate that LMSR significantly outperforms existing approaches.

For future work, we plan to explore more advanced prompt engineering techniques, leverage more advanced LLMs as the foundation model, and develop more efficient fine-tuning strategies to further improve the framework's performance and adaptability in Web API recommendation.

Acknowledgment

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

References

[1] Z. He, C. Chow, J. Zhang, and K. Lam, "H3Rec: Higher-order heterogeneous and homogeneous interaction modeling for group recommendations of web services," *IEEE Transactions on Services Computing*, vol. 16, no. 2, pp. 1212–1224, 2023.

[2] B. Cao, M. Peng, L. Zhang, Y. Qing, B. Tang, G. Kang, and J. Liu, "Web service recommendation via integrating heterogeneous graph attention network representation and fibinet score prediction," *IEEE Transactions on Services Computing*, vol. 16, no. 5, pp. 3837–3850, 2023.

[3] B. Cao, M. Peng, Z. Xie, J. Liu, H. Ye, B. Li, and K. K. Fletcher, "PRKG: pre-training representation and knowledge-graph-enhanced web service recommendation for mashup creation," *IEEE Transactions on Network and Service Management*, vol. 21, no. 2, pp. 1737–1749, 2024.

[4] G. Kang, Y. Wang, H. Ren, B. Cao, J. Liu, and Y. Wen, "KS-GNN: keyword search via graph neural network for Web API recommendation," *IEEE Transactions on Network and Service Management*, vol. 21, no. 5, pp. 5464–5474, 2024.

[5] C. B. Njima, C. G. Guégan, Y. Gamha, and L. B. Romdhane, "Web service composition in mobile environment: A survey of techniques," *IEEE Transactions on Services Computing*, vol. 17, no. 2, pp. 689–704, 2024.

[6] L. Kong, H. Ding, and G. Hu, "GCNSLIM: graph convolutional network with sparse linear methods for e-government service recommendation," *Knowledge-Based Systems*, vol. 292, p. 111593, 2024.

[7] Z. Liu, Q. Z. Sheng, D. Chu, X. Xu, H. Zheng, and K. Feng, "Proactive recommendation of composite services in multi-Access edge computing," *IEEE Transactions on Services Computing*, vol. 17, no. 2, pp. 631–644, 2024.

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

[9] Z. Zhang, Y. Zhang, M. Dong, K. Ota, Y. Zhang, and Y. Ren, "Collaborative tag-aware graph neural network for long-tail service recommendation," *IEEE Transactions on Services Computing*, vol. 17, no. 5, pp. 2124–2138, 2024.

[10] X. Wang, M. Xi, Y. Li, X. Pan, Y. Wu, S. Deng, and J. Yin, "SEHGN: semantic-enhanced heterogeneous graph network for Web API recommendation," *IEEE Transactions on Services Computing*, vol. 17, no. 5, pp. 2836–2849, 2024.

[11] M. Alinia and S. M. H. Hasheminejad, "DiSA-CF: A distance-integrated self-attention model for collaborative filtering in web service recommendation," *Expert Systems With Applications*, 2025, doi: https://doi.org/10.1016/j.eswa.2024.125223.

[12] X. Hu, "Biased random walk based Web API recommendation in heterogeneous network," in *IEEE International Conference on Web Services (ICWS)*, 2024, pp. 172–177.

[13] H. Kou, J. Xu, and L. Qi, "Diversity-driven automated Web API recommendation based on implicit requirements," *Applied Soft Computing*, 2023, doi: https://doi.org/10.1016/j.asoc.2023.110137.

[14] S. Wu, S. Shen, X. Xu, Y. Chen, X. Zhou, D. Liu, X. Xue, and L. Qi, "Popularity-aware and diverse Web APIs recommendation based on correlation graph," *IEEE Transactions on Computational Social Systems*, vol. 10, no. 2, pp. 771–782, 2023.

[15] G. Kang, L. Ding, J. Liu, B. Cao, and Y. Xu, "Web API recommendation based on self-attentional neural factorization machines with domain interactions," *IEEE Transactions on Network Science and Engineering*, vol. 10, no. 6, pp. 3953–3963, 2023.

[16] S. Qin, Y. Zhao, H. Wu, L. Zhang, and Q. He, "Harnessing the power of large language model for effective Web API recommendation," *IEEE Transactions on Industrial Informatics*, 2025, doi:10.1109/tii.2025.3552722.

[17] X. Zheng, G. Wang, G. Xu, J. Yang, B. Han, and J. Yu, "A LLM-driven and motif-informed linearizing graph transformer for Web API recommendation," *Applied Soft Computing*, 2025, doi:10.1016/J.ASOC.2024.112547.

