Dynamic Relation Graph Learning for Time-Aware Service Recommendation

Chunyu Wei, Yushun Fan, Jia Zhang, *Senior Member, IEEE*, Zhixuan Jia, and Ruyu Yan

***Abstract*—Driven by Service-Oriented Computing, time-aware service recommendation aims to support personalized mashup development, adapting to the rapid shifts of users' dynamic preferences. Recently, users' social connections have shown significant benefits to time-aware service recommendation, and graph neural networks have demonstrated great success in learning the pattern of information flow among users. However, the current paradigm always presumes a given social network, which is not necessarily consistent with the similarities of service preferences among users and is expensive to collect for most service platforms. We propose a novel idea to learn the graph structure among historical mashups and make time-aware service recommendation for dynamic mashup creation collectively in a coupled framework. This idea raises two challenges, i.e., scalability and accuracy. To solve both challenges simultaneously, we introduce the Dynamic Relation Graph Learning (DRGL) framework for time-aware service recommendation. For scalability, our framework has a coarse-to-fine recalling strategy to learn the graph structure among the mashups, which enables the exploration of potential links among all historical mashups while maintaining a tractable amount of computation. For accuracy, we leverage recent advances in self-attention mechanisms to the mashup modeling and propose a transformer-based mashup encoder, which considers long-range dependencies in dense mashups for more accurate mashup representations. Extensive experiments show that the DRGL model consistently outperforms the state-of-the-art methods in terms of prediction accuracy for mashup creation.**

***Index Terms*—Service recommendation, graph neural networks, social network, time-aware, mashup creation.**

I. Introduction

OVER the past decade, the widespread use of Service-Oriented Architecture (SOA) has led to a rapid growth in the number of Web services available on the Internet [1]. These services can be combined as reusable components to create value-added mashups, i.e., a sequence of services working in proper order in a certain period. Mashup is a new technique of application development, which allows users to integrate existing Web services to create composite Web applications and react to complex business needs. Compared with the need to build everything from scratch, developers do not have to reinvent the wheel for each new application. Instead, they can focus on achieving the unique needs of the application, which greatly releases the developer's creativity. For example, Vizlingo, a widely-used mobile app, brings text messages and social media posts to life by animating each word with a video clip created by the user. To offer a complete social experience, it incorporates several APIs such as YouTube, Facebook, and Twitter. These APIs work together in a specific order to achieve the functionality of Vizlingo, better meeting the needs of users.

Typically, developers can search for and assess potentially valuable services in repositories and then use mashup tools to centrally incorporate the chosen services into their applications swiftly. However, the number of available Web APIs has grown rapidly in recent years. To help users select the optimal services from the vast array of candidates, service recommendation has emerged as a crucial instrument, using various filtering techniques, such as collaborative [2], content-based [3], or hybrid [4] filtering. However, these methods only consider users' inherent interests and service characteristics, without taking into account the time-dependent cooperation relationships between services within a mashup. Furthermore, users' preferences can change over time [5]. To address these limitations, *time-aware service recommendation* [1], [6], [7] focuses on a user's service invocation behaviors within a period of time and predicts the next service needed by the user to create a dynamic, personalized mashup.

To improve service recommendation with more useful information, researchers have proposed *social-aware service recommendation* [8], [9], which incorporates friends' service invocation behaviors to generate interest-adaptive services for target users. More recently, social information has also been integrated into time-aware service recommendation [10], [11], [12]. These methods typically utilize users' social relationships to construct a relation graph, allowing the propagation of users' dynamic preferences in social networks to be accurately modeled using advanced Graph Neural Networks (GNNs) [10], [11]. This enables the timely recommendation of the latest popular services to target users.

Existing graph-based methods for social-aware service recommendation often predefine a social network and utilize a static graph as input for their models [10]. However, these approaches have two limitations. First, it can be difficult for real-world service platforms to obtain users' social relations

Manuscript received 27 February 2023; revised 30 June 2023 and 6 October 2023; accepted 17 October 2023. Date of publication 19 October 2023; date of current version 15 April 2024. This research has been supported by the National Natural Science Foundation of China (No.62173199). The associate editor coordinating the review of this article and approving it for publication was D. Huang. *(Corresponding author: Yushun Fan.)*

Chunyu Wei, Yushun Fan, Zhixuan Jia, and Ruyu Yan are with the Beijing National Research Center for Information Science and Technology, Department of Automation, Tsinghua University, Beijing 100084, China (e-mail: cy-wei19@mails.tsinghua.edu.cn; fanyus@tsinghua.edu.cn; jzx21@tsinghua.edu.cn; yanry18@tsinghua.edu.cn).

Jia Zhang is with the Department of Computer Science, Southern Methodist University, Dallas, TX 75205 USA (e-mail: jiazhang@smu.edu).

Digital Object Identifier 10.1109/TNSM.2023.3325977

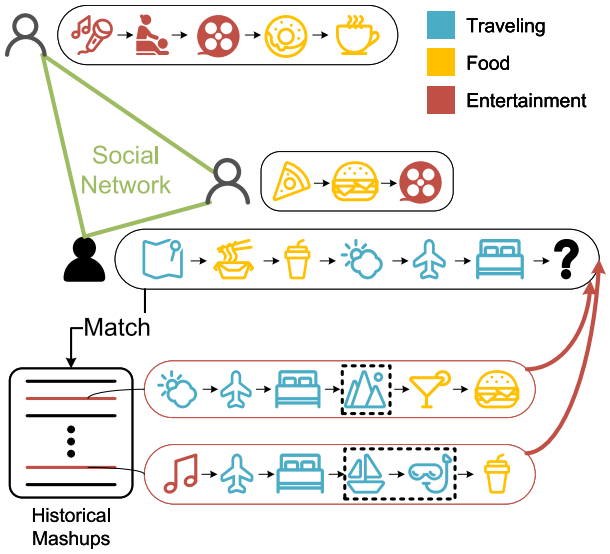

Fig. 1. The misalignment of social relations and service preference similarities. We propose to learn relations between the target mashup and the historical mashups to improve the time-aware service recommendation.

due to privacy restrictions, rendering the above graph-based methods unusable without a given graph. Second, the social relations among users may not align with similarities in service preferences. Figure 1 provides an example where the target user does not share similar service preferences with users in his social network. In such cases, the introduction of a social network may not only fail to help time-aware service recommendation, but also introduce noise to the learning process. Recent studies [13], [14] propose to learn the dynamic relationships among users using hand-crafted rules or parametric weights in the network. However, each user is involved with a large number of mashups over a long period of time, and existing approaches often compromise by characterizing a user's service preferences based on recently invoked mashups, potentially losing valuable information about past mashups.

We propose to collectively make time-aware service recommendations for dynamic mashup creation by learning the graph structure, i.e., the relations among mashups, as illustrated in Figure 1. However, this approach faces two notable challenges:

- *Scalability.* The number of historical mashups is much larger in magnitude compared to the number of users in the service ecosystem. Moreover, mining potential relations between two arbitrary mashups requires n^2-order calculation, where n is the overall number of mashups. This can easily reach tens of thousands, especially with an increase in the number of users and usage time. Hence, it is crucial to solve the scalability issue to mine potential relations among possible mashups.
- *Accuracy.* Current studies [10], [15] for time-aware service recommendation use sequential neural networks, such as Recurrent Neural Networks (RNNs), to learn the temporal dependencies among services in a mashup. However, RNN-based approaches are unsuitable for large-scale parallel mashup computations on graphs, as they suffer from time-consuming iterative propagation. Additionally, they often assume a rigidly ordered sequence over mashups, which is not always true for user behaviors in real-world applications. Moreover, gradient explosion/vanishing can occur when capturing the long-range dependency in the mashup.

To address these challenges synergistically, we propose a novel framework called Dynamic Relation Graph Learning (DRGL). DRGL is designed to make time-aware service recommendations for dynamic mashup creation.

To address the scalability issue, we propose a coarse-to-fine recalling strategy to learn the graph structure among mashups. The coarse recalling process utilizes a non-parametric memory bank to store the massive historical mashups as 0-1 vectors, where each feature (contained service) for each mashup is stored discretely. We formulate the mashup coarse recalling process as a metric learning problem, where distances (i.e., similarity) between mashups are calculated by counting the number of shared items (i.e., inner-products of two vectors). For the target mashups, we recall k_1 potential neighbors from all the historical mashups, enabling exploration of possible relations from the whole graph while maintaining tractable computation. The fine recalling process is the *self-adaptive graph structure learning*, which adaptively selects a smaller number of similar neighbors (i.e., k_2) from the coarse neighbor recalling results. This stage uses more information, including the temporal dependency and potential purpose contained in mashups, to calculate similarities among mashups through a finer-designed parametric network. The coarse-to-fine recalling strategy allows us to scale up and learn graph structures from massive historical mashups (i.e., n mashups), with the setting of $k_2 < k_1 \ll n$.

To address the accuracy issue, we propose a transformer-based mashup encoder that leverages recent advances in self-attention mechanisms for mashup modeling [16], [17]. The transformer-based mashup encoder adaptively assigns weights to each service at each time step, enabling consideration of long-range dependencies in dense mashups. With the self-attention mechanism, this encoder is much faster than RNN-based approaches, particularly in computing multiple mashups on a graph for relation learning. Moreover, it draws context from all services in the mashup, while extracting the essential services that contribute to the real purpose of the dynamic mashup.

This paper presents the following main contributions:

- We propose DRGL, a framework that learns the graph structure of historical mashups for time-aware service recommendation, providing scalability and controllable computation costs. This study is the first to learn the graph structure of historical mashups directly for time-aware service recommendation.
- We propose a transformer-based mashup encoder that accurately extracts the purpose of a mashup and accelerates parallel computation of multiple mashups when learning the graph structure.
- Our experiments on a real-life local service dataset show that DRGL outperforms state-of-the-art methods and proves the effectiveness of the learned graph structures.

The remainder of this article is organized as follows. Section II formally defines the problem. Section III introduces our DRGL model framework in detail. Section IV presents conducted experiments with analyses. Section V discusses related work. Finally, Section VI draws conclusions.

II. Problem Definition

In this section, we formally define the problem of time-aware service recommendation for dynamic mashup creation, and introduce some essential notations [10].

Definition 1 (Service Ecosystem): In a service ecosystem, $\mathbb{U} = \{u_1, u_2, \ldots, u_M\}$ and $\mathbb{S} = \{s_1, s_2, \ldots, s_N\}$ denote the sets of **users** and **services**, respectively.

Traditional service recommender systems suggest relevant services to users based on their historical behaviors, without considering the temporal order of their invoked services. However, users' service preferences change rapidly in most service ecosystems, and thus the users' recently involved services are crucial for modeling their real purpose. In practice, users tend to consume several services in a given time window for a specific purpose, and these consumer-centric services can be composed as a **Mashup**, which we define as follows:

Definition 2 (Mashup): A mashup m_t represents a sequence of services $s_t^1, s_t^2, \ldots, s_t^n$ invoked by a user sequentially within a specific time period *t*, where s_t^n represents the *n*-th service invoked in the *t*-th mashup. The set of all mashups, containing mashups invoked in different periods, is denoted as $\mathbb{M} = m_1, m_2, \ldots, m_t$.

