Information Processing and Management 62 (2025) 104219

Contents lists available at ScienceDirect

Information Processing and Management

journal homepage: www.elsevier.com/locate/ipm

LLM-enhanced service Semantic Representation and Category co-occurrence feature Augmentation for Web API recommendation

Guobing Zou [a,*], Pengtao Li [a], Song Yang [a], Shengxiang Hu [a], Shengye Pang [a], Yanglan Gan [b]

[a] School of Computer Engineering and Science, Shanghai University, Shanghai, 200444, China
[b] School of Computer Science and Technology, Donghua University, Shanghai, 201620, China

ARTICLE INFO

Keywords:
Web API recommendation
Service Semantic Representation
Mashup-API category co-occurrence graph
High-order service feature augmentation

ABSTRACT

Web API recommendation for mashup development has become increasingly challenging due to the rapid growth of available APIs. Current approaches face two critical limitations: semantic inconsistency in service descriptions and insufficient exploitation of categorical relationships. To address these challenges, we propose SRCA (LLM-enhanced service Semantic Representation and Category co-occurrence feature Augmentation), a novel framework that integrates LLMs' sophisticated semantic processing capabilities with graph-based category co-occurrence relationship modeling and feature enhancement for Web API recommendation. At its core, SRCA employs specially designed requirement-focused and functional-oriented prompts to guide LLMs in transforming diverse service descriptions into unified structures, while leveraging the LLMs' comprehensive semantic understanding capabilities to extract accurate functional features. This semantic understanding is further enhanced by a feature augmentation model that constructs a mashup-API category co-occurrence graph to discover complex service relationships by capturing both explicit categorical similarities and implicit functional correlations while preserving essential semantic characteristics. Extensive experiments on a large-scale dataset from ProgrammableWeb, comprising 7739 mashups and 1342 Web APIs across 443 categories, demonstrate SRCA's superior performance over state-of-the-art baselines, with significant improvements in recommendation accuracy (13.89% in Precision@5, 16.96% in MAP@5, and 12.48% in NDCG@5).

1. Introduction

Web APIs have emerged as fundamental building blocks in modern software development, driven by the widespread adoption of cloud computing and microservice architectures. These programmatic interfaces enable seamless access to remote data and functionality across diverse platforms and devices, facilitating rapid application development without requiring deep knowledge of underlying implementations (Lizarralde et al., 2020). This architectural paradigm has given rise to mashup applications, which empower developers to create sophisticated composite web applications by integrating multiple Web APIs, rather than repeatedly implementing similar functionalities across different projects (Imran et al., 2012).

* Correspondence to: 99 Shangda Road, Baoshan District, Shanghai, 200444, China.
E-mail addresses: gbzou@shu.edu.cn (G. Zou), 22721507@shu.edu.cn (P. Li), yangsong@shu.edu.cn (S. Yang), shengxianghu@shu.edu.cn (S. Hu), pangsy@shu.edu.cn (S. Pang), ylgan@dhu.edu.cn (Y. Gan).

https://doi.org/10.1016/j.ipm.2025.104219
Received 11 May 2025; Accepted 11 May 2025
Available online 2 June 2025

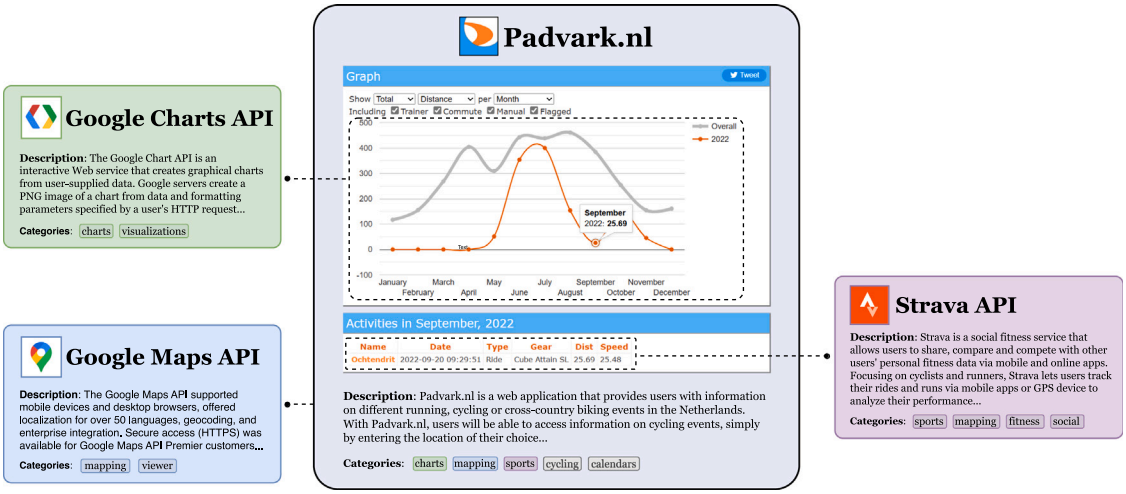

Fig. 1. An illustrative example of mashup application: Padvark.nl integrates three Web APIs (Strava, Google Maps, and Google Charts) to create a comprehensive cycling event platform.

1.1. Motivation

To illustrate the challenges in mashup development, consider Padvark.nl, as shown in Fig. 1, a web application that provides information about running and cycling events in the Netherlands. This mashup integrates three distinct APIs: the Strava API for accessing fitness activity data, the Google Maps API for location visualization, and the Google Charts API for data representation. While each API serves a specific purpose, the challenge lies in identifying these optimal API combinations from thousands of available options. For instance, ProgrammableWeb, the world's largest API registry, currently catalogs over 24,000 APIs across approximately 500 categories, making manual API selection increasingly impractical (Wang, Xiang, et al., 2023).

Recent research has attempted to address this challenge through various approaches (Rahman & Liu, 2020; Yu et al., 2023), broadly categorized into three types: collaborative filtering (CF)-based, content-based, and deep learning-based approaches. CF-based approaches (Ul Ain et al., 2022; Zheng et al., 2011) leverage historical mashup cases to identify APIs that align with current requirements. Content-based approaches (Li et al., 2017; Wang et al., 2019) employ text mining and machine learning techniques, including topic modeling and text clustering, to extract semantic features from service descriptions. More recently, deep learning approaches (Cao et al., 2020; Wang et al., 2024; Wang, Xi, & Yin, 2023) have been developed to model complex relationships between mashups and APIs. However, these existing solutions face two critical limitations:

Semantic Inconsistency and Information Noise in Service Descriptions: Web API and mashup descriptions are typically written by different developers using varying vocabularies and styles, leading to semantic ambiguity. Current approaches using traditional natural language processing or topic modeling (Rahman & Liu, 2020; Sang et al., 2023; Zhong et al., 2018) struggle to maintain semantic consistency and often fail to effectively filter out irrelevant information while preserving essential functional details. While recent studies have attempted to address this issue through advanced text processing techniques (Shi et al., 2019; Wu et al., 2022), the fundamental challenge lies in achieving both standardized vocabulary usage and precise functional feature extraction. Recent breakthroughs in Large Language Models (LLMs) offer promising solutions through their sophisticated semantic understanding capabilities and in-context learning abilities, which enable unified description processing while maintaining semantic coherence across diverse expression styles.

Insufficient Exploitation of Category-based Functional Relationships: While existing approaches have made progress in processing textual descriptions (Kang, Liang, et al., 2024; Wu et al., 2022), they often underutilize the valuable categorical information in Web APIs and mashups. Category labels not only provide precise summaries of service functionalities but also reveal important patterns of service compatibility and complementarity. Although some recent studies have explored service relationship networks (Wang et al., 2024; Wang, Xi, & Yin, 2023), they are still insufficient in taking full advantage of the multi-order relationships and implicit functional patterns that emerge from service category co-occurrence structures. This limitation hinders the effective identification and utilization of functionally compatible and complementary services, influencing the representation and learning of deep latent service features for Web API recommendation.

1.2. Research objectives

Motivated by the limitations of existing approaches in handling heterogeneous service descriptions and utilizing category relationships, we aim to develop a novel framework for Web API recommendation that effectively bridges semantic understanding and category-based functional relationship mining. Specifically, our research objectives are:

(1) To develop an effective approach for unifying diverse service descriptions and extracting rich semantic representations through LLMs, addressing both the semantic inconsistency challenge and the need for comprehensive functional understanding in service descriptions.

(2) To design a novel approach for comprehensively modeling and utilizing the complex functional relationships embedded in service categories, addressing the insufficient exploitation of categorical information in existing approaches.

1.3. Contributions

To achieve these objectives, we propose a novel framework called LLM-enhanced service **S**emantic **R**epresentation and **C**ategory co-occurrence feature **A**ugmentation (SRCA) for Web API recommendation. The main contributions of our work are:

- We introduce an LLM-enhanced semantic representation approach that seamlessly integrates service description unification and semantic feature extraction. Through carefully designed requirement-focused prompt for mashup and functional-oriented prompt for API, it guides LLMs to transform diverse and noisy service descriptions into structured representations while maintaining semantic consistency, enabling more precise identification of service descriptions for better semantic feature representation.
- We develop a novel feature augmentation approach based on a mashup-API category co-occurrence graph that captures both explicit and implicit service relationships. Through a carefully designed graph neural network architecture with neighbor feature aggregation and residual feature integration, this mechanism effectively propagates and integrates categorical knowledge to enhance service representations, leading to more comprehensive understanding of multi-order service relationships and complementary functionalities, particularly valuable for scenarios with incomplete requirements.
- We conduct extensive experiments on a large-scale real-world dataset from ProgrammableWeb, comprising 7739 mashups and 1342 Web APIs across 443 categories. Through detailed ablation studies, we validate the effectiveness of both components, with our approach achieving significant improvements over state-of-the-art baselines (e.g., 13.89% in Precision@5, 16.96% in MAP@5). The dataset is publicly available on our laboratory's website.[1]

The remainder of this paper is organized as follows. Section 2 primarily reviews the current research on Web API recommendation. Section 3 defines and formulates the Web API recommendation problem. Section 4 presents the approach in detail. Section 5 shows and analyzes the experimental results. Finally, Section 6 discusses the proposed SRCA in brief, and Section 7 summarizes the paper.

2. Related work

2.1. CF-based web API recommendation

The development of Web API recommendation has been significantly influenced by collaborative filtering techniques, which analyze historical mashup-API relationship patterns to guide API recommendation. Cao et al. (2014) propose a mashup service recommendation approach based on content similarity and collaborative filtering. Chen et al. (2014) focused on QoS-aware service selection by integrating user-based and item-based collaborative filtering approaches, enabling more accurate Web service recommendations through comprehensive QoS information analysis. Rahman et al. (2017) employed matrix factorization techniques to extract features of mashups and Web APIs, and through a two-level topic model, clustered mashup services to recommend Web APIs. A significant advancement came with context-sensitive consideration, as demonstrated by Wu et al. (2018) who proposed a context-sensitive matrix-factorization collaborative QoS approach (CSMF), leveraging implicit and explicit contextual factors in QoS data. Recent advancements in recommendation strategies have explored variant probabilistic models, as shown in Yao et al. (2021)'s work on probabilistic matrix factorization with implicit correlation regularization, which enhanced recommendation diversity while maintaining accuracy.

While these technical advancements have significantly improved recommendation performance, CF-based approaches continue to face fundamental challenges in extracting semantic features, particularly in scenarios with service semantic inconsistencies and diverse information noises.

