# Introduction

With the popularity of service-oriented software development (SOSD), a large number of Web services have been released on the Internet and lots of developers have built their mashups (a new type of Web application) that can provide specific functionalities by integrating one or more Web services, which reduce the cost and workload of system development and increase the reusability and quality of software systems [1]. The rapid growth of the number of web services brings severe challenges to the management and reuse of services. Service recommendation, a technology that recommends appropriate component services for mashup developers during their development process, has emerged at the historic moment.

Service recommendation usually refers to recommending services according to the explicit and implicit preferences of target mashups [1]. If a requirement to build a mashup is explicitly provided, content-based algorithms calculate the function matching degree between a service and the mashup based on the similarity between the content information of the service and the requirement.【】 Collaborative filtering (CF) based algorithms mine the implicit preference of the mashup based on the services that it has invoked.【】 Furthermore, other information of service usages, such as co-invocation and popularity, is also utilized to improve recommendation performance 【】.

Most of the existing service recommendation algorithms are designed for the mashup development in static scenarios, that is, they first model a semi-finished mashup with some component services offline, and then recommend some other services for this mashup once. However, these algorithms can hardly work in a real online conversational scenario where developers declare their requirements online and dynamically select component services to build a new mashup. The scenario summarized from the real experience of mashup developers can be described as follows. Suppose a developer wants to build a new mashup and enters his functional requirement into the service recommendation system (SRS). Then SRS analyzes the requirement and generates a list of candidate services, from which the developer selects one or a few component services as needed. If the recommendation result does not fully satisfy the requirement of the developer, SRS needs to continue to generate a new service list according to the requirement and the selected services. This step loops until the mashup is completely set up. Such scenario is different from the traditional static developing scenario, and it also has some special requirements for SRS.

1. **The entire developing process is indeed a session or transaction.** On the one hand, what a developer concerns in this process is how to realize his current short-term goal, that is, how to select some component services in terms of their functions and their relationship. And the mashups that the developer built before have little impact on the current session. On the other hand, this session is an integral whole and next service is supposed to coordinate with the selected services to meet the developer’s requirements. Therefore, **SRS ought to consider the complex interaction among the mashup to be built, the selected service and next service when making recommendations.**
2. The session to build a new mashup is established and completed online. SRS needs to work in the following two situations: (1) if developers do not select any component services ever, SRS should make personalized recommendations as soon as their requirements are declared online. (2) when developers have selected one or a few component services, SRS need to generate new recommendations based on initial requirements and these real-time feedbacks.

There is no difference between the first case and the typical user cold start problem that have been researched by many works【】. We will study service recommendation in the second case, that is, **given a developer's requirements and one or a few component services that have** **been selected (referred as to the selected services) in the current session, how can SRS recommend next service for the developer to complete the mashup?**

Most of the existing service recommendation technologies cannot meet the above needs or work well in the above case. Model-based CF methods first train their models and obtain the representations of mashups and services in the training set offline, and then predict the ratings of services over these existing mashups during the online serving phase, e.g., MF, NCF,LDA【】. But they cannot get the representations of the new mashups built online from the trained model. Although user-based CF and content-based methods can work in this scenario, they only consider the relationship between the new mashup and next service, and ignore that between the selected services and next service. In fact, the selected services have some impacts on the selection of next service. As a combination of content-based and CF-based algorithms, hybrid algorithms face the same problems as these two algorithms.

In this study, we propose an interaction-oriented model to recommend next service in the online conversational mashup development scenario. We employ Multi-Layer Perceptron (MLP) to capture the **complex interaction** among the new mashup, the selected services and next service. Specifically, we use the **attention mechanism** to distinguish the influence of different selected services on the next service. Our model is applicable to various kinds of information, for example, the content information and the invocation records between mashups and services. From each information, we first obtain the feature representations of a new mashup, the selected services and next service, then learn a specific interaction in this space and finally make a prediction. Our model is also very combinable. We can integrate the interactions in different spaces to make more accurate recommendation. Our model can serve online and make new recommendation in real time as long as some services are newly selected.