The importance of modeling the temporal order of services in mashups motivates us to investigate time-aware service recommendation for dynamic mashup creation. We formally define the **Time-aware Service Recommendation** problem as follows:

Definition 3 (Time-Aware Service Recommendation): Given an ongoing mashup $m_{t+1} = \{s_{t+1}^1, s_{t+1}^2, \ldots, s_{t+1}^n\}$, the goal of time-aware service recommendation is to recommend a set of services from $\mathbb{S}$ that the target user is likely to be interested in during the next time step $n+1$, i.e., s_{t+1}^{n+1}, which can work in conjunction with the former services as a mashup.

Recent research has brought the graph structure into time-aware service recommendation to model relevant interests and mutual influence among mashups as in our previous works [10], [14]. We formalize the mashup relation graph as follows:

Definition 4 (Mashup Relation Graph): Let $\mathbb{G} = (\mathbb{M}, \mathbb{E})$ represent a **mashup relation graph**, where $\mathbb{E} \subseteq (\mathbb{M}, \mathbb{M})$ is the edge set in the graph denoting the relationships among the comprising mashups. We do not differentiate between directed and undirected graphs in this work.

Previous works have constructed mashup relation graphs by leveraging existing social networks [10], [11], where the relationships among users are treated as links between their invoked mashups. However, there is a domain gap between social relationships and mashup similarities, making them misaligned in most cases. Additionally, social relationships are expensive to collect and may become a limitation of graph neural networks as they are static and fixed. Therefore, in this paper, we propose to learn the graph structures among mashups and make time-aware service recommendations for mashup creation in a synergistic way. We formally define this problem as follows:

TABLE I
Notations and Explanations

Notation	Explanation
$\mathbb{U}$	user set
$\mathbb{S}$	service set
$\mathbb{M}$	mashup set
$\mathbb{G}$	mashup relation graph
$\mathbb{E}$	edge set of the mashup relation graph
u_i	user i
s_i	service i
s_t^n	the n-th service invoked in the t-th mashup
m_t	the t-th mashup in $\mathbb{M}$

Problem Formulation: Given the existing mashup set $\mathbb{M}$ and an ongoing mashup m_{t+1}, our task is to first learn the relations $\mathbb{E}^{t+1}$ among $\mathbb{M} \cup m_{t+1}$, which form a mashup relation graph $\mathbb{G}^{t+1} = (\mathbb{M} \cup m_{t+1}, \mathbb{E}^{t+1})$ as defined in Definition 4. Then, we recommend suitable services to m_{t+1} based on $\mathbb{G}^{t+1}$ for dynamic mashup creation, as defined in Definition 3.

The notations used throughout the article are summarized in Table I.

III. DRGL Model Framework

In this section, we first outline the overall architecture of our DRGL framework and give detailed descriptions of its main components, then analyze its learning process including the design of the loss function, followed by discussing the computation complexity of DRGL.

A. Model Architecture

Figure 2 shows the architecture of DRGL, which consists of five core modules. The framework is introduced from left to right.

(1) The ***Coarse Mashup Recalling Layer*** computes a graph adjacency matrix from the memory banks to discover hidden associations among massive mashups.

(2) The ***Self-Attentive Mashup Encoder*** models the real purpose of the dynamic mashup with a self-attention mechanism, following the recent advance in Transformer.

(3) The ***Self-adaptive Graph Learning*** utilizes mashups' dynamic purpose to adaptively discover a group of more influential neighbors in the coarse recalling results and omit those lower-impact edges in the graph.

(4) The ***Purpose Propagation Module*** leverages a graph-convolution network to re-scale weights of the mashup's neighbors and propagate their real purpose to the target mashup by a well-designed combination mechanism.

(5) The ***Service Recommendation Module*** obtains matching scores by modeling the similarity between each candidate service and the current target mashup.

Modules (1) and (3) implement a coarse-to-fine recalling strategy for graph structure learning among massive mashups. Modules (2) and (4) handle the mashup purpose modeling and propagation process. These two processes are performed

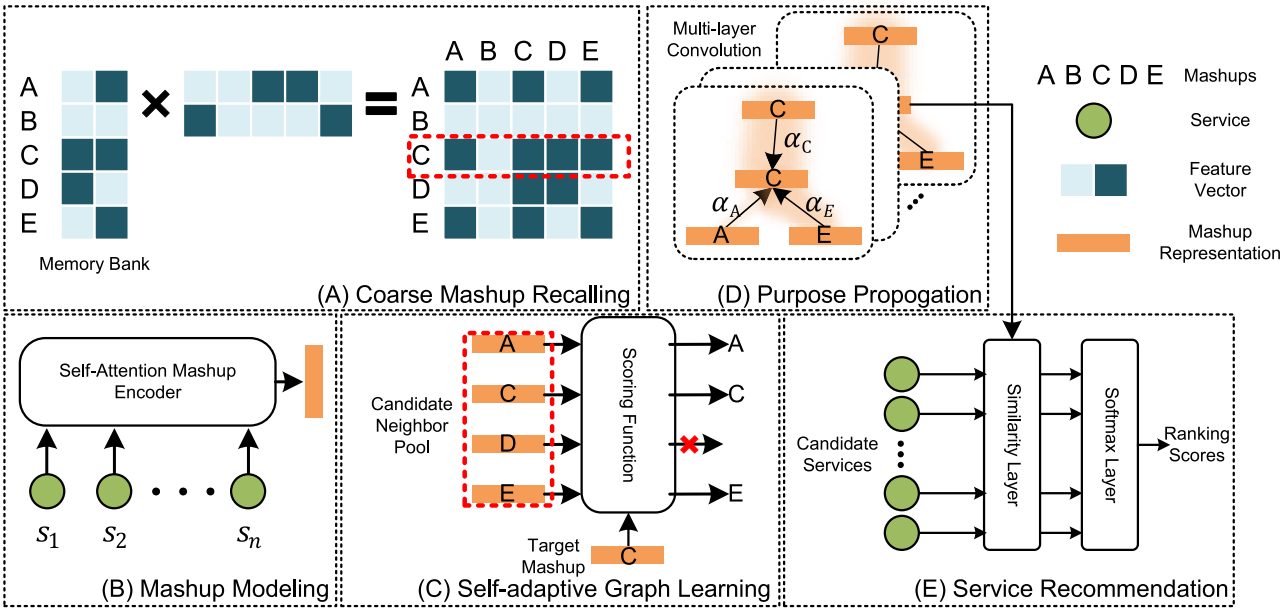

Fig. 2. **Model Framework.** The overview of the model structure of DRGL. It consists of five parts: a coarse mashup recalling layer, a mashup modeling module, a self-adaptive graph learning module, a purpose propagation module, and the final service recommendation module.

alternately and support each other to improve the time-aware service recommendation for dynamic mashup creation.

B. Coarse Mashup Recalling

The service platform contains a vast number of historical mashups, making it challenging to discover potential relationships among them in a learnable way [14] while optimizing model parameters efficiently. To overcome this problem, our DRGL framework employs a non-parametric coarse mashup recalling layer, which uses limited computing resources to identify potential relations between the target mashup and historical mashups.

For each mashup m_t, we represent its components using a service-level binary vector $v_t \in \{0,1\}^N$, where N is the total number of services in the platform. The i-th entry $v_t^i = 1$ indicates that service s_i is included in mashup m_t, while $v_t^i = 0$ means that it is absent. We can compute the similarity between two mashups by taking the inner product of their corresponding feature vectors $\mathbf{S}_{ij} = v_i v_j^T$, which represents the number of common services between mashups m_i and m_j.

To recall potential neighbors from the historical mashups, we maintain a feature memory bank $V \in \{0,1\}^{t\times N}$ to store all t mashups as recalling candidates. For a batch of target mashups, we concatenate their feature vectors into a query matrix $V_q = [v_1, v_2, \dots]$ and compute similarities between the targets and all candidates as follows:

$$\mathbf{S} = V_q V^T. \tag{1}$$

In practice, since only a small portion of the total N services are included in each mashup, the service-level binary vector v_t is extremely sparse. Therefore, when computing the similarity $S_{ij} = v_i v_j^T$, we can only focus on the services contained in the target mashup m_i, i.e., we only need to consider the non-zero entries in v_i and the values of corresponding positions in v_j. For example, suppose mashup m_i includes five services $[s_1, s_{20}, s_{500}, s_{20000}, s_{20005}]$, when calculating S_{ij}, we only need to count the number of ones in the positions [1, 20, 500, 20000, 20005] of v_j. This approach reduces the computation complexity from $O(N)$ to $O(n)$, where n represents the average number of services included in each mashup and $n \ll N$. Moreover, with our binary 0-1 setting, we can model the inner product of real-valued multiplication as the "and" operation in the corresponding positions that computers are good at, which further improves computational efficiency.

For each target mashup m_i, we select the top k_1 candidates with the highest scores in $\mathbf{S}_{i\cdot}$ as its coarse recalling neighbors, which can be denoted as: $\mathbf{N}_i = argtop_{k_1}(\mathbf{S}_{i\cdot})$.

C. Self-Attentive Mashup Encoder

Once we have the coarse-recalled neighbors, we use a self-attentive mashup encoder to capture the mashup's fast-changing purpose and comprehensive preference. Figure 3 illustrates how the mashup encoder is built using L stacked Transformer layers. At each Transformer layer, the service representation at each position is updated iteratively by aggregating information from all positions at the previous layer. Unlike conventional RNN-based methods, self-attention creates a global receptive field that captures temporal dependencies among services over long distances. In addition, abandoning the recurrent structure enables parallel computing, which enhances efficiency, especially when encoding numerous mashup vertices simultaneously. Next, we will discuss the mashup encoder's components in more detail.

1) Embedding Layer: Distributed representation techniques [18] have exhibited great potential in many fields. We encode a service s into a low-dimension latent space with an embedding vector $\mathbf{e}_s \in \mathbb{R}^d$, where d is the embedding size. As mentioned above, the Transformer layer does not

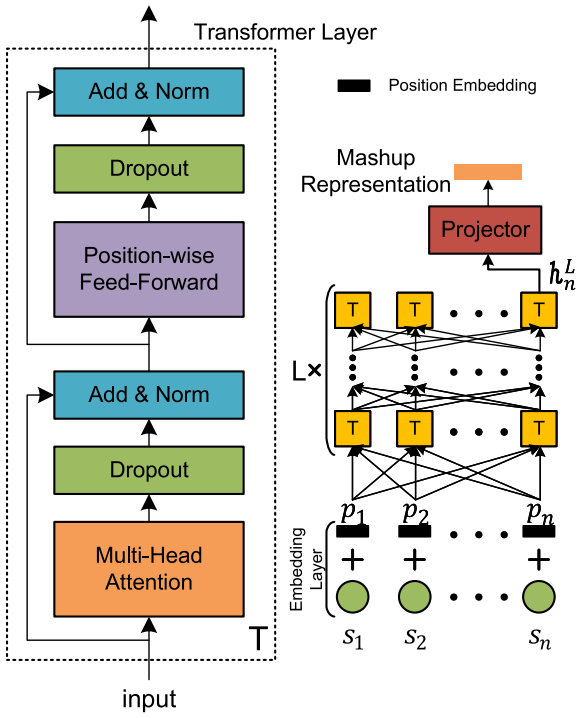

Fig. 3. A schematic view of the self-attentive mashup encoder.

distinguish the order of the services in the input mashup. In order to reflect the sequential feature of the mashup, we inject positional embeddings [16] into the input service embedding at the bottoms of the Transformer as shown in Figure 3, which can be formalized as follows:

$$\mathbf{h}_n^0 = \mathbf{e}_{s_n} + \mathbf{p}_n, \tag{2}$$

where $\mathbf{p}_n \in \mathbb{R}^d$ is the d-dimensional position embedding for position index n.