2.2. Content-based Web API recommendation

Content-based approaches have evolved from simple text matching to sophisticated semantic analysis techniques, focusing on extracting and analyzing semantic features from service descriptions. Early content-based approaches primarily relied on basic text processing techniques (He, Zhou, Zhang, Wang, Ye, Chen, Chen, et al., 2017; He, Zhou, Zhang, Wang, Ye, Chen, Grundy, & Yang, 2017). These foundational techniques were later enhanced through hybrid approaches that integrated content analysis with collaborative elements, as demonstrated by Yao et al. (2013), who adopted a three-way aspect model considering both service rating data and descriptions.

[1] https://scdm-shu.github.io/papers/datasets/PW-Dataset.zip

Recent advancements have focused on multi-dimensional feature analysis and sophisticated modeling techniques. Xia et al. (2015) proposed a category-aware service clustering and recommendation approach. Cao et al. (2020) further advanced this direction by proposing an integrated content and network-based clustering approach. More recent developments have leveraged advanced natural language processing techniques, as shown by Pan et al. (2023)'s work combining TextRank with BERT embeddings, and Ren and Wang (2022)'s distribution-aware SVM approach.

However, while content-based approaches have evolved significantly, they still face certain challenges. These approaches primarily focus on shallow semantic features from service descriptions, limiting their ability to capture deeper functional relationships. Moreover, their recommendation performance heavily relies on description quality when evaluating mashup-API relevance, making them vulnerable to incomplete requirement specifications in real-world scenarios.

2.3. Deep learning-based Web API recommendation

The emergence of deep learning technologies has brought new opportunities for addressing the semantic consistency and functional completeness challenges in Web API recommendation. Recent studies have leveraged various deep neural network architectures to model complex service relationships and extract deep semantic features.

Traditional deep learning approaches primarily focus on processing service descriptions and semantic features. Shi et al. (2019) propose a service recommendation approach based on description expansion and deep neural networks, employing an LSTM model with attention mechanisms to mine functionally relevant features. Wu et al. (2022) introduce a multi-model fusion approach combining CF-based and content-based approaches with multi-task learning optimization. This multi-task learning framework jointly optimizes API recommendation and mashup category prediction, demonstrating the benefits of learning shared representations across related tasks. Gu et al. (2022) proposed a compositional semantics-based service bundle approach to address the semantic gap between mashup and service descriptions.

Recent advancements have particularly focused on graph neural networks (GNNs) for modeling complex service relationships. To address the challenge of sparse semantic features, Kang, Liang, et al. (2024) proposed integrating network structure information with content analysis, while their subsequent work (Kang, Wang, et al., 2024) combined keyword-driven service recommendation with GNNs. Liu et al. (2023) proposed a dynamic graph network to address service evolution, while Wang et al. (2024) introduced a semantic-enhanced heterogeneous graph network for handling service feature interactions. Hu (2024) further extend the investigation of addressing this issue by introducing biased random walks on heterogeneous networks to capture diverse connectivity patterns.

Despite these advancements, deep learning-based approaches still face challenges in mining high-order latent features for effective Web API recommendations. Current semantic modeling approaches struggle to fully bridge the gap between user requirements and service descriptions, particularly with domain-specific and implicit functional requirements. Although GNN-based approaches have improved the modeling of service relationships, they face difficulties in capturing complex compatibility patterns in dynamic service ecosystems. To address these limitations, we propose SRCA, a novel framework that aims to bridge the semantic gap between user requirements and service descriptions while capturing rich contextual information for Web API recommendation, thereby leading to a more effective service recommendation.

3. Problem formulation

To formally present the Web API recommendation problem, we introduce the following definitions.

Web API. Within a Web service repository S, let $S_A = \{a_1, a_2, \ldots, a_p\}$ represent the set of all registered Web APIs. Each Web API $a \in S_A$ is denoted as $a = \langle D_a, C_a \rangle$. Here, D_a is the functional description of a, and C_a is the set of category labels assigned to a from C, the universal set of predefined category labels within the repository (i.e., $C_a \subset C$).

Mashup Service. Similarly, let $S_M = \{m_1, m_2, \ldots, m_q\}$ represent the set of all registered mashup services in S. Each mashup service $m \in S_M$ is denoted as $m = \langle D_m, C_m, I_m \rangle$. Here, D_m is the requirement description, $C_m \subset C$ is the set of category labels associated with m (drawn from the same universal set C), and $I_m \subset S_A$ is the collection of Web APIs used to implement m.

Task Formulation. Given the Web service repository S containing the API set S_A and the mashup set S_M, the Web API recommendation task aims to identify suitable APIs for implementing new mashup services. Formally, for a given mashup requirement R, the task is to learn a recommendation function $\Omega(R, S_A, S_M) \rightarrow R_A$, where $R_A \subseteq S_A$ represents the recommended set of APIs.

4. Methodology

Fig. 2 illustrates the overall framework of SRCA, which consists of three key components that work synergistically to achieve accurate Web API recommendation:

(1) LLM-Enhanced Service Semantic Representation. This component leverages LLMs to enhance service semantic representation through unified description processing. Through carefully designed requirement-focused and functional-oriented prompts, it first transforms diverse service descriptions into unified structures that highlight core functionalities and requirements. The approach then utilizes the LLM's internal representations to extract rich semantic features, creating unified semantic representations that accurately capture service characteristics while maintaining semantic consistency.

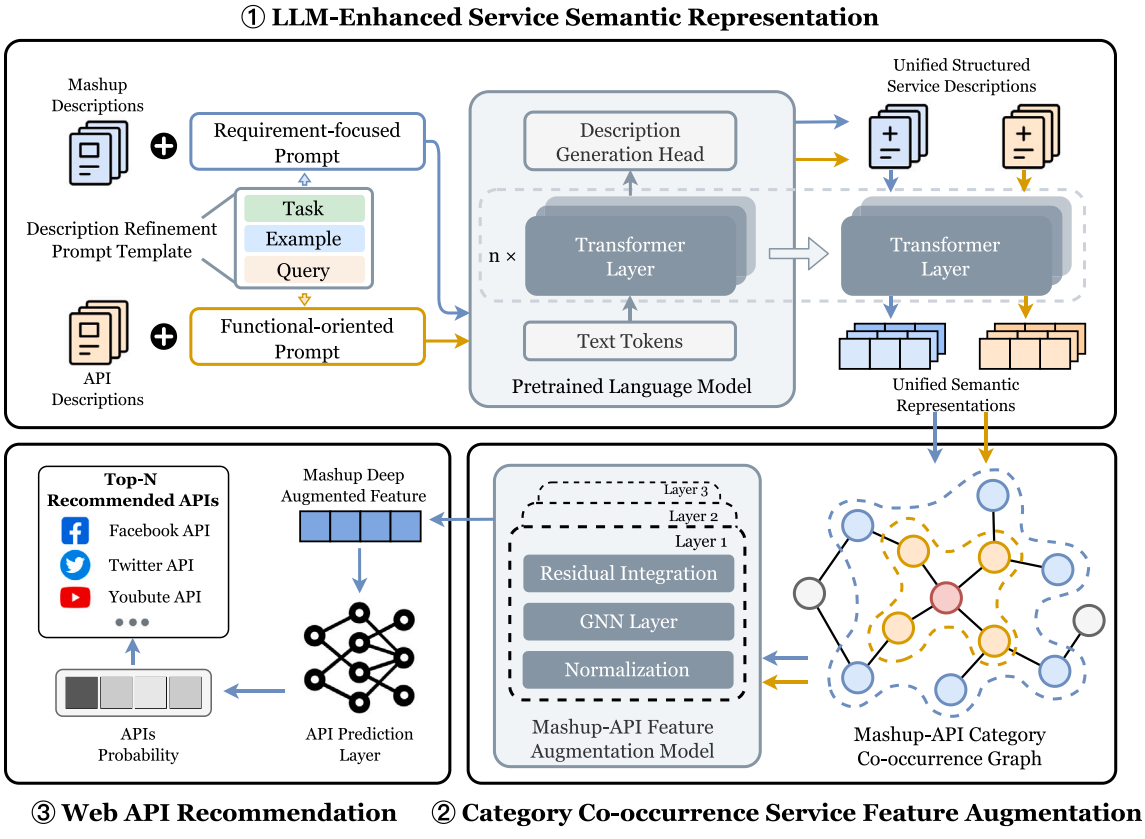

Fig. 2. The Framework of SRCA for Web API Recommendation.

(2) Category Co-occurrence Service Feature Augmentation. This component introduces a feature augmentation model that enriches service representations by leveraging category-based functional relationships. It constructs a mashup-API category co-occurrence graph that captures both explicit categorical similarities and implicit functional correlations between services. Through feature propagation and integration mechanisms, the model effectively augments the original semantic features with rich functional contexts derived from category co-occurrence patterns. The enhanced representations combine deep semantic understanding with comprehensive functional relationships, resulting in more expressive and enriched service features.

(3) Web API Recommendation. Building upon the service semantic features and category-based augmentation, this component implements a recommendation model that learns sophisticated mappings between mashup requirements and API functionalities. By formulating the recommendation task as a multi-label classification problem, it employs a carefully designed multi-layer perceptron architecture to capture complex non-linear relationships in the enriched feature space. Through rigorous optimization of the learned mappings, the model effectively transforms mashup requirements into precise API relevance scores, generating accurate API recommendations that align with each mashup's specific needs.

4.1. LLM-enhanced service semantic representation

This section presents an LLM-enhanced service semantic representation approach addressing the two challenges in service descriptions: semantic inconsistency and information noise. These challenges significantly impact the accuracy of service semantic feature representation, as successful Web API recommendations heavily depend on precise matching between mashup requirements and API descriptions. The choice of leveraging LLMs for service description processing is theoretically grounded in their unique capabilities. Through extensive pre-training on diverse textual data, LLMs have developed sophisticated semantic understanding abilities that enable standardized vocabulary usage, core information identification, and semantic consistency maintenance across different service descriptions. These capabilities are particularly valuable for service description processing, where the goal is to extract precise functional features while maintaining semantic coherence across diverse expression styles. Through specially designed prompts, it first transforms diverse and noisy service descriptions into unified structures. Then, leveraging the rich internal representations of LLMs, which encode both general language patterns and domain-specific knowledge, LLMs can effectively capture service semantic representations in terms of functional characteristics of both mashups and APIs.

4.1.1. Service description unification with task-specific prompts

Service descriptions for mashups and Web APIs present two significant challenges for recommendation tasks. First, these descriptions are typically written in natural language by developers worldwide, leading to considerable variations in vocabulary choice and expression styles. Second, Web API descriptions often contain substantial noise information unrelated to functional matching, such as version histories and release dates. These challenges significantly impact the accuracy of recommendation systems, as successful mashup recommendations heavily depend on precise matching between mashup requirements and Web API capabilities (Kang, Wang, et al., 2024; Wang, Xi, & Yin, 2023; Wu et al., 2022).

The primary goal of a language model is to estimate the probability of the next word w_n given a sequence of words $w_1, w_2, \ldots, w_{n-1}$, a concept that can be formalized as the probability formula $P(w_n|w_1, w_2, \ldots, w_{n-1})$. Currently, the development of language models has evolved into large-scale models, such as GPT-4(OpenAI et al., 2024) and LLaMA 3(Grattafiori et al., 2024). These LLMs, through pre-training on extensive datasets, have mastered complex language rules and rich knowledge, thus exhibiting exceptional performance in semantic understanding and text generation. Particularly in processing the descriptions of mashup services and Web API services, LLMs can accurately identify key functional information within complex natural language descriptions and effectively rewrite these descriptions to focus more on core functionalities. Therefore, we choose to use LLMs to unify the descriptions of mashups and Web APIs, thereby enhancing the accuracy of functional matching between mashups and Web APIs.