[18] T. Ye, L. Dong, Y. Xia, Y. Sun, Y. Zhu, G. Huang, and F. Wei, "Differential transformer," *arXiv preprint arXiv:2410.05258*, 2024.

[19] T. Dettmers, A. Pagnoni, A. Holtzman, and L. Zettlemoyer, "Qlora: Efficient finetuning of quantized LLMs," *Advances in Neural Information Processing Systems (NeurIPS)*, 2024.

[20] J. Wei, X. Wang, D. Schuurmans, M. Bosma, F. Xia, E. Chi, Q. V. Le, D. Zhou *et al.*, "Chain-of-thought prompting elicits reasoning in large language models," *Advances in neural information processing systems*, vol. 35, pp. 24 824–24 837, 2022.

[21] H. Touvron, T. Lavril, G. Izacard, X. Martinet *et al.*, "Llama: Open and efficient foundation language models," *arXiv preprint arXiv:2302.13971*, 2023.

[22] X. He, L. Liao, H. Zhang, L. Nie, X. Hu, and T.-S. Chua, "Neural collaborative filtering," in *International Conference on World Wide Web(WWW)*, 2017, pp. 173–182.

[23] Y. Zhong, Y. Fan, W. Tan, and J. Zhang, "Web service recommendation with reconstructed profile from mashup descriptions," *IEEE Transactions on Automation Science and Engineering*, vol. 15, no. 2, pp. 468–478, 2018.

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

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

[26] X. Wang, M. Xi, and J. Yin, "Functional and structural fusion based Web API recommendations in heterogeneous networks," in *IEEE International Conference on Web Services (ICWS)*, 2023, pp. 91–96.

[27] L. Yao, X. Wang, Q. Z. Sheng, B. Benatallah, and C. Huang, "Mashup recommendation by regularizing matrix factorization with API co-

invocations," *IEEE Transactions on Services Computing*, vol. 14, no. 2, pp. 502–515, 2021.
[28] S. Meng, J. Xu, H. Wang, R. Yuan, J. Zhang, and Q. Li, "Time-aware Scalable Recommendation with Clustering-based Distributed Factorization for Edge Services," *World Wide Web*, vol. 25, no. 5, pp. 1831–1849, 2022.
[29] B. Cao, X. F. Liu, M. M. Rahman, B. Li, J. Liu, and M. Tang, "Integrated content and network-based service clustering and web apis recommendation for mashup development," *IEEE Transactions on Services Computing*, vol. 13, no. 1, pp. 99–113, 2020.
[30] G. Pan, Y. Chang, H. Qi, and Q. Hu, "Web service category recommendation with feature word semantic enhancement and tag co-occurrence," in *International Conference on Networking and Network Applications (NaNA)*, 2023, pp. 686–689.
[31] L. Ren and W. Wang, "A granular svm-based method for top-n web services recommendation," *IEEE Transactions on Services Computing*, vol. 15, no. 1, pp. 457–469, 2022.
[32] J. Wang, X. Zhang, Q. Wang, W. Zheng, and Y. Xiao, "Qos prediction method via multi-task learning for web service recommendation," in *IEEE International Conference on Web Services (ICWS)*, 2024, pp. 1353–1355.
[33] M. Boulakbech, N. Messai, Y. Sam, and T. Devogele, "Deep learning model for personalized web service recommendations using attention mechanism," in *International Conference on Service-Oriented Computing(ICSOC)*, vol. 14419, 2023, pp. 19–33.
[34] Y. Xiao, J. Liu, G. Kang, R. Hu, B. Cao, Y. Cao, and M. Shi, "Structure reinforcing and attribute weakening network based API recommendation approach for mashup creation," in *IEEE International Conference on Web Services (ICWS)*, 2020, pp. 541–548.
[35] R. Yan, Y. Fan, J. Zhang, J. Zhang, and H. Lin, "Service recommendation for composition creation based on collaborative attention convolutional network," in *IEEE International Conference on Web Services (ICWS)*, 2021, pp. 397–405.
[36] G. Wang, J. Yu, M. Nguyen, Y. Zhang, S. Yongchareon, and Y. Han, "Motif-based Graph Attentional Neural Network for Web Service Recommendation," *Knowledge-Based Systems*, vol. 269, p. 110512, 2023.
[37] L. Wu, Z. Zheng, Z. Qiu, H. Wang, H. Gu, T. Shen, C. Qin, C. Zhu, H. Zhu, Q. Liu *et al.*, "A survey on large language models for recommendation," *World Wide Web*, 2024, doi: https://doi.org/10.1007/s11280-024-01291-2.
[38] Y. Chang, X. Wang, J. Wang, Y. Wu, L. Yang, K. Zhu, H. Chen, X. Yi, C. Wang, Y. Wang *et al.*, "A survey on evaluation of large language models," *ACM Transactions on Intelligent Systems and Technology*, vol. 15, no. 3, pp. 1–45, 2024.