2) Transformer layer: As illustrated in Figure 3, the Transformer layer T contains two parts, namely Multi-Head Attention and Position-aware Feed-Forward Network.

Multi-Head Attention: The multi-head attention first linearly projects the service representation $\mathbf{H}^l = [\mathbf{h}_1^l, \mathbf{h}_2^l, \ldots, \mathbf{h}_n^l]$ into h subspaces with h groups of learnable parameters, followed with an attention function to produce the head representation, which is:

$$head_i = Attention\Big(\mathbf{H}^l\mathbf{W}_Q^{i,l}, \mathbf{H}^l\mathbf{W}_K^{i,l}, \mathbf{H}^l\mathbf{W}_V^{i,l}\Big), \tag{3}$$

where $\mathbf{W}_Q^{i,l}, \mathbf{W}_K^{i,l}, \mathbf{W}_V^{i,l} \in \mathbb{R}^{d\times d/h}$ are learnable parameters for $head_i$ at l-th Transformer layer. Specifically, the *Attention* function is implemented as the scaled dot-product attention following [16]:

$$Attention(\mathbf{Q}, \mathbf{K}, \mathbf{V}) = softmax\left(\frac{\mathbf{Q}\mathbf{K}^T}{\sqrt{d/h}}\right)\mathbf{V}, \tag{4}$$

where $\mathbf{Q}$, $\mathbf{K}$, and $\mathbf{V}$ are the projected result from the same $\mathbf{H}^l$. The term $\sqrt{d/h}$ is known as the temperature for softmax to avoid extremely small gradients. Afterward, we concatenate all these heads followed by a linear projection to obtain the self-attention output:

$$\mathbf{H}_{hidden}^l = [head_1; head_2; \ldots; head_h]\mathbf{W}_O^l. \tag{5}$$

Position-wise Feed-Forward Network: To depict the non-linear and cross-dimension interaction of each service representation, we also introduce the position-wise feed-forward network to the output $\mathbf{H}_{hidden}^l$ of the self-attention. It contains two fully connected layers and a *GELU* activation:

$$\mathbf{h}^{l+1} = GELU\Big(\mathbf{h}_{hidden}^l\mathbf{W}_1^l + \mathbf{b}_1^l\Big)\mathbf{W}_2^l + \mathbf{b}_2^l, \tag{6}$$

where $GELU(x) = x\Phi(x)$ is a derived ReLU activation from Bert [17]. $\Phi(x)$ is the cumulative distribution function of the standard Gaussian distribution.

To capture more complex interaction among services in a mashup hierarchically, we stack multiple Transformer layers to obtain a deep mashup encoder. Following [16], we adopt the residual connection [19], layer normalization [20], and dropout [21] to enable a deep architecture. Formally, we use the output of the latest service from the last Transformer layer $\mathbf{h}_n^{l+1}$ as the mashup representation: $\mathbf{h}_{m_t}$.

D. Self-Adaptive Graph Learning

In Section III-C, we obtained mashup representations that capture the fast-changing purpose more accurately. We then use self-adaptive graph learning to select related neighbors in a finer way based on these representations. Existing methods for graph structure learning typically construct a graph between a user and their potential neighbors by multiplying two matrices to measure node similarity [22], [23], [24]. However, these approaches may limit the model's ability to extract nonlinear patterns among nodes. Thanks to the well-designed coarse neighbor recalling process, we only need to calculate pairwise similarities among a subset of mashups, allowing us to design a more elaborate algorithm with controllable computation and memory. To extract complex relationships between mashups, we apply a two-layer neural network, which is illustrated by the following equation:

$$\tilde{\mathbf{S}}[t,k] = \mathrm{ReLU}\Big(\mathbf{l}^T\mathrm{ReLU}(\mathbf{W}\,concat([\mathbf{h}_{m_t}, \mathbf{h}_{m_k}]) + \mathbf{b})\Big), \tag{7}$$

Here, $\tilde{\mathbf{S}}[t,k] \in \mathbb{R}$ refers to the inferred similarity from mashup m_t to its neighbor m_k, and $\tilde{\mathbf{S}} \in \mathbb{R}^{N\times N}$ denotes the finer similarity matrix. Note that the entry $\tilde{\mathbf{S}}[t,k]$ is only calculated when m_k is in mashup m_t's coarse-recalling neighbors; otherwise, it is set to zero. Since we already make the similarity matrix sparse through coarse neighbor recalling in Section III-B, the computation cost can be greatly reduced to learn the finer similarity matrix. $\mathbf{W} \in \mathbb{R}^{d\times 2d}$, $\mathbf{b} \in \mathbb{R}^d$ and $\mathbf{l} \in \mathbb{R}^d$ are model parameters. We then select the top-k_2 neighbors with the highest relevance for the target mashup m_t as the finer selection and set the other neighbors' weights as zeros to form the adjacency matrix $\tilde{\mathbf{A}}$, which is formulated as follows:

$$\tilde{\mathbf{N}}_t = \mathrm{argtop}_{k_2}\Big(\tilde{\mathbf{S}}[t,:]\Big), \tag{8}$$

$$\tilde{\mathbf{A}}[t,k'] = \begin{cases} Softmax\Big(\tilde{\mathbf{S}}[t,k']\Big), & k' \in \tilde{\mathbf{N}}_t \\ 0, & \text{Otherwise} \end{cases}, \tag{9}$$

where argtop_{k_2} returns the index of the top-k_2 most relevant neighbors of the mashup and $\tilde{\mathbf{N}}_u$ denotes the neighbors selected from $\mathbf{N}_u$ after the similarity learning in Equation (7). The neighbor-wise *Softmax* is applied to normalize the self-adaptive adjacency matrix $\tilde{\mathbf{A}}$.

E. Purpose Propagation Module

After obtaining the inferred fine-grained neighbors and their influence weight from the graph structure learning process, we use a GCN-based model to obtain the neighbor context information of the target mashup by integrating its related neighbor mashup. In this section, we first introduce how we construct a purpose propagation graph, and then describe how we use a graph convolutional network (GCN)-based model to fuse related neighbors' purpose representations to update the target mashup's purpose representation.

According to Section III-D, we build a weighted graph with the nodes corresponding to target m_t and its selected neighbors in $\tilde{\mathbf{N}}_t$. We set the weight of each connected edge as the entry value $\tilde{\mathbf{A}}[t,k']$ in the self-adaptive adjacency matrix $\tilde{\mathbf{A}}$. Based on Section III-C, we use the purpose representation $[\mathbf{h}_{m_1}, \mathbf{h}_{m_2}, \ldots, \mathbf{h}_{m_t}]$ as the node's initial representation $[\mathbf{p}_{m_1}^{(0)}, \mathbf{p}_{m_2}^{(0)}, \ldots, \mathbf{p}_{m_t}^{(0)}]$. We use $\mathbf{P}^{(0)}$ to denote the original representation matrices, where each of their rows $\mathbf{P}^{(0)}[j,:] = \mathbf{p}_{m_j}^{(0)}$.

To simulate purpose propagation between pairs of users, we utilize a GCN-based module in DRGL. GCN is capable of updating a node's signal by aggregating and transforming its neighborhood information through different graph convolution operators, e.g., Chebyshev convolution and diffusion convolution [25]. By stacking multiple propagation layers, the mashup vertex incrementally gains more and more information from further reaches of the high-order neighbors. In this paper, we define our GCN-based purpose propagation as follows:

$$\mathbf{P}^{(l)} = \text{ReLU}\Big(\Big(\tilde{\mathbf{A}}\mathbf{P}^{(l-1)} + \mathbf{P}^{(l-1)}\Big)\mathbf{W}^{(l)}\Big), \tag{10}$$

where $\mathbf{W}^{(l)}$ is the shared and learnable weight matrix at layer l and $\mathbf{P}^{(l)}$ is the updated mashup representation matrix at layer l. Note that we keep target mashups' origin information by adding the term $\mathbf{P}^{(l-1)}$. We obtain the target mashup's final representation by stacking the purpose propagation layer L times.

F. Service Recommendation

To avoid the issue of gradient vanishing, we embrace a skip connection to add the mashup's original purpose representation $\mathbf{p}_{m_{t+1}}^{(0)}$ and the propagation output $\mathbf{p}_{m_{t+1}}^{(L)}$:

$$\mathbf{p}_{m_{t+1}} = \mathbf{p}_{m_{t+1}}^{(0)} + \mathbf{p}_{m_{t+1}}^{(L)}, \tag{11}$$

where $\mathbf{p}_{m_{t+1}}$ is the final representation of the target mashup m_{t+1}'s purpose on the next service. We obtain the probability of service s becoming the next potential service in the target mashup m_{t+1} on the whole service set $\mathbb{S}$ with a softmax function:

$$prob\Big(s|s_{t+1}^1, s_{t+1}^2, \ldots, s_{t+1}^n; \Big\{m_k, k \in \tilde{\mathbf{N}}_{t+1}\Big\}\Big) = \frac{\exp\big(\mathbf{e}_s^T \mathbf{p}_{m_{t+1}}\big)}{\sum_{i\in\mathbb{S}} \exp\big(\mathbf{e}_i^T \mathbf{p}_{m_{t+1}}\big)}, \tag{12}$$

where $\mathbf{e}_s$ is the embedding of the candidate service s. The embedding of a service can be calculated from its inherent features, such as the methods leveraging its functionality [26], non-functional features (i.e., QoS) [2], and service descriptions [27] etc.

G. Parameter Learning

In this section, we discuss parameter optimization and the training efficiency of our DRGL framework.

1) Optimization: Our optimization process tries to maximize the likelihood of positive services in the target mashup. To this end, we choose the negative log-likelihood as a loss function to supervise the training process, which is formulated as follows:

$$\mathcal{L} = -\sum_{t=1}^{T} log\ prob\Big(s_{t+1}^n | s_{t+1}^1, s_{t+1}^2, \ldots, s_{t+1}^{n-1}; \Big\{m_k, k \in \tilde{\mathbf{N}}_{t+1}\Big\}\Big), \tag{13}$$

where n represents the number of services in the mashup.

2) Time Complexity Analysis: The computation in our DRGL consists of four parts.