However, due to the typically large parameter size of LLMs, employing traditional task-specific fine-tuning approaches becomes impractical. This impracticality stems from several critical challenges: substantial computational resources required for parameter updates, extensive storage requirements for maintaining multiple fine-tuned models, and the risk of catastrophic forgetting during the fine-tuning process. To effectively address these challenges, prompting engineering techniques have been proposed. These techniques fundamentally leverage LLMs' inherent in-context learning (ICL) capabilities. Rather than actual learning through parameter updates, ICL enables the activation of skills already embedded within the model through carefully crafted prompts containing exemplars and/or instructions (Dong et al., 2024; Rubin et al., 2022). This mechanism allows models to perform specific tasks by providing appropriate context with the designed task-oriented prompt, effectively utilizing the knowledge acquired during pre-training.

Effective prompt engineering typically follows several key design principles (Schulhoff et al., 2025). First, structured prompt design helps guide the model's attention and response generation by providing clear, organized instructions. Second, the inclusion of few-shot examples demonstrates the desired output format and style, enabling the model to learn from these demonstrations and apply similar patterns to new inputs. These principles have shown remarkable effectiveness in helping LLMs generate precise, task-specific outputs while maintaining the consistency across different inputs. Specifically, we design task-specific prompts T based on the objective of description processing, which are then combined with the description D of the mashup or Web API to form a prompt. This process can be represented as:

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

where concat() represents the concatenation operation, and X is the composite sequence that integrates task-specific guidance information with the relevant description. Subsequently, this prompt is fed into a pre-trained LLM $\mathcal{M}$. $\mathcal{M}$ uses an autoregressive approach to predict the next word in the sequence based on the prompt X and the already generated part of the result sequence $\{y_1, y_2, \ldots, y_{i-1}\}$:

$$y_i = \arg\max_{c_i} P(c_i|y_{i-1}, y_{i-2}, \ldots, y_1, X) \tag{2}$$

where c_i represents a word in the LLM's vocabulary. By maximizing these conditional probabilities, we can determine the entire sequence and ultimately generate a complete output sequence $Y = \{y_1, y_2, \ldots, y_N\}$. Integrating the above steps, the process of using an LLM to process mashup and Web API descriptions through prompting engineering can be formalized as follows:

$$Y = \mathcal{M}(\text{concat}(T, D)) \tag{3}$$

Based on this prompting strategy, we implement two specialized prompts for service description unification, as shown in Fig. 3: a Requirement-focused Prompt for Mashup (RPM) and a Functional-oriented Prompt for API (FPA). Our prompt design consists of three specialized components that collaborate together to enhance service descriptions by effectively leveraging LLM capabilities to deal with semantic inconsistencies and information noise reduction. The Task component establishes a technical documentation expert role with clear reorganization guidelines, which helps standardize vocabulary usage and guides the focus toward essential functional information while filtering out those functionally irrelevant descriptions. The Example component provides carefully selected service description examples that demonstrate the desired transformation patterns, effectively activating the LLM's ICL capabilities for accurate and consistent service semantic feature representation. The Query component ensures processing stability through a uniform input format, facilitating reliable and coherent description transformation.

Within the Task component, we implement differentiated designs for mashup and API descriptions to better capture their unique characteristics. For mashup descriptions, we focus on extracting requirement-focused information by unifying descriptions into four complementary aspects: functional requirements that specify core service needs, target scenarios that outline application contexts, target users that define the scope of terminal clients, and expected outcome that describe the desired results of mashup requirement description unification. This structured analysis of requirements provides a comprehensive basis for identifying suitable API candidates. In contrast, for API descriptions, we focus on extracting functional-oriented information through four corresponding sections: core functionality that presents primary service capabilities, key features that highlight technical characteristics, use cases that illustrate application scenarios, and primary benefits that emphasize distinctive advantages of API services. This systematic

Requirement-focused Prompt for Mashup

 # Task
You are a technical documentation expert.
Your task is to **reconstruct the given mashup description** into a unified format that highlights key requirements.
Please analyze the description and reorganize it into the following sections:
1. Functional Requirements: What are the main functions this mashup needs?
2. Target Scenario: In what scenarios would this mashup be used?
3. Target Users: Who are the intended users?
4. Expected Outcome: What results or benefits are expected?
Requirements for the response:
- Use clear, formal, and consistent language
- Focus on essential information only
- Keep each section concise but informative
- Maintain technical accuracy
- Exclude implementation details or technical specifications

 # Example
Input
Mashup Name: Padvark.nl
Original Description: Padvark.nl is a web application that ...
Output
Functional Requirements:
- Location-based event search functionality
- ...

Target Scenario:
- Finding local sports events
- ...

.....

 # Query
Mashup Name: {mashup_name}
Original Description: {mashup_description}

Functional-oriented Prompt for API

 # Task
You are a technical documentation expert.
Your task is to **reconstruct the given API description** into a unified format that emphasizes its key functional features.
Please analyze the description and reorganize it into the following sections:
1. Core Functionality: What are the primary functions this API provides?
2. Key Features: What are the distinctive technical features or capabilities?
3. Use Cases: What are the main application scenarios?
4. Primary Benefits: What advantages does this API offer?
Requirements for the response:
- Use clear, formal, and consistent language
- Focus on functional aspects only
- Keep each section concise but informative
- Maintain technical accuracy
- Exclude non-functional information (e.g., version history, release dates)

 # Example
Input
API Name: Google Maps
Original Description: The Google Maps API allow for the ...
Output
Core Functionality:
- Provides comprehensive mapping and location-based services
- ...

Key Features:
- Cross-platform compatibility (mobile and desktop)
- ...

.....

 # Query
API Name: {api_name}
Original Description: {api_description}

Fig. 3. Requirement-focused Prompt for Mashup (RPM) and Functional-oriented Prompt for API (FPA).

organization of API capabilities enables precise matching with mashup requirements. The parallel structure between requirement analysis and capability description creates clear semantic correspondence points, allowing for more accurate feature comparison and alignment in the recommendation process, thereby improving the accuracy of Web API recommendation for mashup creation.

The complete unification process through LLM can be formalized by defining separate transformations for both mashup and API descriptions:

$$\begin{aligned} D'_m &= \mathcal{M}(\text{concat}(T_m, D_m)) \\ D'_a &= \mathcal{M}(\text{concat}(T_a, D_a)) \end{aligned} \tag{4}$$

where T_m and T_a denote the requirement-focused and functional-oriented prompts, D_m and D_a represent the original descriptions, and D'_m and D'_a are the unified structured and enhanced service descriptions, respectively.

4.1.2. Semantic representation through LLM internal states

Traditional approaches to extracting semantic representations from service descriptions typically face significant limitations. Task-specific models like CNN (Kim, 2014) and LSTM (Sutskever et al., 2014) require complex architectural redesign for specific tasks and extensive training data. Meanwhile, general-purpose sentence embedding models such as BERT (Devlin et al., 2019) and BGE (Xiao et al., 2024) may not effectively capture domain-specific semantic nuances.

To overcome these limitations, we propose leveraging the rich internal representations of LLMs, which offer several unique advantages. Through extensive pre-training on diverse texts, LLMs have developed sophisticated semantic understanding capabilities that span multiple levels of abstraction. Their internal states encode both general language patterns and domain-specific knowledge, making them particularly suited for capturing the nuanced semantics of service descriptions. Additionally, these capabilities are immediately available without requiring task-specific training or fine-tuning, offering a robust and efficient solution for semantic feature extraction.

The choice of utilizing LLM's internal states is grounded in established findings from LLM interpretability research (Geva et al., 2021; Rogers et al., 2020). Studies have shown that different layers in transformer-based models capture distinct aspects of linguistic and semantic information, with higher layers typically encoding more abstract semantic concepts. Specifically, the final layer has been found to represent high-level semantic abstractions that are ideal for capturing service functionality and requirements. This aligns perfectly with our need to extract meaningful semantic features that reflect the essential characteristics of both mashups and APIs.

Based on these theoretical insights into LLM interpretability, we implement our semantic feature extraction process as follows. When processing a unified description D' (i.e., D'_m or D'_a) through the LLM, we obtain a sequence of hidden states $\mathbf{H} = [\mathbf{h}_1, \mathbf{h}_2, \ldots, \mathbf{h}_n]$ from its final layer, where $\mathbf{h}_i \in \mathbb{R}^d$ represents the hidden state for token i, and d is the hidden dimension. These hidden states

are particularly valuable as they capture both token-level semantics and global contextual information, thanks to the self-attention mechanisms in the transformer architecture. To derive a fixed-length semantic vector that comprehensively represents the structured description, we employ mean pooling over these hidden states:

$$\mathbf{e}_{D'} = \frac{1}{n}\sum_{i=1}^{n}\mathbf{h}_i \tag{5}$$

where $\mathbf{e}_{D'} \in \mathbb{R}^d$ is the final semantic representation. While alternative pooling strategies such as max pooling or attention-weighted pooling exist, mean pooling offers an optimal balance between computational efficiency and semantic preservation. Specifically, the averaging operation helps capture the aggregate semantic information while being robust to sequence length variations and potential noise in individual token representations. This process is applied consistently to both mashup descriptions D'_m and Web API descriptions D'_a, resulting in feature matrices $\mathcal{E}_m \in \mathbb{R}^{|S_M|\times d}$ and $\mathcal{E}_a \in \mathbb{R}^{|S_A|\times d}$ respectively.

The semantic representation, working synergistically with service description unification, creates a powerful foundation for further service feature augmentation through three key mechanisms: (1) rich semantic understanding derived from LLM's pre-trained knowledge, eliminating the need for task-specific training; (2) comprehensive yet focused representation of service functionalities through unified semantic aspects guided by task-specific prompts; and (3) precise functional alignment between mashup requirements and API capabilities through parallel structural organization. Maintaining semantic consistency from unification through extraction provides the foundation for further category co-occurrence service feature augmentation, which significantly enhances the service feature learning for better Web API recommendation in mashup creation.

4.2. Category co-occurrence service feature augmentation

This section presents a graph-based feature augmentation approach that addresses the challenges in exploiting categorical relationships for service recommendation. While the semantic representations obtained through our LLM-enhanced approach provide valuable functional features, they primarily capture individual service characteristics. In real-world scenarios, mashup requirements often focus primarily on core functionalities while omitting important contextual information about functionally-similar services and related capabilities. Motivated by the above limitation, we model service relationships through graph structures. Specifically, we construct a category co-occurrence graph where services sharing the same categories are connected, naturally indicating their functional similarities. Through the constructed graph structure, we can leverage both explicit similarities between directly connected services and discover implicit relationships between services connected through intermediate nodes. We then employ graph neural networks to effectively learn these high-order relationships through iterative message passing and feature aggregation, further augmenting the service feature extraction based on the independent service semantic representation.

4.2.1. Mashup-API category co-occurrence graph

The effectiveness of Web API recommendation heavily relies on comprehensive understanding of service relationships. While semantic features provide valuable insights into service functionalities, they may not fully capture the complex relationships between services, particularly in scenarios where mashup requirements are incomplete or focus primarily on core functionalities. To address this challenge, we consider two complementary perspectives for modeling service relationships.

First, from the functional organization perspective, category labels serve as precise and succinct summaries of service functionalities, naturally grouping services with similar or complementary capabilities together (Wang, Xi, & Yin, 2023; Wu et al., 2022). Second, from the structural perspective, network-based modeling approaches have demonstrated remarkable capability in revealing complex service relationships that are difficult to observe directly from semantic aspects (Cao et al., 2023; Xiao et al., 2020; Yan et al., 2021). These two perspectives complement each other, with category labels providing rich functional context and network structures enabling the discovery of implicit functional patterns.