The main contributions of this work are:

1. We summarize a scenario of building a mashup in an online conversational session and analyze its requirements for the SRS that work in it.
2. We design an interaction-oriented model for online session-based service recommendation. With the help of MLP and the attention mechanism, our model can recommend services that are compatible with the selected services and satisfy users’ developing requirements when serving online.
3. We propose two model examples that learn specific interaction using the content information and the invocation between mashups and services respectively. Combining these two examples, we obtain a hybrid model that integrates the interactions in different information spaces and makes more accurate recommendation.
4. The experimental results on the real data set show that compared with the existing algorithms, our hybrid model has achieved the best results in various indicators.

# an interaction-oriented model FOR online conversational mashup development

## Problem Statement.

A service repository can be expressed as (M, S), where M is the set of mashups that have been built and S is the set of existing services. The invocation history between Mashups and services can be regarded as implicit feedback data and converted into an invocation matrix *MS*, where the value at m-th row and s-th column is:

.

Each mashup or service is attached with many types of information. The content information of a mashup or a service (such as textual descriptions and tags) describes its requirements or functionality. Besides, a service also has the information about its provider.

Above all, the problem to be solved in this paper can be described as follows. Suppose a developer has provided his requirement (referred to as *MReq*) to build a new mashup and selected one or a few component services (referred to as the selected services) online, how can SRS recommend next service to the developer based on the available information (i.e., *MReq*, the selected services, *MS*, the content and provider information in the repository) at this time?

## An interaction-oriented Model.

There exists a complicated relationship or interaction among MReq, the select services and next service. On the one hand, next service and the select services may be substitute or complementary to each other, etc. On the other hand, the select services and next service work together to satisfy the requirements of the developer, . The interaction determines whether the developer will select a candidate service as next service or not to some extent.

Therefore, we design a deep learning based model to capture the complex interaction and predict the probability of the developer selecting next. The K services with the highest scores or probabilities are taken as the recommendation list.

Our model consists of three layers: the feature extraction layer, the interaction layer and the output layer.

### The feature extraction layer:

Many kinds of information is available in the repository, such as the content information and the invocation between mashups and services. In this layer, we input one kind of information of a new mashup , the selected services and a service s, and then get their feature representation in the same space using some strategies and feature extractors. Note that what we get here is the representation of a mashup or a single service.

### The interaction layer

This layer learns the interaction among m, the selected services, and s. Due to the complex relationship among them, the hasty assumptions and definitions on the interaction may not fit the reality. Considering that MLP can theoretically fit arbitrary function [], we employ the MLP to model the interaction. We select PReLU as activation function because it improves model fitting with nearly zero extra computational cost and little overfitting risk.【】

An intuitive idea is that we first use multiple MLPs to learn the interactions among m, s and each selected service respectively, and then integrate these interactions by another MLP. But this method consumes too much computing resources. Alternatively, we first compress the representations of all selected services into a lower-dimensional vector based on the attention mechanism. On this basis, it takes only one MLP to learn the complex interaction among m, s and the selected services.

#### Attention based feature integration

To get an overall representation of the selected services, the simplest method is to directly **concatenate** their representations. However, the concatenation will increase the dimension of the final representation and reduce the efficiency of the model, especially when the size of selected services is large. What’s more, next-item is actually not sensitive to the order of the selected services. But the concatenation result does not satisfy this property obviously.

Another popular solution is to preform average **pooling** or sum pooling on the representation of each service. Although generating a fixed-length representation, the pooling method has its disadvantage.Let's analyze a case first. Suppose a developer wants to build a mashup to help tourists prepare for their trip, and he has already selected a service to make hotel reservation, a weather forecast service, Baidu Map and an electronic payment service, Alipay. Considering that tourists need to prepay fee online after booking a hotel and Alipay has been selected, the developer may invoke WeChat Pay to complement Alipay next. Obviously, the only services that play a leading role in the selection of WeChat Pay are the hotel reservation service and Alipay. So we believe that each selected service has a different influence on the selection of next service. It means that the representation of each service should be integrated with a distinct weight. However, the pooling method actually weight them equally.

Inspired by the above analysis, we design an **attention based** method to effectively **integrate the representation of each selected service.** It **pays attention to the selected services that are more related to** (i.e., with similar or same functions to substitute each other, or with different functions to complement each other) **next service** and ignore unnecessary services.