(1) In the coarse recalling layer, we match binary vectors in the memory bank to recall mashups sharing the same services as the target mashup. Let n be the average service number in a mashup, then the time complexity of coarse screening is $O(nt)$, where t is the number of total mashups in the memory bank V. The time complexity of $\text{argtop}_{k_1}(\cdot)$ operator for t mashups is less than $O(t)$, which is omitted.

(2) When inferring the comprehensive purpose of a target mashup and its k_1 neighbors, we use a transformer-based mashup encoder to learn the node's representation. Thus for a target mashup and its k_1 neighbors, the time complexity would be $O(k_1(n^2d + nd^2))$, where d denotes the embedding size. The term n^2d comes from the self-attention, while nd^2 comes from the position-wise feed-forward network.

(3) The self-adaptive graph learning computes the scores of the edges between the target mashup and each of its neighbors based on a shallow network. The complexity for this part is $O(k_1 d)$. The time complexity of $\text{argtop}_{k_2}(\cdot)$ operator for all k_1 mashups is $O(k_1 log k_2)$, which can be omitted due to $k_2 < d$.

(4) According to the relevant scores from the (3), k_2 neighbors' purpose representation will be aggregated to update the target mashup. The time will be $O(k_2 dL)$, where L denotes the number of propagation layers. Hence, the overall theoretical time complexity for a target mashup in one full step of model training is $O(nt + k_1 n^2 d + k_1 nd^2 + k_1 d + k_2 dL)$. In practice, we set $k_2 < k_1 \ll t$ and usually $L \leq 3$.

From the analyses above, we can find that the heaviest computation, in theory, is from the coarse recalling

layer, nt. However, existing graph structure learning methods [14], [22], [23] require to build a $t \times t$ similarity matrix by multiplying two embedding matrices for the global candidates recalling process, which has a time complexity of $O(td)$ for each. In most cases, $n < d$ and our non-parametric formulation eliminate the need for computing and storing the gradients for the embedding matrices, making it more scalable for big data applications. What's more, the self-attentive layer in the mashup encoder is fully parallelizable, which is amenable to GPU acceleration. In contrast, RNN-based methods [14] have a dependency on time steps (i.e., computation on time step n must wait for results from time step $n - 1$), which leads to an $O(n)$ time on sequential operations. In conclusion, our algorithm is computationally feasible in practice, and thus will support real-time queries in real-world service recommendation systems.

IV. Experiments

In this section, we present our extensive experiments with analysis.

A. Dataset Description

Because there lacks a large Web service repository containing sufficient time-aware mashup invocation and social connections for graph structure learning, We chose to use an offline service repository to construct our testbed. According to [10], offline services and Web services share three common features:

- Both offline services and Web services are massive and heterogeneous [3].
- Temporal dependencies exist in both fields. More specifically, a series of services consumed in a time window usually aim for some major purpose and are performed in a relatively fixed order, which can be composed as a "mashup" [1].
- Due to the inherent dependencies among services, there exist massive mashups sharing similar purpose in the historical database, and connections can be established among them [14].

Gowalla is a known website where users can record and share offline services they consume by checking-in, e.g., a meal in a restaurant or a show in a theater. We adopted the Gowalla platform to test and evaluate our DRGL on time-aware service recommendation for mashup creation. Cho et al. [28] randomly collected a total of 6,442,890 service records of 196,591 users over the time period of February 2009 - October 2010. We constructed the associated social network of these users using Gowalla's public API. Note that we also include the available social links to evaluate some social-based methods, and the effectiveness of our learned structures compared with the predefined social networks, even though DRGL does not rely on these social features. To further verify the effectiveness of our DRGL on datasets of different scales, we also introduce another book service dataset, LibraryThing. LibraryThing is a popular book-reviewing website that allows users to create an online catalog of books they have read or want to read. We treat each review as an observation of service invocation and segment users' behaviors into week-long mashups for their high activeness. Similar to Gowalla, we also collect explicit social network information on LibraryThing.

TABLE II
Statistics of the Datasets

Dataset	Gowalla	LibraryThing
Users	11,459	21,821
Services	16,435	11,805
Invocations	1,098,766	574,875
Avg. invocations per user	95.89	26.34
Avg. services per mashup	6.11	9.26
Social Connections	94,764	52,492
Density(Social Connections)	0.0721%	0.0110%
Avg. friends per user	8.27	2.41

Table II summarizes the numerical properties of the Gowalla and LibraryThing dataset.

In line with the approach taken by [10], our goal in the task of time-aware service recommendation is to recommend a service that is suitable for a target mashup, based on its containing services, in order to enable cooperative functionality. We treated each check-in in the dataset as a service consumption and segmented the data into day-long sessions, where each session can be seen as a mashup. Using the empirical frequency of offline consumption, we chronologically reserved the last six months for testing and filtered out services that did not appear in the training set. We then randomly and equally split these reserved mashups into a validation set and a test set.

B. Experimental Settings

1) Baselines: We compared our DRGL with four classes of baseline methods to evaluate the performance of time-aware service recommendation: (1) classical methods utilizing only users' implicit feedback; (2) social-aware methods that incorporate social influence; (3) time-aware methods that consider a user's actions in his current mashup; (4) graph-based time-aware models that consider both social relations and temporal information of users. We used seven baseline models in our experiments, and the class index of each model is indicated beside its name. The baseline models are as follows:

- *BPR-MF* [29] (1): A highly competitive method for implicit feedback-based recommendation that improves matrix factorization (MF) with the BPR objective function.
- *SBPR* [30] (2): A ranking model that considers social relationships in the learning process, assuming that users tend to assign higher ranks to items that their friends prefer.
- *SoReg* [31] (2): A method that utilizes the social network to regularize the latent user factors of matrix factorization.
- *GRU4Rec* [32] (3): A state-of-the-art approach that uses recurrent neural networks for session-based recommendations.
- *NARM* [33] (3): A method that employs RNNs with an attention mechanism to capture a user's main purpose and sequential behavior.

- *BERT4Rec* [34] (3): A sequential recommendation model that employs deep bidirectional self-attention to model user behavior sequences.
- *DGRec* [11] (4): A method that models dynamic user behaviors with an RNN and context-dependent social influence with graph attention neural networks based on the given social networks.
- *SGHAN* [10] (4): A method that models the service-level and friend-level differences existing in the mashup relation graph defined by the existing social networks.

2) Evaluation Metrics: To evaluate the performance of all algorithms for time-aware service recommendation, we used two widely-used metrics: Normalized Discounted Cumulative Gain@K (NDCG@K) and Hit Ratio@K (HR@K). Both metrics are higher the better. NDCG@K measures the position of the hits by assigning higher scores to hits at top ranks, making it position-aware. On the other hand, HR@K evaluates whether the test item is present on the recommendation list or not. The metrics are defined as follows:

$$NDCG@K = \frac{1}{R_N}\sum_{i=1}^{N}\frac{2^{rel_i-1}}{\log_2(1+i)} \tag{14}$$

$$HR@K = \frac{\sum_{i=1}^{K} rel_i}{|y_u^{test}|} \tag{15}$$

where K is the size of the recommendation list, $rel_i = 0$ or 1 indicates whether the service at rank i is in the test set, and R_N is the maximum possible cumulative component through ideal ranking. Also, $|y_u^{test}|$ is the number of services used by user u in the test set.

3) Implementation Details: We conducted all experiments on an Ubuntu server with the following specifications: [CPU: Intel Xeon E5-2680 v4 * 2, GPU: NVIDIA RTX 3090 * 4, RAM: 384GB]. To implement DRGL, we used Pytorch [35], a widely used Python library for neural networks.

We initialized the latent vectors with small random values for all the models, and for fair comparisons, we fixed the dimensions of latent factors to 100. Additionally, we set the number of hidden units of LSTMs or RNNs in all comparison models as 100. To tune hyperparameters, we conducted a grid search for the learning rate, which we varied amongst [0.005, 0.01, 0.02, 0.05]. We also applied a decay rate of 0.9 every five epochs. We used the Adam optimizer with $\beta_1 = 0.9$, $\beta_2 = 0.999$, and $\epsilon = 1e^{-8}$. To prevent overfitting, we added L_2 norm with a coefficient tuned from [0.001, 0.005, 0.01, 0.02, 0.1]. We set the batch size of all experiments as 200 and selected the best models by early stopping when the HR@20 on the validation set did not increase for 3 consecutive epochs.

For DRGL, we set the recall number k_1 of the coarse recalling as 100 and set k_2 to 20 based on the observation in Figure 5. We attempt to recall more potential neighbors but observed no significant improvement. In DRGL, we used an embedding matrix to represent service features, which projected a service s into a latent space with an embedding vector $\mathbf{e}_s$. Note that various embedding methods can be utilized for service representation learning. For example, Lam and Rossiter [27] analyze service descriptions (i.e., WSDL file) to obtain service embeddings. What's more, some methods propose to learn service embeddings based on their functionality [26] or non-functional features (i.e., QoS) [2]. However, to illustrate the effectiveness of our DRGL, we only used the ID feature of services to obtain the initial embeddings in our experiments since these methods are not the focus of our article.

TABLE III
OVERALL PERFORMANCE COMPARISON ON GOWALLA

Model	NDCG@10	HR@10	NDCG@20	HR@20
BPR-MF [30]	0.0351	0.0672	0.0405	0.0889
SBPR [31]	0.0465	0.0768	0.0509	0.0941
SoReg [32]	0.0442	0.0740	0.0498	0.0963
GRU4Rec [33]	0.0901	0.1574	0.1038	0.2122
NARM [34]	0.0905	0.1568	0.1047	0.2133
BERT4Rec [34]	0.0948	0.1662	0.1108	0.2256
DGRec [11]	0.1171	0.2051	0.1356	0.2788
SGHAN [10]	0.1240	0.2192	0.1436	0.2969
DRGL	**0.1323**	**0.2298**	**0.1539**	**0.3152**
Improv.	+6.73%*	+4.83%*	+7.17%*	+6.18%*

TABLE IV
OVERALL PERFORMANCE COMPARISON ON LIBRARYTHING

Model	NDCG@10	HR@10	NDCG@20	HR@20
BPR-MF [30]	0.0027	0.0059	0.0041	0.0114
SBPR [31]	0.0041	0.0079	0.0057	0.0145
SoReg [32]	0.0030	0.0065	0.0048	0.0135
GRU4Rec [33]	0.0075	0.0161	0.0107	0.0286
NARM [34]	0.0104	0.0205	0.0121	0.0305
BERT4Rec [34]	0.0080	0.0172	0.0115	0.0309
DGRec [11]	0.0080	0.0164	0.0108	0.0273
SGHAN [10]	0.0094	0.0195	0.0132	0.0340
DRGL	**0.0107**	**0.0217**	**0.0144**	**0.0366**
Improv.	+3.47%*	+6.42%*	+9.00%*	+7.46%*

C. Performance Comparisons

Table III summarizes the performance of different algorithms in terms of HR@K and NDCG@K with recommendation size $K = 10$, 20 on the Gowalla dataset. Our DRGL outperforms other methods in all evaluation metrics, and one-sample t-tests show that the improvements of DRGL over the strongest baseline are statistically significant (p-value < 0.05). Besides, We have the following observations:

- BPR-MF presents very limited performance, since it only considers the overall unordered interaction histories of a user. SBPR and SoReg perform better than BPR-MF in most cases, indicating that social relation information can help improve recommendation accuracy.
- Methods incorporating the temporal information bear better performances than those which do not. For example, GRU4Rec and NARM significantly outperform BPR-MF and obtain better performance than SBPR and SoReg. It might be because the temporal dependency of services in the mashups implicitly contains the purpose of the user, which is of great help to recommend the next service for the target mashup.