Based on these perspectives, we propose to construct a mashup-API category co-occurrence graph that captures both explicit categorical similarities and implicit functional relationships between services. As illustrated in Fig. 4, the graph construction process begins by representing both mashups and Web APIs as nodes, with their initial features derived from the LLM-enhanced unified semantic representations. Services are then connected based on their category overlap, with edge weights determined by the degree of categorical similarity.

Formally, we define the mashup-API category co-occurrence graph as $G = (S, L, W)$, where:

- $S = S_M \cup S_A$ represents the set of nodes, including mashup services S_M and Web API services S_A
- W contains edge weights w_{ma} between mashup m and API a, calculated using the Jaccard coefficient of their category sets:

$$w_{ma} = \text{Jaccard}(C_m, C_a) = \frac{|C_m \cap C_a|}{|C_m \cup C_a|} \tag{6}$$

- L defines the set of edges between services with sufficient category overlap:

$$L = \{(m, a) | w_{ma} \geq \theta, m \in S_M, a \in S_A\} \tag{7}$$

where θ is a threshold controlling edge formation.

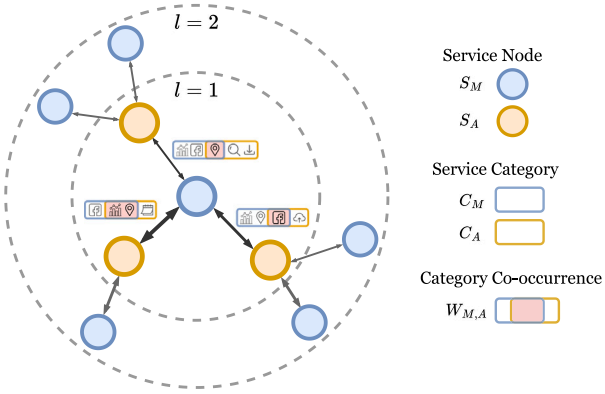

Fig. 4. Illustration of mashup-API category co-occurrence graph structure.

Through this graph structure, we effectively integrate the rich functional context from category labels with the powerful modeling capability of network structures. The weighted connections not only capture explicit functional correlations through direct category relationships, but also enable the discovery of implicit functional patterns through structural propagation. This comprehensive modeling of service relationships provides essential complementary information to semantic features, particularly valuable for identifying functionally compatible and complementary services in complex mashup scenarios where requirements may be incomplete or primarily focused on core functionalities.

4.2.2. High-order service feature aggregation

While the mashup-API category co-occurrence graph effectively models the complex functional relationships between services, fully leveraging this structural information for service recommendation requires sophisticated feature aggregation and extraction mechanisms. To this end, we design a graph neural network-based model that progressively aggregates and extracts high-order service features. The key technical challenge in this process is to effectively balance three competing objectives: maintaining the stability of feature propagation across multiple aggregation layers, preserving the discriminative power of individual service features, and capturing complex higher-order relationships in the service interaction patterns. We address these challenges through a carefully crafted architecture that combines feature aggregation operations with residual connections for preserving the original semantic features, ensuring both effective high-order integration and semantic consistency in the extracted representations.

As an initial step, before feeding the LLM-enhanced service unified semantic representations into the graph neural network, we first apply a dimension mapping layer to accommodate varying hidden dimensions from different LLMs:

$$\mathbf{e}_i^{(0)} = W\mathbf{e}_i + b \tag{8}$$

where $\mathbf{e}_i$ is the original semantic feature of node i (either mashup or API), and W and b are learnable parameters. This mapping ensures consistent feature dimensions throughout the network while preserving the semantic information captured by the LLM.

As illustrated in Fig. 5, our model processes features through multiple graph convolution layers to capture increasingly complex category relationships. For a specific target mashup node, its first-order neighbors (i.e., directly connected API nodes) provide functional information directly related to specific categories. Second-order neighbors (i.e., other mashup nodes connected through these APIs) reflect broader usage patterns and complementary functionalities (Yan et al., 2021), while higher-order neighbors capture the global context of service relationships. This multi-hop information aggregation allows our model to understand both direct functional similarities and indirect service relationships.

The feature propagation at each layer comprises three key components that work in concert to effectively capture and integrate these multi-order relationships:

(1) **Neighbor Feature Aggregation:** For each node, we aggregate information from its immediate neighbors using weighted message passing:

$$\begin{aligned}\mathbf{x}_m^{(k+1)} &= \sum_{a\in\mathcal{N}_m} \frac{w_{ma}}{\sqrt{|\mathcal{N}_m|}\sqrt{|\mathcal{N}_a|}}\mathbf{e}_a^{(k)} \\ \mathbf{x}_a^{(k+1)} &= \sum_{m\in\mathcal{N}_a} \frac{w_{am}}{\sqrt{|\mathcal{N}_a|}\sqrt{|\mathcal{N}_m|}}\mathbf{e}_m^{(k)}\end{aligned} \tag{9}$$

where $\mathcal{N}_m$ and $\mathcal{N}_a$ represent the neighbor sets of mashup m and API a respectively. The normalization term $\frac{1}{\sqrt{|\mathcal{N}_i|}\sqrt{|\mathcal{N}_j|}}$ ensures balanced feature propagation by preventing high-degree nodes from dominating the feature aggregation while maintaining numerical stability.

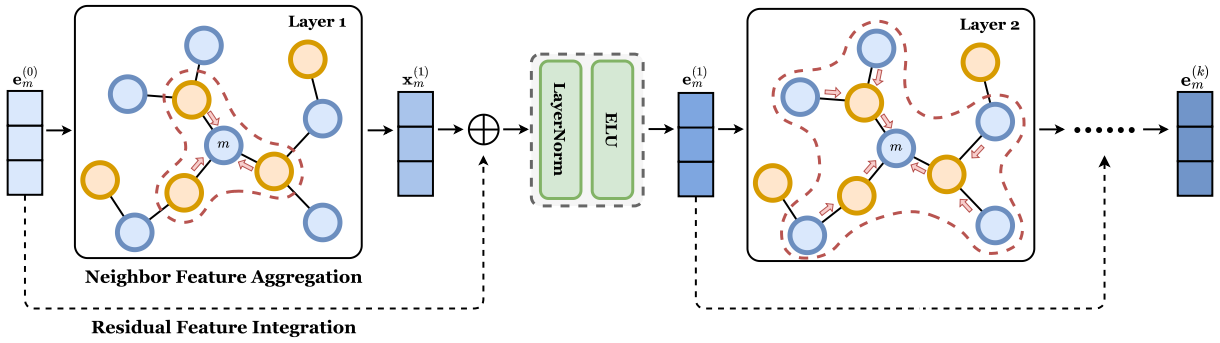

Fig. 5. The detail of graph-based mashup-API feature augmentation model.

(2) **Residual Feature Integration:** The aggregated neighbor features are then combined with the node's current features through residual connections, which help preserve the original semantic information and facilitate gradient flow during training. To prevent feature distribution shift and over-smoothing (a common problem in deep GNNs where node features become increasingly homogeneous), we apply layer normalization to the combined features:

$$\begin{aligned}\mathbf{y}_m^{(k+1)} &= \text{LayerNorm}(\mathbf{e}_m^{(k)} + \mathbf{x}_m^{(k+1)}) \\ \mathbf{y}_a^{(k+1)} &= \text{LayerNorm}(\mathbf{e}_a^{(k)} + \mathbf{x}_a^{(k+1)})\end{aligned} \tag{10}$$

where LayerNorm performs feature-wise normalization across the feature dimension. Specifically, it first normalizes the features to zero mean and unit variance, then applies learnable affine transformation:

$$\text{LayerNorm}(\mathbf{x}) = \gamma \odot \frac{\mathbf{x} - \mu}{\sqrt{\sigma^2 + \epsilon}} + \beta \tag{11}$$

Here, μ and σ^2 are the mean and variance computed across the feature dimension, ϵ is a small constant for numerical stability, and γ and β are learnable scale and shift parameters that allow the network to modulate the normalized values. This normalization scheme is crucial for stabilizing the training process and maintaining the expressiveness of node representations across multiple graph convolution layers.

(3) **Non-linear Transformation:** To introduce non-linearity while preserving gradient information, we employ the Exponential Linear Unit (ELU) (Clevert et al., 2016) activation function. Compared to alternatives like ReLU, ELU provides smoother gradients and helps mitigate the dying neuron problem through its negative values, making it particularly suitable for deep graph architectures. The transformation is applied as:

$$\begin{aligned}\mathbf{e}_m^{(k+1)} &= \text{ELU}(\mathbf{y}_m^{(k+1)}) \\ \mathbf{e}_a^{(k+1)} &= \text{ELU}(\mathbf{y}_a^{(k+1)})\end{aligned} \tag{12}$$

This non-linear transformation enables the model to capture complex relationships in the category co-occurrence patterns while maintaining stable gradient flow during training.

The feature augmentation model effectively combines graph convolution layers with residual connections to achieve two key objectives: preserving distinctive service semantic features while integrating category co-occurrence functionally-similar service relationships. The residual design maintains node distinctiveness during feature propagation, while layer normalization ensures stable feature distribution across the network. This architecture enables precise integration of high-order category relationships without losing the discriminative power of individual service features, ultimately producing enhanced service representations that capture both independent semantic features and aggregate functionally-similar contextual service features for advancing the performance of Web API recommendation.

4.3. Web API recommendation

Effective Web API recommendation requires rich and comprehensive service representations that can facilitate accurate matching between mashup requirements and API descriptions. The LLM-enhanced Service Semantic Representation excels at extracting fine-grained semantic features from service descriptions through sophisticated language understanding. However, while these semantic features reflect individual service functionalities effectively, they are still limited in capturing the broader service features from mashup-API contextual connections that exist in the service ecosystem. Conversely, although the Category Co-occurrence Feature Augmentation effectively discovers service relationships through implicit categorical patterns, it relies on initially accurate service representations to ensure better service feature propagation and aggregation. This complementary nature motivates the integration of the two aspects: the service semantic representation provides detailed functional characterization of individual services, while the category co-occurrence feature augmentation enriches service feature extraction derived from similar services with related categories, helping to better match the demands between mashup requirements and API descriptions. The resulting augmented

service representations, which consist of both semantic and structural service features, serve as the foundation for facilitating Web API recommendation task. Given a mashup service m, our objective is to generate a ranked list of Web APIs that optimally fulfill its requirements. We formulate this as a multi-label classification problem (Zhang & Zhou, 2014), where each mashup can simultaneously require multiple APIs.

To effectively map the enriched mashup features to API recommendations, we employ a multi-layer perceptron (MLP) architecture. This choice is primarily motivated by MLPs' proven capability to learn complex non-linear mappings between high-dimensional feature spaces — a critical requirement for capturing the complex relationships between mashup requirements and API capabilities. The recommendation process consists of multiple non-linear transformations followed by a final sigmoid activation:

$$\begin{aligned} \mathbf{h}_i &= \alpha(\mathbf{W}_i\mathbf{h}_{i-1} + \mathbf{b}_i), \quad i \in \{1, \dots, n-1\} \\ \hat{\mathbf{y}}_m &= \sigma(\mathbf{W}_n\mathbf{h}_{n-1} + \mathbf{b}_n) \end{aligned} \tag{13}$$

where $\mathbf{W}_i$ and $\mathbf{b}_i$ are the learnable parameters of the ith layer, $\alpha(\cdot)$ is the activation function for intermediate layers, and $\sigma(\cdot)$ is the sigmoid function that maps outputs to probability-like scores in $[0, 1]$. The final output $\hat{\mathbf{y}}_m \in [0, 1]^{|S_A|}$ represents the predicted relevance scores for all available APIs.