Song Yang is currently a PhD candidate in the School of Computer Engineering and Science, Shanghai University, China. Before that, he received a Bachelor degree in 2019 and Master degree in 2022 both in Computer Science and Technology at Shanghai University, respectively. His research interests include service computing, edge computing and service recommendation. He has published seven papers on IEEE Transactions on Network and Service Management, ACM Transactions on Autonomous and Adaptive Systems, AAAI, International Conference on Web Services (ICWS), International Conference on Collaborative Computing (EAI CollaborateCom), etc.

Guobing Zou is a full professor and vice dean of the School of Computer Science, Shanghai University, China. He received his PhD degree in Computer Science from Tongji University, Shanghai, China, 2012. He has worked as a visiting scholar in the Department of Computer Science and Engineering at Washington University in St. Louis from 2009 to 2011, USA. His current research interests mainly focus on services computing, edge computing, data mining and intelligent algorithms, recommender systems. He has published more than 120 papers on premier international journals and conferences, including IEEE Transactions on Services Computing, IEEE Transactions on Network and Service Management, IEEE ICWS, ICSOC, IEEE SCC, AAAI, Information Sciences, Expert Systems with Applications, Knowledge-Based Systems, etc.

Shengxiang Hu is currently a PhD candidate in the School of Computer Engineering and Science at Shanghai University, China. Prior to his ongoing PhD work, he received his Master's degree in Computer Science and Technology from Shanghai University in 2021. His primary areas of research encompass Quality of Service (QoS) prediction, graph neural networks, and natural language processing. He has authored and co-authored more than 20 scholarly papers in premier venues including IEEE Transactions on Service Computing, Knowledge-based Systems, AAAI, IEEE ICWS, ICSOC, and PPSN, etc.

Shengye Pang is an Assistant Professor at the School of Computer Engineering and Science, Shanghai University. He received his PhD in 2024 from the School of Computer Science and Technology at Zhejiang University. His primary research interests lie in service computing and service network. He has published more than 20 papers on premier international journals and conferences, including IEEE Transactions on Service Computing, WWW, ACM International Conference on Multimedia, IEEE International Conference on Web Services, ACM Transactions on Sensor Networks, ACM Transactions on Autonomous and Adaptive Systems.

Yanglan Gan received the PhD degree in computer science from Tongji University, Shanghai, China, 2012. She is a full professor in the School of Computer Science and Technology, Donghua University, Shanghai, China. Her research interests include bioinformatics, service computing, and data mining. She has published more than 50 papers on premier international journals and conferences, including Bioinformatics, Briefings in Bioinformatics, BMC Bioinformatics, IEEE/ACM Transactions on Computational Biology and Bioinformatics, IEEE Transactions on Services Computing, IEEE Transactions on Network and Service Management, IEEE ICWS, ICSOC, Neurocomputing, and Knowledge-Based Systems.

Bofeng Zhang is a full professor and dean of the School of Computer and Information Engineering, Shanghai Polytechnic University, Shanghai, China. He received his PhD degree from the Northwestern Polytechnic University (NPU) in 1997, China. He experienced a Postdoctoral Research at Zhejiang University from 1997 to 1999, China. He worked as a visiting professor at the University of Aizu from 2006 to 2007, Japan. He worked as a visiting scholar at Purdue University from 2013 to 2014, US. His research interests include personalized service recommendation, intelligent human-computer interaction, and data mining. He has published more than 200 papers on international journals and conferences.

Yixin Chen received the PhD degree in computer science from the University of Illinois at Urbana Champaign, in 2005. He is currently a full professor of Computer Science at Washington University in St. Louis, MO, USA. His research interests include artificial intelligence, data mining, deep learning, and big data analytics. He has published more than 210 papers on premier international journals and conferences, including AIJ, JAIR, IEEE TSC, IEEE TPDS, IEEE TC, IEEE TKDE, IEEE TII, IJCAI, AAAI, ICML, KDD, etc. He won the Best Paper Award at AAAI and a best paper nomination at KDD. He received an Early Career Principal Investigator Award from the US Department of Energy and a Microsoft Research New Faculty Fellowship. He was an Associate Editor for the ACM Transactions on Intelligent Systems and Technology, IEEE Transactions on Knowledge and Data Engineering, and Journal of Artificial Intelligence Research. He is a Fellow of the IEEE, AAAI and AAIA.