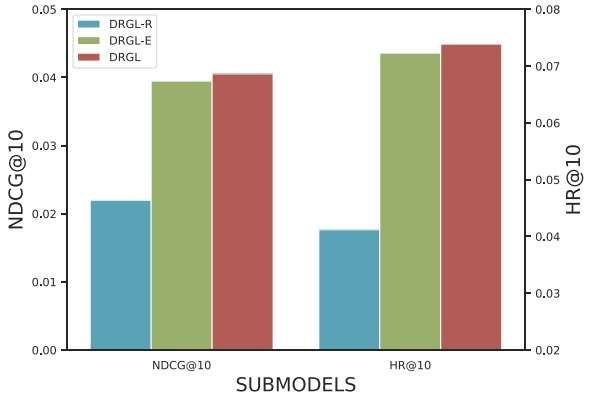

Fig. 4. Performance of variants of DRGL.

- The graph-based methods, including our DRGL, consistently outperform the other methods. These results verify that incorporating mashup relations is essential to enrich mashup representations and help with time-aware service recommendation.
- DRGL outperforms both DGRec and SGHAN by a large margin. DGRec and SGHAN are performed directly on the user's existing social networks. However, there exist gaps between the social relations and the similarities of mashup purpose among users, which may hamper the mashup representation learning. The results demonstrate that a learned graph structure has more expressive power to capture the dynamical dependencies among mashups, which is a critical issue in time-aware service recommendations.

D. Study of Coarse Mashup Recalling

The coarse mashup recalling process is one of the key characteristics in our proposed DRGL, allowing us to scale up to explore the possible relations among massive (over 600,000) historical mashups while maintaining an amount of tractable computation. We designed experiments to further explore the impacts of the coarse mashup recalling in DRGL and analyze the effectiveness of the adopted non-parametric memory bank matching, by constructing the following variants of DRGL:

- *DRGL-R:* A variant model of DRGL, in which the coarse neighbor screening process is replaced by randomly recalling k_1 mashup from the historical mashups.
- *DRGL-E:* A variant model of DRGL, in which the non-parametric feature matching is replaced by the inner product of the mashup embeddings. We obtain the mashup embedding by the average pooling of embeddings of its contained service.

Since it's infeasible to apply the **DRGL-E** on complete mashup set with its huge computation and memory cost for parametric computing, we create a 1/10 subset from the original Gowalla dataset to carry out the above experiments.

Figure 4 shows the experimental results of DRGL and its two variants. For the interest of space, here we only show the results of the NDCG@10 and HR@10 on the Gowalla dataset. From Figure 4, two observations can be made.

TABLE V
ABLATION STUDY COMPARING DRGL AND ITS VARIANTS

Variation	NDCG@10	HR@10	NDCG@20	HR@20
DRGL-last	0.1230	0.2193	0.1422	0.2953
DRGL-average	0.1227	0.2181	0.1438	0.3023
DRGL-RNN	0.1275	0.2231	0.1492	0.3070
DRGL	0.1323	0.2298	0.1539	0.3152

First, **DRGL-R** show the worst performance. This shows that randomly selected candidates in massive historical mashups may have nothing to do with the purpose of the target mashup. What's worse, the noise introduced by the irrelevant candidates may hamper the representation learning of our model. Thus, the coarse neighbor recalling is essential for the downstream graph learning by providing potentially relevant neighbors. Second, DRGL exhibits close performance to DRGL-E and even slightly surpasses it. This shows that our non-parametric feature matching method is not weaker than the learnable similarity calculation method in accuracy, especially in the large-scale recalling scenarios. Also, the average pooling operation may be affected by a few extremely deviating noisy services in the mashup, so that the mashup representation we get does not really reflect the purpose of the current mashup, thus failing to recalling the relevant mashup candidates. We argue that our non-parametric feature matching method based on the simple majority principle in the coarse stage recalling is more robust to noise and more able to recall relevant mashups.

E. Study of Self-Attentive Mashup Encoder

We conducted ablation studies to validate the effectiveness of our proposed transformer-based mashup encoder in DRGL and gain further insights into its self-attentive mechanism. We simply replace the mashup encoder in DRGL and create the following three variants:

- *DRGL-last:* considers only the last service embeddings in the mashup as the purpose of the mashup.
- *DRGL-average:* averages the service embeddings in the mashup and does not consider temporal dependency in a mashup.
- *DRGL-RNN:* uses the vanilla RNN as the mashup encoder to encode the series of service in the mashup.

Table V shows the performance of different variants. DRGL-last and DRGL-average perform the worst. DRGL-last ignore the previous service invocation and the last service has limited information to fully summarize the comprehensive purpose of the entire mashup. DRGL-average fails to consider the temporal dependencies among services in a mashup, thus may not precisely capture the mashup's evolving purpose. Compared to DRGL-RNN, the full DRGL performs better, suggesting that self-attention mechanism is a more powerful tool for the temporal mashup encoding. The reason behind may be that the transformer-based mashup encoder enables a global receptive field on the entire mashup, which can extract the real purpose of the dynamic mashup in terms of just a small number of services that really matter.

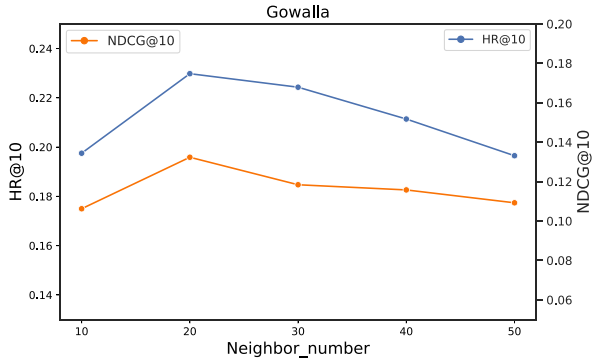

Fig. 5. Performance of DRGL under different selected neighbor numbers.

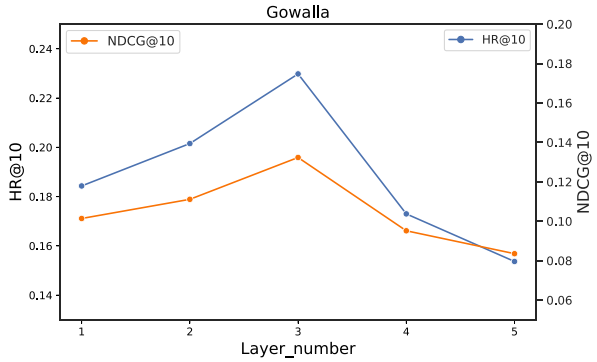

Fig. 6. Performance of DRGL under different convolution layer numbers.

F. Parameter Sensitive Studies

1) Effect of Neighbor Number k_2: This section investigates the influence of parameter k_2 controlling the number of selected neighbors on the fine-grained self-adaptive graph learning process. We vary k_2 from 10 to 50 while keeping other parameters fixed and measure the NDCG@20 and HR@20 results, as shown in Figure 5. We observe from Figure 5 that, with the increase of k_2, the performance is boosted at first since a larger group of neighbors can bring more useful information for the purpose propagation module. However, the performance drops after $k_2 = 20$ which might because it nullifies the second-stage graph structure learning and brings too much noise when introducing too many duplicate mashups.

2) Effect of Convolution Layers: DRGL uses a multi-layered convolutional approach to propagate information from high-order neighbors to enhance mashup representation learning and service recommendation. We experiment with layer numbers ranging from 1 to 5 to investigate the effect of multiple convolution layers on DRGL. Figure 6 shows the results. From Figure 6, we have the following two observations: First, our DRGL with more than one single convolution layer have better performance, demonstrating the benefits of information propagation from a mashup's high-order neighbors. Second, when further stacking more convolution layers on top of the 3-th layer, both metrics start to decrease. This is likely due to over-smoothing, where the target mashup absorbs too much neighbor information and loses its own. Additionally, a deeper architecture may introduce noise, and distant neighbors have little marginal improvement to offset this noise. This finding confirms that 3 convolution layers are optimal for our scenario, which is consistent with the finding in [10].

TABLE VI
COMPUTATION COST ON DIFFERENT USER SCALE

User Scale	Training Time (s/epoch)		
	DRGL	**DRGL-CN**	**DRGL-SL**
20K	12	9	100
100K	48	43	593
500K	204	186	3202
1M	456	334	8325

G. Computational Cost

In addition to our theoretical analysis in Section III-G2, we conducted empirical tests on synthetic datasets with varying numbers of users to compare the computation costs of DRGL and its variants. We introduce two variations: (1) DRGL-CN, which solely relys on the first-stage coarse mashup recalling to select neighbors; and (2) DRGL-SL, which solely relies on the second-stage self-adaptive graph learning to select neighbors. To create datasets of different user scales, we randomly sampled 10,000 users and their interaction history from the Gowalla dataset and replicated them several times. Table VI displays the training time with a GTX3090.

By observing Table IV, we can draw two conclusions. The first conclusion is that, the computation time of DRGL increases linearly with the increase in user scale, which demonstrates the excellent scalability of our algorithm in dealing with real-world mashup creation scenarios. The second conclusion is that the main bottleneck of the algorithm lies in the self-attentive mashup encoder. When we abandon coarse neighbor recalling and rely solely on the complex self-attentive mashup encoder for full-user calculation, the computational time will increase by multiple folds. This also reflects the importance of our coarse-to-fine recalling strategy in improving computational efficiency, and enhancing the scalability of the algorithm.

V. RELATED WORK

In this section, we discuss related work from four aspects: mashup creation, time-aware service recommendation, attention mechanisms, and graph neural networks.

A. Mashup Creation

Instead of recommending individual services separately based on their semantics [36], [37] or Quality-of-Service (QoS) [2], [3], [38], [39], [40], mashup creation methods recommend a bundle of compatible services to satisfy the functional requirements of the mashup as completely as possible. Our method aligns with this approach.

One approach to mashup creation is to use service co-occurrence information, as frequent service compositions suggest that these services are likely to collaborate with each other again in the future [41]. For example, SoCo [42]

proposed a mashup creation algorithm that utilizes frequent pair mining techniques, while in [43], a mashup creation algorithm mines reusable mashup composition patterns with pattern matching and retrieval to recommend services. However, these methods do not consider the functional requirements of a mashup, making it difficult to guarantee recommendation precision. In recent years, researchers have introduced information retrieval techniques to improve mashup creation. For instance, Bianchini et al. [44] present a proactive recommendation system that uses the semantic descriptions of mashup components to suggest mashup designs. Category-aware service clustering and distributed recommending methods are proposed for automatic mashup creation in [45], while Jain et al. [46] propose an approach that recommends APIs to create a mashup given a free-form text description by incorporating three heterogeneous but complementary semantic factors. To capture the deep relationships among services, [47], [48] combined a relational topic model and factorization machines technique for better mashup creation. Samanta and Liu [49] integrated the hierarchical Dirichlet process and probabilistic matrix factorization to rank Web services for mashup creation.