4.4. Model optimization

In Web API recommendation, each mashup typically uses only a small subset of available APIs, creating a highly imbalanced learning scenario. To effectively handle this characteristic, we employ the Focal Loss (Lin et al., 2017) as our training objective. For a single mashup, the loss is defined as:

$$\begin{aligned} \mathcal{L}_m(\hat{\mathbf{y}}_m, \mathbf{y}_m) = \quad & -\frac{1}{|A|}\sum_{a \in S_A} (\alpha \mathbf{y}_m[a](1 - \hat{y}_m[a])^\gamma \log(\hat{y}_m[a]) \\ & + (1-\alpha)(1 - \mathbf{y}_m[a])\hat{y}_m[a]^\gamma \log(1 - \hat{y}_m[a])) \end{aligned} \tag{14}$$

where $\mathbf{y}_m[a] = 1$ if mashup m has used API a, and 0 otherwise. The Focal Loss introduces two key parameters that work together to improve learning effectiveness. The balancing factor $\alpha \in [0, 1]$ assigns different weights to positive and negative samples, where a larger α increases the importance of positive samples (used APIs), helping the model maintain sensitivity to the minority class. Meanwhile, the focusing parameter $\gamma \geq 0$ adaptively adjusts sample weights based on prediction confidence — through the modulating factors $(1 - \hat{y})^\gamma$ and $\hat{y}^\gamma$, it automatically down-weights well-classified examples and focuses learning on harder cases where the model is less confident.

To prevent overfitting and improve generalization, we incorporate L2 regularization into the overall objective function:

$$\mathcal{L}(\theta) = \min\left(\frac{1}{|M|}\sum_{m \in M} \mathcal{L}(\hat{\mathbf{y}}_m, \mathbf{y}_m) + \lambda\|\theta\|_2^2\right) \tag{15}$$

where $\theta = \{\mathbf{W}_1, \mathbf{b}_1, \dots, \mathbf{W}_n, \mathbf{b}_n\}$ includes all trainable parameters, and λ controls the strength of regularization.

For model optimization, we employ a combination of effective training strategies. The Adam optimizer (Kingma & Ba, 2015) is adopted for its capabilities of adaptive learning rate adjustment, ensuring stable convergence across different parameter scales. We implement mini-batch training to balance computational efficiency against optimization stability, where the stochastic nature of batch sampling introduces beneficial regularization effects. To prevent overfitting, we employ early stopping strategy by monitoring validation performance with a patience mechanism, ensuring the model maintains strong generalization capabilities without excessive training iterations.

5. Experiments

In order to evaluate the performance of SRCA, we conducted a series of experiments on a real-world dataset and compared it with several state-of-the-art Web API recommendation approaches. These approaches are evaluated in terms of recommendation efficiency, accuracy, and coverage.

5.1. Datasets

For experimental evaluation, we constructed our dataset from ProgrammableWeb, one of the world's largest Web API repositories. The raw dataset consists of 8420 mashups and 21,614 Web APIs, where each mashup contains four types of metadata (name, description, category, and API invoke information) and each API includes three types of metadata (name, description, and category). These real-world service descriptions provide a comprehensive foundation for evaluating API recommendation effectiveness.

To ensure data quality and experimental validity, we conducted several essential preprocessing steps. We first removed invalid mashups and APIs that lacked necessary description information or API invoke records. Additionally, we only retained APIs that had been invoked at least once, ensuring that all included APIs had demonstrated practical utility. Through this careful preprocessing, the final dataset was refined to contain 7739 mashups, 1342 APIs, and 443 unique categories. Table 1 presents the detailed statistics of this processed dataset, highlighting its comprehensive coverage of active APIs and rich category diversity.

Following standard practice in machine learning research, we employed a hold-out validation strategy for experimental evaluation. The dataset was randomly partitioned into three subsets: a training set containing 6193 mashups (80%), a validation

Table 1
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

set with 773 mashups (10%), and a test set comprising 773 mashups (10%). This partition ensures sufficient training data while maintaining representative validation and test sets for reliable model evaluation. The mashups in each subset maintain their complete API invocation records and associated metadata, providing a robust foundation for assessing recommendation performance.

5.2. Evaluation metrics

Since Web API recommendation is fundamentally a recommendation task, we adopt widely-used metrics from recommendation systems. As for mashup creation, these metrics are particularly relevant as mashup developers need effective guidance to discover appropriate APIs from numerous API candidates. To systematically evaluate our recommendation approach, we employ four standard metrics: Precision@N, Recall@N, Mean Average Precision (MAP@N), and Normalized Discounted Cumulative Gain (NDCG@N). These metrics comprehensively assess both recommendation accuracy and ranking quality, which are essential for facilitating the performance of Web API recommendation.

We evaluate these metrics at $N = \{5, 10, 15, 20\}$, aligning with the practical requirements of mashup creation where developers typically need to identify and integrate multiple complementary APIs. These different ranges help assess the recommendation effectiveness across diverse application scenarios of service-oriented mashup creation, as developers often need to explore various granularity of mashup service by integrating different number of API services.

Precision@N and Recall@N are single-value metrics that evaluate the fundamental effectiveness of Web API recommendation. Precision@N measures the ratio of actually used APIs among the top N recommendations, helping assess how accurately the service recommendation model identifies APIs that match mashup developers' requirements:

$$\text{Precision@N} = \frac{|realAPIs \cap topNAPIs|}{|topNAPIs|} \tag{16}$$

where $|realAPIs|$ represents the number of actual used APIs, and $|topNAPIs|$ is the number of services in the top N recommendations.

Recall@N represents the ratio of hit APIs to all actually used APIs, indicating the service recommendation model's capability to surface relevant APIs from the service ecosystem:

$$\text{Recall@N} = \frac{|realAPIs \cap topNAPIs|}{|realAPIs|} \tag{17}$$

During the API recommendation, the order of recommended APIs significantly impacts mashup developers' efficiency in API selection. NDCG@N addresses this by considering position-based importance, ensuring that the most suitable APIs are prioritized in the recommended API list:

$$\text{NDCG@N} = \frac{\text{DCG@N}}{\text{IDCG@N}} \tag{18}$$

where DCG@N assesses the importance and position of each item in the recommendation list, measured by a weighted sum:

$$\text{DCG@N} = \sum_{i=1}^{N} \frac{rel_i}{\log_2(i+1)} \tag{19}$$

IDCG@N represents the maximum cumulative gain in an ideal scenario, used to normalize DCG@N.

MAP@N provides a comprehensive assessment of the API recommendation performance across different mashup creation scenarios, calculated as:

$$\text{MAP@N} = \frac{1}{|M|} \sum_{m \in M} AP_m@N \tag{20}$$

where $AP_m@N$ represents the average precision of all hit positions in the recommendation list for mashup m:

$$\text{AP@N} = \frac{\sum_{i=1}^{N} rel_i * \text{Precision@i}}{\sum_{i=1}^{N} rel_i} \tag{21}$$

5.3. Baselines

To evaluate the performance of SRCA, we selected a series of representative baseline approaches spanning different technical approaches and development periods. These baselines include both classical approaches that have proven their effectiveness over time and state-of-the-art approaches published in recent years. Most of these approaches have publicly available implementations, facilitating reproducibility. For fair comparison, we processed our dataset into the required input formats for each baseline approach while maintaining consistent evaluation conditions.

- **CF** (Cremonesi et al., 2012): This classic collaborative filtering approach recommends APIs by analyzing the API usage patterns of similar mashups. Specifically, it generates recommendations for a target mashup based on the APIs used by its most similar mashups.
- **NCF** (He, Liao, et al., 2017): Extending traditional collaborative filtering with neural networks, this approach learns separate feature matrices for mashups and APIs through matrix factorization. It combines these features using multi-layer perceptrons to predict mashup-API relevance scores, with service features initialized using Glove.6B.300d pre-trained word embeddings.
- **LSTM** (Nowak et al., 2017): Built upon bidirectional LSTM with attention mechanism, this model captures semantic features from service descriptions to enable API recommendation. It processes both mashup and API descriptions using Glove.6B.300d embeddings as input, ranking APIs based on their cosine similarities with the target mashup's requirements.
- **SPR** (Zhong et al., 2018): Through topic modeling analysis, this approach discovers latent topics from service descriptions using the author-topic model. It involves analyzing topic distributions of both mashup and API descriptions, identifying the desired APIs whose topical patterns align with the target mashup's requirements.
- **RWR** (Wang et al., 2019): Representing service relationships in a knowledge graph structure, this approach encodes mashup contexts using tag and category entities. The recommendation strategy employs random walk with restart algorithm to traverse the graph, determining API candidates based on their structural proximity to the target mashup's requirements.
- **FC-LSTM** (Shi et al., 2019): This approach integrates dual attention mechanisms on functionality and context into an LSTM architecture for precise service feature extraction. It enhances service descriptions through topic modeling, while utilizing pre-trained Glove.6B.300d word embeddings to capture semantic relationships in the recommendation.
- **MTFM** (Wu et al., 2022): This approach integrates multi-model fusion and multi-task learning for API recommendation. It combines a semantic component for requirement understanding with feature interaction modeling, while leveraging both content-based and collaborative filtering advantages through feature fusion and mashup category prediction.
- **FSFM** (Wang, Xi, & Yin, 2023): This approach addresses API recommendation through a dual-component architecture: a structural interaction component encoding relationships in mashup-API heterogeneous networks, and a functional semantic component capturing multi-level requirement semantics. The fusion of these components particularly enhances long-tail Web API recommendation.
- **SEHGN** (Wang et al., 2024): This approach enhances heterogeneous graph networks with a multi-semantic aggregator for feature association modeling. It combines global-local semantic embeddings with graph-based feature learning, specifically addressing API compatibility dependencies and sparse invocation scenarios.

5.4. Implementation details

We implemented SRCA using Python 3.11 with PyTorch 2.5 and PyTorch Geometric 2.6 for graph neural network operations. All experiments 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. For competing approaches, we maintained their original implementations and hyperparameter settings as reported in their respective papers to ensure fair comparison.

In the semantic representation component, we leveraged LLMs due to their sophisticated language understanding and generation capabilities. Specifically, we selected the LLaMA series models, which offer state-of-the-art performance while maintaining an open-source architecture that enables flexible task adaptation. Compared to traditional BERT-like models, LLaMA's decoder-based architecture better supports both understanding and structured content generation, while its open nature provides advantages over closed-source alternatives (e.g., GPT-4,[2] Claude-3.5[3]) in terms of data privacy and customization. Considering the trade-offs between recommendation effectiveness and computational resources, we experimented with three variants of the latest LLaMA-3 series (1B, 3B, and 8B parameters)[4] to evaluate the impact of model scale on recommendation performance. For inference, we set the temperature to 0 and used greedy decoding to ensure deterministic outputs. The extracted semantic representations were projected into 1024-dimensional vectors through a linear transformation layer.

For the category co-occurrence feature augmentation component, we constructed a two-layer graph neural network. The edge weight threshold θ was empirically set to 0.2 to balance graph connectivity and information quality. Each graph convolution layer was followed by layer normalization and ELU activation. The dimension of the hidden layers was maintained at 1024 to match the semantic representations.

[2] GPT-4: https://openai.com/gpt-4.

[3] Claude-3.5: https://www.anthropic.com/claude.

[4] We used the official LLaMA models from Meta AI: https://ai.meta.com/llama/.

Table 2
Performance comparison of different approaches when N ranges from 5 to 20.

Approaches	N=5				N=10			
	Precision	Recall	MAP	NDCG	Precision	Recall	MAP	NDCG
CF	0.1218	0.3559	0.2349	0.3006	0.0972	0.5651	0.2576	0.3695
NCF	0.1350	0.3871	0.3783	0.4191	0.0860	0.4859	0.3859	0.4500
LSTM	0.1360	0.3932	0.3978	0.4346	0.0874	0.4962	0.4093	0.4692
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
FC-LSTM	0.1654	0.5615	0.4811	0.5282	0.0930	0.6386	0.4987	0.5608
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
SRCA (LLaMA-3.2-1B-Instruct)	0.2476	0.7501	0.6953	0.7349	0.1342	0.7851	0.7052	0.7456
SRCA (LLaMA-3.2-3B-Instruct)	0.2567	0.7698	0.7205	0.7523	0.1354	0.8056	0.7311	0.7733
SRCA (LLaMA-3.1-8B-Instruct)	**0.2623**	**0.7837**	**0.7421**	**0.7788**	**0.1420**	**0.8216**	**0.7533**	**0.7904**
Gains	13.89%	5.01%	16.96%	12.48%	9.31%	2.75%	17.76%	11.40%

Approaches	N=15				N=20			
	Precision	Recall	MAP	NDCG	Precision	Recall	MAP	NDCG
CF	0.0699	0.6039	0.2576	0.3777	0.0551	0.6299	0.2557	0.3814
NCF	0.0624	0.5346	0.3867	0.4606	0.0496	0.5638	0.3824	0.4642
LSTM	0.0637	0.5522	0.4083	0.4802	0.0508	0.5832	0.4031	0.4836
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
FC-LSTM	0.0737	0.6566	0.5058	0.5674	0.0569	0.6576	0.5125	0.5750
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
SRCA (LLaMA-3.2-1B-Instruct)	0.0933	0.8073	0.7083	0.7525	0.0717	0.8199	0.7099	0.7564
SRCA (LLaMA-3.2-3B-Instruct)	0.0957	0.8299	0.7341	0.7707	0.0736	0.8376	0.7340	0.7778
SRCA (LLaMA-3.1-8B-Instruct)	**0.0982**	**0.8472**	**0.7564**	**0.7980**	**0.0751**	**0.8555**	**0.7576**	**0.8007**
Gains	8.03%	3.28%	16.96%	11.50%	6.52%	2.91%	16.99%	11.38%

The Web API recommendation component utilized a two-layer MLP with ReLU activation. The hidden layer dimension was set to 512, with dropout ($p = 0.3$) applied between layers to prevent overfitting. The output layer dimension matched the number of candidate APIs, with sigmoid activation for calculating recommendation scores.

For model training, we employed the Adam optimizer with a learning rate of 0.0001 and weight decay of 0.0005 for L2 regularization. To address the class imbalance in API usage patterns, we adopted Focal Loss with parameters $\alpha = 0.25$ and $\gamma = 2.0$. The model was trained using mini-batches of size 64, with early stopping (patience=10) monitoring the validation loss to prevent overfitting.

5.5. Comparison of recommendation performance

Table 2 presents the experimental results of various competing approaches under different numbers of recommended Web APIs (N). The optimal and second-best results are highlighted with deep and light gray backgrounds respectively. Additionally, the "Gains" row demonstrates SRCA's percentage improvement over the best baseline results.

Among all compared baseline approaches, traditional approaches based on collaborative filtering (CF, NCF) exhibit the poorest performance. While these approaches aim to leverage historical mashup patterns, they heavily rely on the unification and completeness of mashup descriptions. The inherent challenges of vocabulary inconsistency and information noise in service descriptions, as analyzed in existing limitations, significantly impact their ability to accurately identify functionally similar mashups, leading to unsatisfactory performance of Web API recommendation.

Content-based approaches (SPR, RWR) achieve better results by leveraging multi-dimensional service features. SPR employs topic modeling to reconstruct service profiles, effectively addressing vocabulary differences and enhancing description quality. RWR introduces a knowledge graph to encode service contextual information and utilizes random walk strategies to mine similar neighbors. These approaches show notable improvements over CF-based approaches,demonstrating the importance of semantic understanding in API recommendation.

Deep learning approaches (LSTM, FC-LSTM) further advance recommendation performance through sophisticated feature extraction. While basic LSTM shows moderate improvements, FC-LSTM significantly enhances performance by combining topic modeling for description extension with functional and contextual attention mechanisms. This architecture effectively captures both semantic and functional relationships, highlighting the importance of advanced feature extraction techniques.

Recent hybrid approaches (MTFM, FSFM, SEHGN) represent the current state-of-the-art in API recommendation. MTFM combines semantic and interaction features while leveraging auxiliary category classification tasks. FSFM advances this approach by incorporating graph neural networks to model higher-order relationships. SEHGN further enhances the hybrid framework through multi-semantic aggregation and semantic embedding techniques, achieving strong performance in API recommendation tasks.

Our proposed SRCA demonstrates consistent superiority across all metrics and settings. At N = 5, SRCA (8B) achieves significant improvements over SEHGN: 13.89% in Precision, 5.01% in Recall, 16.96% in MAP, and 12.48% in NDCG. These substantial performance gains can be attributed to the synergistic effect of our two innovative components. The LLM-enhanced description

Table 3
Performance comparison of ablation approaches when N ranges from 5 to 20.

	N = 5				N = 10			
Approaches	Precision	Recall	MAP	NDCG	Precision	Recall	MAP	NDCG
SRCA w/o SR	0.2411	0.7312	0.6773	0.7180	0.1318	0.7739	0.6894	0.7314
SRCA w/o CA	0.2451	0.7501	0.6918	0.7366	0.1387	0.7955	0.7050	0.7513
SRCA	**0.2623**	**0.7837**	**0.7421**	**0.7788**	**0.1420**	**0.8216**	**0.7533**	**0.7904**
	N = 15				N = 20			
Approaches	Precision	Recall	MAP	NDCG	Precision	Recall	MAP	NDCG
SRCA w/o SR	0.0921	0.7975	0.6932	0.7391	0.0710	0.8107	0.6947	0.7431
SRCA w/o CA	0.0958	0.8175	0.7188	0.7587	0.0735	0.8342	0.7106	0.7636
SRCA	**0.0982**	**0.8472**	**0.7564**	**0.7980**	**0.0751**	**0.8555**	**0.7576**	**0.8007**

unification and semantic representation effectively addresses the semantic inconsistency and information noise through sophisticated language understanding capabilities, while the category co-occurrence feature augmentation further enhances service features by capturing both explicit and implicit functional relationships between mashups and APIs. Therefore, it overcomes key limitations of existing approaches that either struggle with semantic processing or underutilize categorical contexts.

To further understand the impact of model scale on recommendation performance, we conducted experiments with different parameter sizes. The progression from 1B to 3B parameters brings substantial improvements (3%–4% across metrics), particularly in MAP and Precision at smaller N values, indicating enhanced ability to identify highly relevant APIs. Further improvements are observed with the 8B parameter model (2%–3% additional gains), demonstrating how increased model capacity enhances recommendation performance. These improvements can be attributed to larger models' sophisticated capabilities in semantic processing: they demonstrate superior understanding of service descriptions through richer contextual comprehension, more effectively unify diverse description styles while preserving essential functional information, and excel at capturing subtle semantic relationships between mashup requirements and API capabilities. Notably, SRCA demonstrates robust performance across different parameter scales, with even the 1B version delivering substantial improvements over traditional approaches, highlighting its adaptability to various computational resources while maintaining competitive performance.

5.6. Ablation study

To thoroughly evaluate the contribution of each key component in SRCA, we conduct ablation studies by creating two variants: SRCA w/o SR which replaces the LLM-enhanced service semantic representation with BERT embeddings (specifically using the [CLS] token representation from BERT-large-uncased, with 24 layers and 1024 hidden dimensions), and SRCA w/o CA which omits the category co-occurrence feature augmentation. Table 3 presents their performance comparisons across different size in recommendation list (N = 5 to 20).

The experimental results demonstrate that both components contribute significantly to the model's overall performance, though their impacts vary in magnitude and nature. When examining the LLM-enhanced service semantic representation module (comparing SRCA with SRCA w/o SR), we observe substantial performance degradation across all metrics, with MAP decreasing by 8.73% and NDCG dropping by 7.81% at N = 5. This phenomenon highlights the advantages of our LLM-based approach over traditional semantic representation models. Beyond providing basic text embeddings, LLMs offer sophisticated language understanding capabilities and can be effectively adapted to specific tasks through carefully designed prompts. Our requirement-focused and functional-oriented prompts enable the model to better unify service descriptions, while the rich internal knowledge and cross-domain understanding capabilities of LLMs contribute to more accurate semantic representations of service functionalities.

The impact of the category co-occurrence feature augmentation module is revealed by comparing SRCA with SRCA w/o CA. Its removal leads to noticeable performance drops, with MAP decreasing by 6.78% and NDCG dropping by 5.42% at N = 5. This decline demonstrates the essential role of category-based contextual information in Web API recommendation. While service semantic representation effectively captures individual service functionalities, it lacks the ability to model relationships between functionally similar services, especially when mashup requirements are incomplete. Our category co-occurrence graph bridges this gap by modeling both explicit categorical co-occurrence service similarities and implicit functional correlations, discovering complex service relationships that might not be apparent purely from semantic features.

The comparative analysis reveals that SRCA w/o CA consistently outperforms SRCA w/o SR across all metrics and N values, highlighting the fundamental role of semantic understanding in service recommendation. This observation aligns with the nature of the recommendation task: accurate comprehension of individual service functionalities serves as the essential foundation, while category-based relationships provide further enhancement for API recommendation. The semantic representation module directly addresses the core challenge of understanding service capabilities, enabling precise matching between mashup requirements and API descriptions. In contrast, the category-based feature augmentation enhances the service semantic representation by incorporating broader service relationships, explaining why the degradation in semantic understanding leads to more severe performance drops.

The superior performance of the full SRCA model over both variants demonstrates effective synergy between service semantic features and category-based feature augmentation. It is particularly pronounced at smaller N values (N = 5,10), which directly

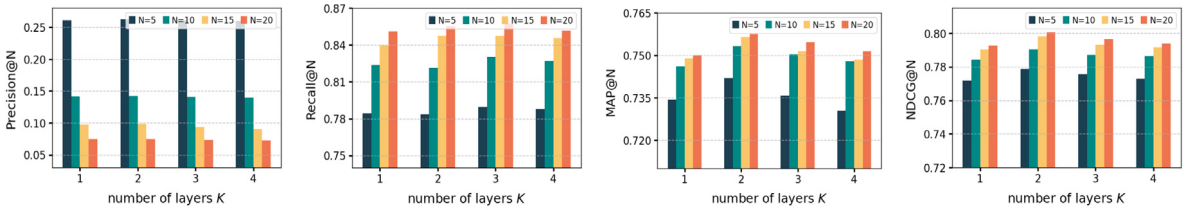

Fig. 6. Performance impact of the GNN layers (K).

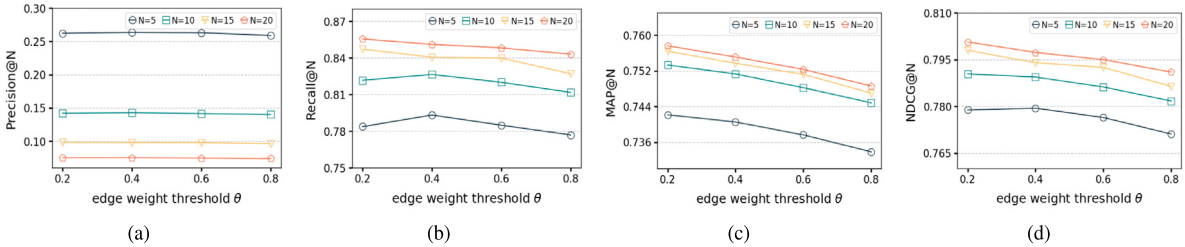

Fig. 7. Performance impact of the edge weight threshold (θ).

aligns with practical mashup creation scenarios where developers typically utilize only 2–5 APIs. The results suggest that our two key components effectively addresses both the semantic understanding of individual services and their contextual relationships for further feature augmentation, leading to more accurate and practical Web API recommendation.