Deep Learning (DL) has gained attention in mashup creation and recommendation research due to its potential. Researchers have explored the use of DL in various ways, such as an integrated content and network-based service clustering and recommendation method proposed by Cao et al. [50]. Att-LDA [51] uses an attention-based topic model to highlight functional-oriented features in service descriptions for accurate semantic learning, while Wu et al. [52] propose a neural framework (MTFM) based on multi-model fusion and multi-task learning for precise mashup creation. Gu et al. [41] propose a compositional semantics-based service bundle recommendation model (CSBR) that addresses the semantic gap between mashup and service descriptions.

While existing studies on mashup creation are inspiring, they often neglect the explicit relationships among mashups, which can be leveraged to enhance the mashup creation. For instance, Li et al. [53] proposed measuring interest similarity between friends in video service systems to improve friend recommendation. Similarly, Zhang et al. [54] define a "social plane" that relies on recommended measurements to enable network performance expectation management. Lu et al. [55] propose a service recommendation model based on data compensation and dynamic user interest grouping in social networks. More recently, Wei et al. [10] proposed using the social network among users to depict the relations among their consumed mashups. However, relying on fixed social networks and heuristic rules may not always be optimal. To address this issue, we propose the DRGL framework in the paper, which jointly learns the graph structure and creates a dynamic mashup in a coupled framework.

B. Time-Aware Service Recommendation

In recent years, there has been a growing interest in modeling users' dynamic interests that change over time [56], [57], [58]. To make service recommendations more personalized, the time dimension has been incorporated into the service filtering process, resulting in a new class of context-aware recommendation known as Time-aware Service Recommendation (TASR). Different approaches have been proposed to tackle the TASR problem, which can be categorized into three groups: conventional, latent representation-based, and neural network (NN)-based methods.

Conventional methods utilize standard data mining or machine learning techniques, such as integrating time information into similarity measurement and QoS prediction for improved service recommendation [59], or using evolutionary clustering algorithms to evolve similar user clusters and accurately model user preferences over time [60]. Latent representation-based methods, such as [61], make recommendations by employing low-dimensional latent representations for each interaction within sequences with shallow models. NN-based approaches aim to capture users' current interests effectively, such as GRU4Rec [32], which applies gated recurrent units (GRU) to model sequential behaviors and make recommendations. Recently, several methods, such as SR-GNN [62], NISER+ [63], and GCE-GNN [64], have utilized graph neural networks (GNNs) to further analyze complex item transitions. In these methods, sequential user behaviors are modeled as graph-structured data.

Recently, graph neural networks (GNNs) have shown great potential in modeling complex relationships embedded in graph-structured data, such as user relationships. DGRec [11] and SGHAN [10] have combined graph attention networks (GATs) [65] with recurrent neural networks (RNNs) to model dynamic interests and mutual influence of users in social networks. However, we argue that fixed social networks may be suboptimal in depicting the relations among mashups and propose DRGL to jointly learn the graph structure among mashups and make time-aware service recommendations in a coupled framework.

C. Attention Mechanisms

Attention mechanisms have proven to be effective in a variety of tasks, including machine translation [66]. The underlying idea is that sequential outputs, such as candidate services for a mashup, depend on relevant parts of some input, which the model should focus on successively. Recent studies have incorporated attention mechanisms into service recommendation [10], [67], [68]. However, these methods essentially added attention as an extra component to the original model, e.g., attention+RNNs in [10]. Transformer [16], which mostly relies on self-attention modules, achieved state-of-the-art accuracy on machine translation tasks, outperforming RNN/CNN-based approaches. Motivated by the success of Transformer, we propose a new mashup encoder for TASR based on self-attention mechanisms. Our method enables considering long-range dependencies on dense mashups and accurately extracting the true purpose of dynamic mashups. BERT4Rec [34] also employs the deep bidirectional self-attention to better model user behavior sequences. Due to the state-of-the-art performance of transformers in sequence

modeling tasks, our paper adopts a similar structure to BERT4Rec for modeling user preferences reflected in our mashup. However, the main contribution of our paper lies in establishing relations between mashups through complex user preferences, thus further improving the representation learning of mashups for achieving effective time-aware service recommendation. In other words, our entire framework is model-agnostic and widely applicable to all sequence modeling methods, such as RNNs and LSTMs. Zeng and Paik [69] propose to using service embedding pretrained by lightweight BERT model to effectively perform dynamic service recommendations in edge computing. This paper only uses the BERT framework to obtain the pre-trained service embeddings, while our paper leverages the powerful modeling capabilities of transformers for serialized data to capture the fast-changing purpose and comprehensive preference of users in mashups. Moreover, this paper overlooks the impact of mashup relations on user preferences.

D. Graph Neural Networks

In recent years, graph neural networks (GNNs) have gained significant attention as they provide an efficient way to represent real-world data in the form of graphs, such as social networks, citation networks, and road maps. However, traditional operations designed for Euclidean data (e.g., convolutions) cannot be directly applied to graphs that have a variable size of unordered nodes and different numbers of neighbors [70]. Here we mainly review important graph convolutional network (GCN) models. Kipf and Welling [71] introduced GCNs for semi-supervised graph classification, which propagate node information along edges in the spatial domain. Meanwhile, spatial-based graph convolutional networks [72], [73] directly propagate node information along edges in the spatial domain. However, in these methods, the weights of neighbors rely on predefined static functions when updating the node representations. To overcome this limitation, Graph attention networks (GATs) [65] learn to assign different weights to the neighbors when updating the target nodes.

In service platforms, data can be naturally represented as a graph, where various entities such as users, services, and mashups can be modeled as nodes, and their direct interactions can be modeled as edges on the graph. Therefore, more and more recommender systems are gradually based on graph neural networks to capture deep collaborative signals on service platforms. For example, PinSage [74] is a popular graph neural network (GNN) model for item recommendation that is an extension of the well-known GraphSage [75] model. GraphRec [76] utilizes a graph attention network to encode user-item interactions and user-user social relations for social recommendations. NGCF [77] incorporates high-order connectivities through message-passing propagation to encode collaborative signals explicitly for recommendations. A simplified version of NGCF is introduced as LightGCN [78], which removes feature transformation and nonlinear activation, resulting in state-of-the-art prediction performance for recommendations. With the prevalence of heterogeneous graphs in real-world scenarios, which often involve various node types and relations among these nodes, there has been a growing interest in studying the methods for heterogeneous graph learning. For example, Guan et al. [79] present a novel heterogeneous graph learning-based out-fit recommendation scheme containing four types of entities, and devise a bi-directional graph convolution algorithm to fulfill the efficient optimization, which works on sequentially transferring knowledge via repeating upwards and downwards optimization. Also, Guan et al. [80] define a heterogeneous graph to creatively unify three types of entities and relations in the Personalized Fashion Compatibility Modeling context, and present a metapath-guided personalized compatibility modeling scheme to perform the heterogeneous graph learning, which adopts the pre-defined metapaths to explore the high-order relations among various entities, and hence strengthen the user and item embeddings.

However, existing GNN methods usually assume the graphs are given and fixed, while in this paper, we explore how to learn graph structures among mashups beyond social networks.

SRaSLR [81] propose the service social networks to model the relations among services, which is combined with the service descriptions for better service label recommendation. SRaSLR only considers the relation between individual services, while our paper models the relation between mashups. As mentioned in the introduction, each mashup is composed of multiple services working together in a specific order to achieve the functionality. Therefore, we need to further model the complex preferences and temporal effects within mashups. SRaSLR pre-defines a service relation network based on heuristic rules, while our method learns the relation of mashups dynamically in a learnable way. Fan et al. [82] propose Continuous Time Bipartite Graph (CTBG) to unify sequential patterns and temporal collaborative signals, which apply a temporal kernel to map continuous timestamps on edges to vector. And they introduce TGSRec to encode both sequential patterns and collaborative signals for better sequential recommendation. TGSRec models the temporal collaborative signals between users and items in the form of a graph through ingenious data structure design, and solves it using a graph transformer. However, it does not model the relationships between mashup sequences. We believe that neighbor mashups that are relevant to the target mashup are crucial for inferring the next interaction. In addition, TGSRec also establishes a graph structure based on heuristic rules, but this model-free approach lacks certain flexibility. KPGNN [83] formalize the task of social event detection and utilize a novel heterogeneous GNN-based model to detects events from the incoming social stream while possessing the power of interpreting complex social data to accumulate knowledge. KPGNN differs significantly from our time-aware service recommendation task as it focuses on addressing the social event detection problem. Therefore, the model structure used in this article also differs greatly from our DRGL. The article constructs a heterogeneous-graph based on various event knowledge entities using heuristic rules, rather than employing a learnable adaptive approach. Additionally, this article overlooks the influence of introducing temporal information into the nodes on the graph.

VI. Conclusion

Time-aware service recommendation is used to make recommendations more suitable for users' dynamically changing preferences. This approach uses a target user's service invocation behaviors within a period of time to predict the next service and form a dynamic mashup. Although researchers have proposed using advanced graph neural networks (GNNs) to capture information flow on predefined social networks and improve time-aware service recommendation, social networks are not always available and may not be consistent with shared interests among users.

To expand usage scenarios and further optimize recommendation effects, we present DRGL in this article to learn the graph structure and make time-aware service recommendations collectively for dynamic mashup creation. We address two notable challenges: the scalability issue and the accuracy issue. To address scalability issue, we propose a coarse-to-fine recalling strategy to learn the graph structure among mashups. To address accuracy issue, we leverage recent advances in self-attention mechanisms for mashup modeling and propose a transformer-based mashup encoder. Experimental results on a real-life local service dataset show that DRGL outperforms state-of-the-art methods and demonstrates the effectiveness of the learned graph structures.

References

[1] Y. Zhong, Y. Fan, K. Huang, W. Tan, and J. Zhang, "Time-aware service recommendation for mashup creation," *IEEE Trans. Services Comput.*, vol. 8, no. 3, pp. 356–368, May/Jun. 2015.

[2] Z. Zheng, H. Ma, M. R. Lyu, and I. King, "QoS-aware Web service recommendation by collaborative filtering," *IEEE Trans. Services Comput.*, vol. 4, no. 2, pp. 140–152, Apr.-Jun. 2011.

[3] S. Wang, Y. Ma, B. Cheng, F. Yang, and R. N. Chang, "Multi-dimensional QoS prediction for service recommendations," *IEEE Trans. Services Comput.*, vol. 12, no. 1, pp. 47–57, Jan./Feb. 2019.

[4] H. Mezni and M. Fayala, "Time-aware service recommendation: Taxonomy, review, and challenges," *Softw. Pract. Exp.*, vol. 48, no. 11, pp. 2080–2108, 2018.