5.7. Hyperparameter study

We conduct extensive experiments to investigate the impact of two key hyperparameters in SRCA: the number of graph neural network layers K and the edge weight threshold θ in the category co-occurrence feature augmentation module. For each parameter study, all other parameters are maintained at their optimal settings.

5.7.1. Impact of GNN layers

In the category co-occurrence feature augmentation module, the number of GNN layers K plays a decisive role in determining how information propagates through the graph structure. With fewer layers, the model primarily captures direct categorical relationships between adjacent nodes, while additional layers enable the aggregation of more complex, higher-order categorical patterns across the service ecosystem.

To investigate the impact of layer depth on Web API recommendation performance, we conducted experiments with K ranging from 1 to 4, as shown in Fig. 6. The results demonstrate that a 2-layer architecture achieves optimal performance across most evaluation metrics, with higher layer counts leading to diminishing returns or even performance degradation while incurring increased computational costs.

The performance improvement from $K = 1$ to $K = 2$ can be attributed to the enhanced information flow pattern: with a single layer, mashups only aggregate category information from directly connected API nodes, capturing basic functional similarities. The addition of a second layer allows the model to consider information from other related mashup nodes, helping to reveal indirect relationships and complex demand patterns that are particularly valuable for API recommendation. For instance, when two mashups share similar API usage patterns but have different category labels, the 2-layer architecture can effectively capture these implicit functional similarities through intermediate nodes.

However, as the number of layers increases beyond two, we observe a consistent decline in performance, particularly in precision and ranking metrics (MAP@N and NDCG@N). This degradation can be explained by several factors: First, the exponential growth in the number of reachable nodes leads to the inclusion of many weakly related services, introducing noise that may overshadow the more important direct relationships. Second, excessive information propagation can result in over-smoothing, where node representations become increasingly similar, reducing the model's ability to distinguish between different service categories.

Based on these observations and considering both performance and computational efficiency, we adopt $K = 2$ as the optimal setting for SRCA. This choice aligns well with the practical requirements of API recommendation, where capturing both direct functional similarities and indirect categorical relationships is crucial for generating accurate and relevant recommendations.

5.7.2. Impact of edge weight threshold

The edge weight threshold θ plays a crucial role in constructing the category co-occurrence graph, directly affecting both graph structure and feature propagation patterns. This parameter essentially controls the granularity of service relationship modeling: it determines which category co-occurrences are considered significant enough to establish connections in the graph, thereby influencing how service relationships are captured and how functional information propagates through the network.

Fig. 7 illustrates the model's performance as θ varies from 0.2 to 0.8. The results demonstrate overall superior performance when θ falls within the range of 0.2–0.4, with the optimal results achieved at $\theta = 0.2$. This aligns with our observation that the category similarity values between service nodes predominantly fall within this range. A moderate threshold of 0.2 creates a well-connected graph that effectively balances structural and functional requirements: it preserves sufficient categorical relationships to capture both direct similarities and implicit correlations through intermediate nodes, while still filtering out noise from weak connections. This connectivity pattern enables effective high-order feature aggregation, allowing the graph neural network to discover both explicit and subtle service relationships that are particularly valuable for identifying compatible and complementary services in mashup creation.

As the threshold increases beyond 0.4, we observe consistent performance degradation across all metrics, which can be attributed to several structural and computational factors. Higher thresholds lead to excessive edge pruning, creating an overly sparse graph structure where many meaningful moderate-strength relationships are eliminated. This sparsity has cascading effects on feature propagation: isolated nodes or weakly connected components emerge, limiting the model's ability to aggregate implicit service features across different service communities. Moreover, the reduced connectivity disrupts the discovery of indirect relationships, which are crucial for identifying complementary services that might be connected through intermediate categories.

Considering both the empirical results and theoretical analysis, we adopt $\theta = 0.2$ as the default setting for SRCA. This choice achieves an optimal balance: it maintains sufficient graph connectivity for effective feature propagation while filtering out noise, enables the discovery of both direct and indirect service relationships, and preserves the graph's ability to model complex service interactions. The resulting categorical co-occurrence graph supports comprehensive feature aggregation across multiple orders, leading to more accurate and diverse Web API recommendation for effective mashup creation.

6. Discussion

6.1. Theoretical implications

This study provides two key theoretical implications for the field of Web API recommendation.

The first theoretical implication lies in SRCA's ability to construct a unified and structured expression framework for service descriptions. Previous approaches have primarily relied on generic neural encoders such as CNNs or BERT to process raw service texts, which often suffer from semantic inconsistency and irrelevant information noise. These limitations hinder the extraction of accurate semantic features, especially when dealing with diverse writing styles and unstructured content. In contrast, SRCA employs requirement-focused and functional-oriented prompts to guide LLMs in reorganizing mashup and API descriptions into parallel, structured representations. This organization facilitates a clearer semantic correspondence between mashup requirements and API capabilities. We argue that such structural alignment, combined with the deep semantic understanding capabilities of LLMs, enables the extraction of more accurate, consistent, and functionally focused semantic representations. This hypothesis is supported by ablation studies, which show that replacing the LLM-enhanced service semantic representation with traditional text encoders significantly reduces recommendation accuracy.

The second theoretical implication stems from SRCA's enhancement of service representations through category-based structural modeling. While earlier methods have attempted to incorporate category information, they often underutilize its potential to reveal functional associations among services. SRCA addresses this gap by constructing a mashup-API category co-occurrence graph that captures both explicit categorical similarities and implicit functional correlations. Moreover, SRCA introduces a high-order service feature aggregation mechanism to propagate and integrate multi-hop relational knowledge within the graph. Unlike prior works that treat semantic and structural features separately, SRCA integrates these high-order relational features into the learned service representations through a feature augmentation process. This design allows the structural context derived from category co-occurrence patterns to enhance the semantic representations, leading to more comprehensive modeling of service functionality. The effectiveness of this integration is validated through performance gains observed in experiments, particularly in scenarios involving incomplete service descriptions.

6.2. Practical implications

The SRCA framework provides practical value by addressing key challenges encountered in real-world Web API recommendation tasks, particularly in mashup development scenarios and deployment environments with constrained resources.

One of the central difficulties in mashup development is the increasing complexity of API selection. As the number of available Web APIs continues to grow and functional redundancy becomes more common, developers often face increasing difficulty in accurately identifying APIs that best match their specific requirements, as many services offer similar functionalities with only minor differences in focus or usage scenarios. Manually identifying the most suitable API combinations requires extensive effort and carries a high risk of overlooking relevant services. SRCA mitigates this problem by first transforming diverse and noisy service descriptions into structured, functionally focused representations using prompt-based unification. It then combines deep semantic

understanding from LLMs with category-based structural enhancement to generate precise, requirement-aligned recommendations. This process reduces the manual burden of service selection, enabling developers to concentrate on functional design and adapt more rapidly to dynamic application needs.

In addition to supporting recommendation accuracy, SRCA is also designed to accommodate practical deployment requirements in environments where computational resources are limited or data privacy is a concern. While larger LLMs generally achieve better performance, our experimental results indicate that SRCA maintains stable effectiveness even when integrated with smaller-scale models. This allows the framework to be deployed in on-premise or self-hosted settings, without the need to transmit sensitive service descriptions to external servers. Such deployment flexibility is particularly beneficial for organizations operating under strict privacy constraints or working with restricted hardware environments, further expanding the applicability of SRCA across a wide range of service recommendation scenarios.

6.3. Limitations

While SRCA achieves strong performance and demonstrates practical applicability, certain limitations should be acknowledged.

One concern involves the dependency on LLMs within the semantic representation process. Although smaller LLMs remain effective, larger models consistently yield better results, creating a trade-off between computational cost and recommendation accuracy. In addition, when service descriptions are overly brief or underspecified, the effectiveness of prompt-based unification may be constrained. This increases the risk of generating hallucinated content, which can affect the reliability of the resulting semantic features.

Another limitation arises from the reliance on category information to construct the co-occurrence graph used for structural enhancement. Incomplete or inaccurate labels can lead to missing connections or distorted graph structures, thereby weakening the representation learning process. Additionally, when the co-occurrence graph becomes densely connected, high-order feature aggregation may introduce irrelevant or misleading information into the service representations.

7. Conclusion and future work

In this paper, we propose SRCA, a novel Web API recommendation framework that effectively addresses the challenges of semantic inconsistency and insufficient relationship exploitation in service recommendation. Our framework synergistically combines two key innovations: an LLM-enhanced semantic representation approach that unifies diverse service descriptions through specially designed prompts while extracting rich semantic features, and an innovative category co-occurrence feature augmentation mechanism that captures both direct and higher-order relationships through sophisticated graph-based feature propagation. Extensive experiments on real-world datasets from ProgrammableWeb demonstrate that SRCA substantially outperforms state-of-the-art baselines across multiple evaluation metrics, validating the effectiveness of our unified approach.

For future work, we plan to incorporate real-time user feedback mechanisms into the Web API recommendation, enabling dynamic adaptation to various practical scenarios in mashup creation, such as service compatibility issues, regional availability constraints, and the need for alternative services. By leveraging LLM's semantic understanding capabilities to process user feedback and preferences, we can develop more personalized and context-aware recommendation strategies that better align with developers' specific requirements while maintaining robustness in evolving service environments.

CRediT authorship contribution statement

Guobing Zou: Conceptualization, Funding acquisition, Methodology, Project administration, Writing – review & editing. **Pengtao Li:** Methodology, Software, Writing – original draft, Writing – review & editing. **Song Yang:** Investigation, Methodology, Validation. **Shengxiang Hu:** Investigation, Validation. **Shengye Pang:** Supervision. **Yanglan Gan:** Funding acquisition, Supervision.

Declaration of competing interest

The authors declare that they have no known competing financial interests or personal relationships that could have appeared to influence the work reported in this paper.

Acknowledgments

This work was supported by National Natural Science Foundation of China (No. 62272290, 62172088) and Shanghai Natural Science Foundation (No. 21ZR1400400).

Data availability

Data will be made available on request.

References

Cao, B., Liu, X. F., Rahman, M. M., Li, B., Liu, J., & Tang, M. (2020). Integrated content and network-based service clustering and web APIs recommendation for mashup development. *IEEE Transactions on Services Computing*, *13*(1), 99–113.

Cao, B., Peng, M., Zhang, L., Qing, Y., Tang, B., Kang, G., & Liu, J. (2023). Web service recommendation via integrating heterogeneous graph attention network representation and FiBiNET score prediction. *IEEE Transactions on Services Computing*, *16*(5), 3837–3850.

Cao, B., Tang, M., & Huang, X. (2014). CSCF: A mashup service recommendation approach based on content similarity and collaborative filtering. *International Journal of Grid & Distributed Computing*, *7*(2), 163–172.

Chen, X., Zheng, Z., & Lyu, M. R. (2014). QoS-aware web service recommendation via collaborative filtering. In *Web services foundations* (pp. 563–588). Springer.

Clevert, D., Unterthiner, T., & Hochreiter, S. (2016). Fast and accurate deep network learning by exponential linear units (ELUs). In *International conference on learning representations*.

Cremonesi, P., Picozzi, M., & Matera, M. (2012). A comparison of recommender systems for mashup composition. In *Proceedings of the 3rd international workshop on recommendation systems for software engineering* (pp. 54–58). IEEE.

Devlin, J., Chang, M., Lee, K., & Toutanova, K. (2019). BERT: Pre-training of deep bidirectional transformers for language understanding. In *Proceedings of the 2019 conference of the North American chapter of the association for computational linguistics: human language technologies* (pp. 4171–4186). Association for Computational Linguistics.