[5] G. Fazelnia, E. Simon, I. Anderson, B. Carterette, and M. Lalmas, "Variational user modeling with slow and fast features," in *Proc. 15th ACM Int. Conf. Web Search Data Min. (WSDM)*, Virtual Event, AZ, USA, Feb. 2022, pp. 271–279.

[6] K. Huang, Y. Fan, and W. Tan, "Recommendation in an evolving service ecosystem based on network prediction," *IEEE Trans. Autom. Sci. Eng.*, vol. 11, no. 3, pp. 906–920, Jul. 2014.

[7] C. Yu and L. Huang, "Time-aware collaborative filtering for QoS-based service recommendation," in *Proc. IEEE Int. Conf. Web Services (ICWS)*, Anchorage, AK, USA, 2014, pp. 265–272.

[8] C. Wei, Y. Fan, J. Zhang, and H. Lin, "A-HSG: Neural attentive service recommendation based on high-order social graph," in *Proc. IEEE Int. Conf. Web Services (ICWS)*, Beijing, China, 2020, pp. 338–346.

[9] T. Liang, L. Chen, J. Wu, G. Xu, and Z. Wu, "SMS: A framework for service discovery by incorporating social media information," *IEEE Trans. Services Comput.*, vol. 12, no. 3, pp. 384–397, May/Jun. 2019.

[10] C. Wei, Y. Fan, and J. Zhang, "Time-aware service recommendation with social-powered graph hierarchical attention network," *IEEE Trans. Services Comput.*, vol. 16, no. 3, pp. 2229–2240, May/Jun. 2023.

[11] W. Song, Z. Xiao, Y. Wang, L. Charlin, M. Zhang, and J. Tang, "Session-based social recommendation via dynamic graph attention networks," in *Proc. 11th ACM Int. Conf. Web Search Data Min.*, 2019, pp. 555–563.

[12] P. Gu, Y. Han, W. Gao, G. Xu, and J. Wu, "Enhancing session-based social recommendation through item graph embedding and contextual friendship modeling," *Neurocomputing*, vol. 419, pp. 190–202, Jan. 2021.

[13] L. Song, Y. Bi, M. Yao, Z. Wu, J. Wang, and J. Xiao, "DREAM: A dynamic relation-aware model for social recommendation," in *Proc. 29th ACM Int. Conf. Inf. Knowl. Manag.*, 2020, pp. 2225–2228.

[14] C. Wei, B. Bai, K. Bai, and F. Wang, "GSL4Rec: Session-based recommendations with collective graph structure learning and next interaction prediction," in *Proc. ACM Web Conf. (WWW)*, 2022, pp. 2120–2130.

[15] A. N. Ngaffo, W. E. Ayeb, and Z. Choukair, "A time-aware service recommendation based on implicit trust relationships and enhanced user similarities," *J. Ambient Intell. Humaniz. Comput.*, vol. 12, no. 2, pp. 3017–3035, 2021.

[16] A. Vaswani et al., "Attention is all you need," in *Proc. 30th Annu. Conf. Neural Inf. Process. Syst.*, 2017, pp. 5998–6008.

[17] J. Devlin, M. Chang, K. Lee, and K. Toutanova, "BERT: Pre-training of deep bidirectional transformers for language understanding," in *Proc. Conf. North Amer. Chapt. Assoc. Comput. Linguist. Human Lang. Technol.*, 2019, pp. 4171–4186.

[18] T. Mikolov, I. Sutskever, K. Chen, G. S. Corrado, and J. Dean, "Distributed representations of words and phrases and their compositionality," in *Proc. Adv. Neural Inf. Process. Syst. (NeurIPS)*, 2013, pp. 3111–3119.

[19] K. He, X. Zhang, S. Ren, and J. Sun, "Deep residual learning for image recognition," in *Proc. IEEE Conf. Comput. Vis. Pattern Recognit.*, 2016, pp. 770–778.

[20] L. J. Ba, J. R. Kiros, and G. E. Hinton, "Layer normalization," 2016, *arXiv:1607.06450*.

[21] N. Srivastava, G. E. Hinton, A. Krizhevsky, I. Sutskever, and R. Salakhutdinov, "Dropout: A simple way to prevent neural networks from overfitting," *J. Mach. Learn. Res.*, vol. 15, no. 1, pp. 1929–1958, 2014.

[22] Z. Wu, S. Pan, G. Long, J. Jiang, and C. Zhang, "Graph WaveNet for deep spatial-temporal graph modeling," in *Proc. 28th Int. Joint Conf. Artif. Intell.*, 2019, pp. 1907–1913.

[23] Z. Wu, S. Pan, G. Long, J. Jiang, X. Chang, and C. Zhang, "Connecting the dots: Multivariate time series forecasting with graph neural networks," in *Proc. 26th ACM SIGKDD Int. Conf. Knowl. Discov. Data Mining*, 2020, pp. 753–763.

[24] Q. Zhang, J. Chang, G. Meng, S. Xiang, and C. Pan, "Spatio-temporal graph structure learning for traffic forecasting," in *Proc. AAAI Conf. Artif. Intell.*, vol. 34, 2020, pp. 1177–1185.

[25] Y. Li, R. Yu, C. Shahabi, and Y. Liu, "Diffusion convolutional recurrent neural network: Data-driven traffic forecasting," in *Proc. 6th Int. Conf. Learn. Represent.*, 2018, pp. 1–16.

[26] A. V. Paliwal, B. Shafiq, J. Vaidya, H. Xiong, and N. Adam, "Semantics-based automated service discovery," *IEEE Trans. Services Comput.*, vol. 5, no. 2, pp. 260–275, Apr.-Jun. 2011.

[27] G. Lam and D. Rossiter, "A Web service framework supporting multimedia streaming," *IEEE Trans. Services Comput.*, vol. 6, no. 3, pp. 400–413, Jul.-Sep. 2013.

[28] E. Cho, S. A. Myers, and J. Leskovec, "Friendship and mobility: User movement in location-based social networks," in *Proc. 17th ACM Int. Conf. Knowl. Discov. Data Min. (SIGKDD)*, San Diego, CA, USA, 2011, pp. 1082–1090.

[29] S. Rendle, C. Freudenthaler, Z. Gantner, and L. Schmidt-Thieme, "BPR: Bayesian personalized ranking from implicit feedback," in *Proc. 25th Conf. Uncertain. Artif. Intell. (UAI)*, Montreal, QC, Canada, Jun. 2009, pp. 452–461.

[30] T. Zhao, J. J. McAuley, and I. King, "Leveraging social connections to improve personalized ranking for collaborative filtering," in *Proc. 23rd ACM Int. Conf. Conf. Inf. Knowl. Manag. (CIKM)*, Shanghai, China, Nov. 2014, pp. 261–270.

[31] H. Ma, D. Zhou, C. Liu, M. R. Lyu, and I. King, "Recommender systems with social regularization," in *Proc. 4th Int. Conf. Web Search Web Data Min. (WSDM)*, Hong Kong, China, Feb. 2011, pp. 287–296.

[32] B. Hidasi, A. Karatzoglou, L. Baltrunas, and D. Tikk, "Session-based recommendations with recurrent neural networks," in *Proc. 4th Int. Conf. Learn. Represent. (ICLR)*, San Juan, Puerto Rico, May 2016, pp. 1–10.

[33] J. Li, P. Ren, Z. Chen, Z. Ren, T. Lian, and J. Ma, "Neural attentive session-based recommendation," in *Proc. ACM Conf. Inf. Knowl. Manag. (CIKM)*, Singapore, Nov. 2017, pp. 1419–1428.

[34] F. Sun et al., "BERT4Rec: Sequential recommendation with bidirectional encoder representations from transformer," in *Proc. 28th ACM Int. Conf. Inf. Knowl. Manag.*, 2019, pp. 1441–1450.

[35] A. Paszke et al., "PyTorch: An imperative style, high-performance deep learning library," in *Proc. Conf. Neural Inf. Process. Syst. (NeurIPS)*, Vancouver, BC, Canada, Dec. 2019, pp. 8024–8035.

[36] C. Li, R. Zhang, J. Huai, X. Guo, and H. Sun, "A probabilistic approach for Web service discovery," in *Proc. IEEE Int. Conf. Services Comput. (SCC)*, Santa Clara, CA, USA, 2013, pp. 49–56.

[37] L. Chen, Y. Wang, Q. Yu, Z. Zheng, and J. Wu, "WT-LDA: User tagging augmented LDA for Web service clustering," in *Proc. 11th Int. Conf. Service-Orient. Comput. (ICSOC)*, Berlin, Germany, 2013, pp. 162–176.

[38] S. Li, J. Wen, F. Luo, M. Gao, J. Zeng, and Z. Y. Dong, "A new QoS-aware Web service recommendation system based on contextual feature recognition at server-side," *IEEE Trans. Netw. Service Manag.*, vol. 14, no. 2, pp. 332–342, Jun. 2017.

[39] X. Su, M. Zhang, Y. Liang, Z. Cai, L. Guo, and Z. Ding, "A tensor-based approach for the QoS evaluation in service-oriented environments," *IEEE Trans. Netw. Service Manag.*, vol. 18, no. 3, pp. 3843–3857, Sep. 2021.

[40] G. Kang, J. Liu, Y. Xiao, B. Cao, Y. Xu, and M. Cao, "Neural and attentional factorization machine-based Web API recommendation for mashup development," *IEEE Trans. Netw. Service Manag.*, vol. 18, no. 4, pp. 4183–4196, Dec. 2021.

[41] Q. Gu, J. Cao, and Y. Liu, "CSBR: A compositional semantics-based service bundle recommendation approach for mashup development," *IEEE Trans. Services Comput.*, vol. 15, no. 6, pp. 3170–3183, Nov./Dec. 2022.

[42] A. Maaradji, H. Hacid, R. Skraba, and A. Vakali, "Social Web mashups full completion via frequent sequence mining," in *Proc. World Congr. Services*, 2011, pp. 9–16.

[43] S. R. Chowdhury, F. Daniel, and F. Casati, "Efficient, interactive recommendation of mashup composition knowledge," in *Proc. 9th Int. Conf. Service-Orient. Comput.*, vol. 7084, 2011, pp. 374–388.

[44] D. Bianchini, V. D. Antonellis, and M. Melchiori, "A recommendation system for semantic mashup design," in *Proc. Int. Workshops Database Exp. Syst. Appl. (DEXA)*, Bilbao, Spain, 2010, pp. 159–163.

[45] B. Xia, Y. Fan, W. Tan, K. Huang, J. Zhang, and C. Wu, "Category-aware API clustering and distributed recommendation for automatic mashup creation," *IEEE Trans. Services Comput.*, vol. 8, no. 5, pp. 674–687, Sep./Oct. 2015.

[46] A. Jain, X. Liu, and Q. Yu, "Aggregating functionality, use history, and popularity of APIs to recommend mashup creation," in *Proc. 13th Int. Conf. Service-Orient. Comput.*, vol. 9435, 2015, pp. 188–202.

[47] J. Cao, Y. Lu, and N. Zhu, "Service package recommendation for mashup development based on a multi-level relational network," in *Proc. 14th Int. Conf. Service-Orient. Comput.*, vol. 9936, 2016, pp. 666–674.