Dong, Q., Li, L., Dai, D., Zheng, C., Ma, J., Li, R., Xia, H., Xu, J., Wu, Z., Chang, B., Sun, X., & Sui, Z. (2024). A survey on in-context learning. In *Proceedings of the 2024 conference on empirical methods in natural language processing* (pp. 1107–1128). Association for Computational Linguistics.

Geva, M., Schuster, R., Berant, J., & Levy, O. (2021). Transformer feed-forward layers are key-value memories. In *Proceedings of the 2021 conference on empirical methods in natural language processing* (pp. 5484–5495). Association for Computational Linguistics.

Grattafiori, A., Dubey, A., Jauhri, A., Pandey, A., Kadian, A., Al-Dahle, A., Letman, A., Mathur, A., Schelten, A., Vaughan, A., Yang, A., Fan, A., Goyal, A., Hartshorn, A., Yang, A., Mitra, A., Sravankumar, A., Korenev, A., Hinsvark, A., et al. (2024). The llama 3 herd of models. arXiv preprint arXiv:2407.21783.

Gu, Q., Cao, J., & Liu, Y. (2022). CSBR: A compositional semantics-based service bundle recommendation approach for mashup development. *IEEE Transactions on Services Computing*, *15*(6), 3170–3183.

He, X., Liao, L., Zhang, H., Nie, L., Hu, X., & Chua, T. (2017). Neural collaborative filtering. In *Proceedings of the 26th international conference on world wide web* (pp. 173–182). International World Wide Web Conferences Steering Committee.

He, Q., Zhou, R., Zhang, X., Wang, Y., Ye, D., Chen, F., Chen, S., Grundy, J., & Yang, Y. (2017). Efficient keyword search for building service-based systems based on dynamic programming. In *Proceedings of the 15th international conference on service-oriented computing: Vol. 10601*, (pp. 462–470). Springer.

He, Q., Zhou, R., Zhang, X., Wang, Y., Ye, D., Chen, F., Grundy, J. C., & Yang, Y. (2017). Keyword search for building service-based systems. *IEEE Transactions on Software Engineering*, *43*(7), 658–674.

Hu, X. (2024). Biased random walk based web API recommendation in heterogeneous network. In *Proceedings of the 2024 IEEE international conference on web services* (pp. 172–177). IEEE.

Imran, M., Soi, S., Kling, F., Daniel, F., Casati, F., & Marchese, M. (2012). On the systematic development of domain-specific mashup tools for end users. In *Proceedings of the 12th international conference on web engineering: Vol. 7387*, (pp. 291–298). Springer.

Kang, G., Liang, B., Liu, J., Wen, Y., Xiao, Y., & Nie, H. (2024). Web API recommendation via leveraging content and network semantics. *IEEE Transactions on Network and Service Management*, *21*(6), 5977–5991.

Kang, G., Wang, Y., Ren, H., Cao, B., Liu, J., & Wen, Y. (2024). KS-GNN: Keyword search via graph neural network for web API recommendation. *IEEE Transactions on Network and Service Management*, *21*(5), 5464–5474.

Kim, Y. (2014). Convolutional neural networks for sentence classification. In *Proceedings of the 2014 conference on empirical methods in natural language processing* (pp. 1746–1751). Association for Computational Linguistics.

Kingma, D. P., & Ba, J. (2015). Adam: A method for stochastic optimization. In *International conference on learning representations*.

Li, H., Liu, J., Cao, B., Tang, M., Liu, X. F., & Li, B. (2017). Integrating tag, topic, co-occurrence, and popularity to recommend web APIs for mashup creation. In *Proceedings of the 2017 IEEE international conference on services computing* (pp. 84–91). IEEE.

Lin, T.-Y., Goyal, P., Girshick, R. B., He, K., & Dollár, P. (2017). Focal loss for dense object detection. In *Proceedings of the 2017 IEEE international conference on computer vision* (pp. 2980–2988).

Liu, M., Tu, Z., Xu, H., Xu, X., & Wang, Z. (2023). DySR: A dynamic graph neural network based service bundle recommendation model for mashup creation. *IEEE Transactions on Services Computing*, *16*(4), 2592–2605.

Lizarralde, I., Mateos, C., Zunino, A., Majchrzak, T. A., & Grønli, T. (2020). Discovering web services in social web service repositories using deep variational autoencoders. *Information Processing & Management*, *57*(4), Article 102231.

Nowak, J., Taspinar, A., & Scherer, R. (2017). LSTM recurrent neural networks for short text and sentiment classification. In *Proceedings of the 16th international conference on artificial intelligence and soft computing* (pp. 553–562). Springer.

OpenAI, Achiam, J., Adler, S., Agarwal, S., Ahmad, L., Akkaya, I., Aleman, F. L., Almeida, D., Altenschmidt, J., Altman, S., Anadkat, S., Avila, R., Babuschkin, I., Balaji, S., Balcom, V., Baltescu, P., Bao, H., Bavarian, M., Belgum, J., et al. (2024). GPT-4 technical report. arXiv preprint arXiv:2303.08774.

Pan, G., Chang, Y., Qi, H., & Hu, Q. (2023). Web service category recommendation with feature word semantic enhancement and tag co-occurrence. In *Proceedings of the 2023 international conference on networking and network applications* (pp. 686–689). IEEE.

Rahman, M. M., & Liu, X. F. (2020). Integrated topic modeling and user interaction enhanced WebAPI recommendation using regularized matrix factorization for mashup application development. In *Proceedings of the 2020 IEEE international conference on services computing* (pp. 124–131). IEEE.

Rahman, M. M., Liu, X. F., & Cao, B. (2017). Web API recommendation for mashup development using matrix factorization on integrated content and network-based service clustering. In *Proceedings of the 2017 IEEE international conference on services computing* (pp. 225–232). IEEE.

Ren, L., & Wang, W. (2022). A granular SVM-based method for top-N web services recommendation. *IEEE Transactions on Services Computing*, *15*(1), 457–469.

Rogers, A., Kovaleva, O., & Rumshisky, A. (2020). A primer in BERTology: What we know about how BERT works. *Transactions of the Association for Computational Linguistics*, *8*, 842–866.

Rubin, O., Herzig, J., & Berant, J. (2022). Learning to retrieve prompts for in-context learning. In *Proceedings of the 2022 conference of the North American chapter of the association for computational linguistics: Human language technologies* (pp. 2655–2671). Association for Computational Linguistics.

Sang, C., Deng, X., & Liao, S. (2023). Mashup-oriented web API recommendation via full-text semantic mining of developer requirements. *IEEE Transactions on Services Computing*, *16*(4), 2755–2768.

Schulhoff, S., Ilie, M., Balepur, N., Kahadze, K., Liu, A., Si, C., Li, Y., Gupta, A., Han, H., Schulhoff, S., Dulepet, P. S., Vidyadhara, S., Ki, D., Agrawal, S., Pham, C., Kroiz, G., Li, F., Tao, H., Srivastava, A., et al. (2025). The prompt report: A systematic survey of prompt engineering techniques. arXiv preprint arXiv:2406.06608.

Shi, M., Tang, Y., & Liu, J. (2019). Functional and contextual attention-based LSTM for service recommendation in mashup creation. *IEEE Transactions on Parallel and Distributed Systems*, *30*(5), 1077–1090.

Sutskever, I., Vinyals, O., & Le, Q. V. (2014). Sequence to sequence learning with neural networks. In *Advances in neural information processing systems 27: Annual conference on neural information processing systems 2014: Vol. 27*, (pp. 3104–3112).

Ul Ain, N., Irfan, A., Iltaf, N., & Ul Hassan, M. (2022). A hybrid collaborative filtering technique for web service recommendation using contextual attributes of web services. In *2022 second international conference on distributed computing and high performance computing* (pp. 60–65). IEEE.

Wang, X., Wu, H., & Hsu, C. (2019). Mashup-oriented API recommendation via random walk on knowledge graph. *IEEE Access*, *7*, 7651–7662.

Wang, X., Xi, M., Li, Y., Pan, X., Wu, Y., Deng, S., & Yin, J. (2024). SEHGN: Semantic-enhanced heterogeneous graph network for web API recommendation. *IEEE Transactions on Services Computing*, *17*(5), 2836–2849.

Wang, X., Xi, M., & Yin, J. (2023). Functional and structural fusion based web API recommendations in heterogeneous networks. In *Proceedings of the 2023 IEEE international conference on web services* (pp. 91–96). IEEE.

Wang, Y., Xiang, J., Cheng, H., Chen, W., Xiao, Y., & Kang, G. (2023). Towards dynamic evolutionary analysis of ProgrammableWeb for API-mashup ecosystem. In *Proceedings of the 2023 26th international conference on computer supported cooperative work in design* (pp. 1716–1721). IEEE.

Wu, H., Duan, Y., Yue, K., & Zhang, L. (2022). Mashup-oriented web API recommendation via multi-model fusion and multi-task learning. *IEEE Transactions on Services Computing*, *15*(6), 3330–3343.

Wu, H., Yue, K., Li, B., Zhang, B., & Hsu, C.-H. (2018). Collaborative QoS prediction with context-sensitive matrix factorization. *Future Generation Computer Systems*, *82*, 669–678.

Xia, B., Fan, Y., Tan, W., Huang, K., Zhang, J., & Wu, C. (2015). Category-aware API clustering and distributed recommendation for automatic mashup creation. *IEEE Transactions on Services Computing*, *8*(5), 674–687.

Xiao, Y., Liu, J., Kang, G., Hu, R., Cao, B., Cao, Y., & Shi, M. (2020). Structure reinforcing and attribute weakening network based API recommendation approach for mashup creation. In *Proceedings of the 2020 IEEE international conference on web services* (pp. 541–548). IEEE.

Xiao, S., Liu, Z., Zhang, P., Muennighoff, N., Lian, D., & Nie, J. (2024). C-Pack: Packed resources for general Chinese embeddings. In *Proceedings of the 47th international ACM SIGIR conference on research and development in information retrieval* (pp. 641–649). ACM.

Yan, R., Fan, Y., Zhang, J., Zhang, J., & Lin, H. (2021). Service recommendation for composition creation based on collaborative attention convolutional network. In *Proceedings of the 2021 IEEE international conference on web services* (pp. 397–405). IEEE.

Yao, L., Sheng, Q. Z., Segev, A., & Yu, J. (2013). Recommending web services via combining collaborative filtering with content-based features. In *Proceedings of the 2013 IEEE 20th international conference on web services* (pp. 42–49). IEEE.

Yao, L., Wang, X., Sheng, Q. Z., Benatallah, B., & Huang, C. (2021). Mashup recommendation by regularizing matrix factorization with API co-invocations. *IEEE Transactions on Services Computing*, *14*(2), 502–515.

Yu, C., Hu, R., & Wang, B. (2023). AKGIN: An API knowledge graph and intent network based mashup-oriented API recommendation method. In *Proceedings of the 2023 26th international conference on computer supported cooperative work in design* (pp. 261–266). IEEE.

Zhang, M., & Zhou, Z. (2014). A review on multi-label learning algorithms. *IEEE Transactions on Knowledge and Data Engineering*, *26*(8), 1819–1837.

Zheng, Z., Ma, H., Lyu, M. R., & King, I. (2011). QoS-aware web service recommendation by collaborative filtering. *IEEE Transactions on Services Computing*, *4*(2), 140–152.

Zhong, Y., Fan, Y., Tan, W., & Zhang, J. (2018). Web service recommendation with reconstructed profile from mashup descriptions. *IEEE Transactions on Automation Science and Engineering*, *15*(2), 468–478.