[48] B. Cao, M. Shi, X. F. Liu, J. Liu, and M. Tang, "Using relational topic model and factorization machines to recommend Web APIs for mashup creation," in *Proc. 10th Asia-Pac. Services Comput. Conf. Adv. Services Comput.*, vol. 10065, 2016, pp. 391–407.

[49] P. Samanta and X. Liu, "Recommending services for new mashups through service factors and top-K neighbors," in *Proc. IEEE Int. Conf. Web Services (ICWS)*, 2017, pp. 381–388.

[50] B. Cao, X. F. Liu, M. M. Rahman, B. Li, J. Liu, and M. Tang, "Integrated content and network-based service clustering and Web APIs recommendation for mashup development," *IEEE Trans. Services Comput.*, vol. 13, no. 1, pp. 99–113, Jan./Feb. 2020.

[51] M. Shi, Y. Tang, Y. Huang, and M. Lin, "Mashup tag completion with attention-based topic model," *Service Orient. Comput. Appl.*, vol. 15, no. 1, pp. 43–54, 2021.

[52] H. Wu, Y. Duan, K. Yue, and L. Zhang, "Mashup-oriented Web API recommendation via multi-model fusion and multi-task learning," *IEEE Trans. Services Comput.*, vol. 15, no. 6, pp. 3330–3343, Nov./Dec. 2022.

[53] Z. Li, J. Lin, K. Salamatian, and G. Xie, "Social connections in user-generated content video systems: Analysis and recommendation," *IEEE Trans. Netw. Service Manag.*, vol. 10, no. 1, pp. 70–83, Mar. 2013.

[54] Y. Zhang, P. Calyam, S. Debroy, and S. S. Nuguri, "Social plane for recommenders in network performance expectation management," *IEEE Trans. Netw. Service Manag.*, vol. 15, no. 1, pp. 97–111, Mar. 2018.

[55] X. Lu, J. Liu, S. Gan, T. Li, Y. Xiao, and Y. B. Liu, "Recommendation model based on dynamic interest group identification and data compensation," *IEEE Trans. Netw. Service Manag.*, vol. 19, no. 1, pp. 89–99, Mar. 2022.

[56] L. Wang, Y. Zhang, and X. Zhu, "Concept drift-aware temporal cloud service APIs recommendation for building composite cloud systems," *J. Syst. Softw.*, vol. 174, Apr. 2021, Art. no. 110902.

[57] A. Zenebe, L. Zhou, and A. F. Norcio, "User preferences discovery using fuzzy models," *Fuzzy Sets Syst.*, vol. 161, no. 23, pp. 3044–3063, 2010.

[58] Q. Zhang, D. Wu, G. Zhang, and J. Lu, "Fuzzy user-interest drift detection based recommender systems," in *Proc. IEEE Int. Conf. Fuzzy Syst. (FUZZ-IEEE)*, Vancouver, BC, Canada, Jul. 2016, pp. 1274–1281.

[59] Y. Hu, Q. Peng, X. Hu, and R. Yang, "Time aware and data sparsity tolerant Web service recommendation based on improved collaborative filtering," *IEEE Trans. Services Comput.*, vol. 8, no. 5, pp. 782–794, Sep./Oct. 2015.

[60] C. Rana and S. K. Jain, "An evolutionary clustering algorithm based on temporal features for dynamic recommender systems," *Swarm Evol. Comput.*, vol. 14, pp. 21–30, Feb. 2014.

[61] L. Hu, L. Cao, S. Wang, G. Xu, J. Cao, and Z. Gu, "Diversifying personalized recommendation with user-session context," in *Proc. 26th Int. Joint Conf. Artif. Intell.*, 2017, pp. 1858–1864.

[62] S. Wu, Y. Tang, Y. Zhu, L. Wang, X. Xie, and T. Tan, "Session-based recommendation with graph neural networks," in *Proc. 32rd AAAI Conf. Artif. Intell.*, 2019, pp. 346–353.

[63] P. Gupta, D. Garg, P. Malhotra, L. Vig, and G. Shroff, "NISER: Normalized item and session representations with graph neural networks," 2019, *arXiv:1909.04276*.

[64] Z. Wang, W. Wei, G. Cong, X. Li, X. Mao, and M. Qiu, "Global context enhanced graph neural networks for session-based recommendation," in *Proc. 43rd Int. ACM SIGIR Conf. Res. Develop. Inf. Retrieval (SIGIR)*, Jul. 2020, pp. 169–178.

[65] P. Veličković, G. Cucurull, A. Casanova, A. Romero, P. Lio, and Y. Bengio, "Graph attention networks," in *Proc. 6th Int. Conf. Learn. Represent.*, 2018, pp. 1–12.

[66] D. Bahdanau, K. Cho, and Y. Bengio, "Neural machine translation by jointly learning to align and translate," in *Proc. 3rd Int. Conf. Learn. Represent.*, 2015.

[67] L. Huang, Y. Ma, S. Wang, and Y. Liu, "An attention-based spatiotemporal LSTM network for next POI recommendation," *IEEE Trans. Services Comput.*, vol. 14, no. 6, pp. 1585–1597, Nov./Dec. 2021.

[68] H. Jin, P. Zhang, H. Dong, Y. Zhu, and A. Bouguettaya, "Privacy-aware forecasting of quality of service in mobile edge computing," in *Proc. IEEE World Congr. Services*, 2022, p. 18.

[69] K. Zeng and I. Paik, "Dynamic service recommendation using lightweight BERT-based service embedding in edge computing," in *Proc. 14th IEEE Int. Symp. Embedded Multicore/Many-Core Syst.-Chip*, 2021, pp. 182–189.

[70] Z. Wu, S. Pan, F. Chen, G. Long, C. Zhang, and S. Y. Philip, "A comprehensive survey on graph neural networks," *IEEE Trans. Neural Netw. Learn. Syst.*, vol. 32, no. 1, pp. 4–24, Jan. 2021.

[71] T. N. Kipf and M. Welling, "Semi-supervised classification with graph convolutional networks," in *Proc. 5th Int. Conf. Learn. Represent. (ICLR)*, Toulon, France, Apr. 2017, pp. 1–14.

[72] M. Niepert, M. Ahmed, and K. Kutzkov, "Learning convolutional neural networks for graphs," in *Proc. 33nd Int. Conf. Mach. Learn.*, 2016, pp. 2014–2023.

[73] J. Atwood and D. Towsley, "Diffusion-convolutional neural networks," in *Proc. Adv. Neural Inf. Process. Syst.*, 2016, pp. 1993–2001.

[74] R. Ying, R. He, K. Chen, P. Eksombatchai, W. L. Hamilton, and J. Leskovec, "Graph convolutional neural networks for Web-scale recommender systems," in *Proc. 24th ACM SIGKDD Int. Conf. Knowl. Discov. Data Min.*, 2018, pp. 974–983.

[75] W. L. Hamilton, Z. Ying, and J. Leskovec, "Inductive representation learning on large graphs," in *Proc. 30th Annu. Conf. Neural Inf. Process. Syst.*, 2017, pp. 1024–1034.

[76] W. Fan et al., "A graph neural network framework for social recommendations," *IEEE Trans. Knowl. Data Eng.*, vol. 34, no. 5, pp. 2033–2047, May 2022.

[77] X. Wang, X. He, M. Wang, F. Feng, and T. Chua, "Neural graph collaborative filtering," in *Proc. 42nd Int. ACM SIGIR Conf. Res. Develop. Inf. Retrieval*, 2019, pp. 165–174.

[78] X. He, K. Deng, X. Wang, Y. Li, Y. Zhang, and M. Wang, "LightGCN: Simplifying and powering graph convolution network for recommendation," in *Proc. 43rd Int. ACM SIGIR Conf. Res. Develop. Inf. Retrieval*, 2020, pp. 639–648.

[79] W. Guan, X. Song, H. Zhang, M. Liu, C. Yeh, and X. Chang, "Bi-directional heterogeneous graph hashing towards efficient outfit recommendation," in *Proc. 30th ACM Int. Conf. Multimedia*, 2022, pp. 268–276.

[80] W. Guan, F. Jiao, X. Song, H. Wen, C. Yeh, and X. Chang, "Personalized fashion compatibility modeling via metapath-guided heterogeneous graph learning," in *Proc. 45th Int. ACM SIGIR Conf. Res. Develop. Inf. Retrieval*, 2022, pp. 482–491.

[81] Y. Zhu, M. Liu, Z. Tu, T. Su, and Z. Wang, "SRaSLR: A novel social relation aware service label recommendation model," in *Proc. IEEE Int. Conf. Web Services*, 2021, pp. 87–96.

[82] Z. Fan, Z. Liu, J. Zhang, Y. Xiong, L. Zheng, and P. S. Yu, "Continuous-time sequential recommendation with temporal graph collaborative transformer," in *Proc. 30th ACM Int. Conf. Inf. Knowl. Manag.*, 2021, pp. 433–442.

[83] Y. Cao, H. Peng, J. Wu, Y. Dou, J. Li, and P. S. Yu, "Knowledge-preserving incremental social event detection via heterogeneous GNNs," in *Proc. Web Conf. WWW*, 2021, pp. 3383–3395.

Chunyu Wei received the B.S. degree in control theory and application from Tsinghua University, China, in 2019, where he is currently pursuing the Ph.D. degree with the Department of Automation. His research interests include services management, service recommendation, and social computing.

Yushun Fan received the Ph.D. degree in control theory and application from Tsinghua University, China, in 1990, where he is currently a Professor with the Department of Automation and the Director of the System Integration Institute and the Networking Manufacturing Laboratory. From September 1993 to 1995, he was a Visiting Scientist, supported by Alexander von Humboldt Stiftung, with the Fraunhofer Institute for Production System and Design Technology (FHG/IPK), Germany. He has authored ten books and published more than 300 research papers in journals and conferences. His research interests include enterprise modeling methods and optimization analysis, business process reengineering, workflow management, system integration, object-oriented technologies and flexible software systems, petri nets modeling and analysis, and workshop management and control.

Jia Zhang (Senior Member, IEEE) received the B.S. and M.S. degrees in computer science from Nanjing University, China, and the Ph.D. degree in computer science from the University of Illinois at Chicago. She is currently the Cruse C. and Marjorie F. Calahan Centennial Chair of Engineering and a Professor with the Department of Computer Science, Southern Methodist University. She has published more than 170 refereed journal papers, book chapters, and conference papers. Her research interests emphasize the application of machine learning and information retrieval methods to tackle data science infrastructure problems, with a recent focus on scientific workflows, provenance mining, software discovery, knowledge graph, and interdisciplinary applications of all of these interests in the area of earth science.

Zhixuan Jia is currently pursuing the Ph.D. degree with the Department of Automation, Tsinghua University. His research interests include services computing, service recommendation, and spatial–temporal data mining.

Ruyu Yan received the B.S. degree from Tsinghua University, China, in 2018, where she is currently pursuing the Ph.D. degree with the Department of Automation. Her research interests include services computing, recommender systems, and time-series prediction.