We use the weighted sum of feature of each selected service to represent context services.

where is formally the weight of a selected service . The physical meaning of is the degree of correlation between and *s* or the contribution of to the user's selection on *s*. is jointly determined by the feature of and *s*. For two vectors, we measure their similarity by their element-wise multiplication and their difference by their element-wise subtraction. The results of these two operations can be seen as prior knowledge to help model the correlation between and *s*. We concatenate them with the feature of and *s* and input the result into an MLP to automatically learn the correlation between and *s* and obtains a scalar score. The process can be described as:

where denotes element wise multiplication and denotes element-wise subtraction.

Finally, we input this score into a softmax layer to calculate a final weight scalar **:**

Different from the method of pooling, we adaptively calculates the representation of selected service according to the correlation between each selected service and s. Therefore, using our method, different selected services have different contribution weights to the overall representation and the representation varies with the candidate service. This means that, compared with the average pooling method, our attention-based method can obtain a more adaptable representation and have a better expression ability.

### Output layer

We feed the learnt interaction vector into a softmax layer and regard the output value on “1” as the probability of the service would be selected as next service.

Session，online分析：

Our model notices that mashups are established and set up online in a session mode. To make recommendation, it not only models the interaction among the new mashup, all selected services and next service, but emphasizes the services that have greater impacts on the next service.

Considering that developers gradually build new mashups in the conversation with SRS, our model respond to the user's selection behavior instantly. It recruited the newly selected services and re-evaluated each candidate service according to updated/ newly learned interaction.

model instance

Fed information about the new mashup, all selected services and next service, our model learns their complex interactions and makes predictions. In this section, we input **two kinds of information** (that is, the content information and the invocation between mashups and services) into the model and learn the interaction from different perspectives, **obtaining two instances** of our model. We will detail the main difference between the two **instances, that is, their input** information and feature representation in this section. Finally, we propose a hybrid model to integrate the interactions learned from the two models and make more accurate prediction.

It is worth mentioning that we do not take **the object ID** as the input information of our model like the common recommend methods based on deep learning. The reason is that the ID of the new mashup does not appear in the model trained offline, and it is difficult to train its high-quality embedding with scarce instances online.

Model in content information

When considering a candidate, the developer will consider the functional interaction among MReq, context\_services and s. According to the above analysis, there exist interactions among MReq, context\_services and s. Therefore, our model employ their content information to model their interaction from a functional perspective.

In the service repository, the functional description of a mashup or a service after processing, that is, their content information, generally fall into two forms: word sequence (such as description) and separate word set (such as tags). So is Mreq. We first adopt two deep learning techniques to process these two forms of information respectively, and then concatenate the extracted two features as the functional representations of a mashup or a service.

To apply deep learning technology to feature extraction, we need to preprocess content information with the help of word embedding. We first convert all terms into sparse binary vectors with one-hot encoding, e.g., [0, 0, …, 1, …, 0]… Then we input these vectors into an embedding layer and map each term into a dense vector or an embedding. Finally, we truncate or pad the content information and stack the embeddings of the terms in it, transforming the information into a matrix with a fixed size. The process can be described as

|  |  |  |
| --- | --- | --- |
|  |  | (1) |

where denotes the word-embedding representation of the content information, is the length of the processed information, is the -th term in the information, and is the -dimensional word embedding for .

For the word sequence information, we use the text\_inception we proposed in [] to extract its features. The method first parallels stacked convolution layers to extract local features in the word sequence, improving its efficiency and non-linearity. Then it employ a global average pooling (GAP) layer to emphasize the important features. Finally, an MLP is used to perform non-linear transformation on the features. This process can be simply expressed as:

There is no order among multiple terms in the content information in the form of a separate word set, so text\_inception cannot be applied to it. Instead, we retrieve and average the embedding of each term to get a fixed-length representation of the information:

(8)

where is the embedding of the -th term in the set and is the size of the set.

For a mashup or service, we concatenate the two features extracted from its content information, **and** , and obtain a representation depicting its functionality:

(9)

where denotes the concatenation operation.

At this point, we can transform the content information of MReq, s and selected service in the natural language form into real-valued feature vectors. Then we can use MLP to model their functional interaction and finally make a prediction.

Model in historical information

Besides the content information, the historical invocations between existing mashups and services can also improve the recommendation result. To apply our model to the historical information, the most difficulty is how to get the representations of MReq, s and selected service based on the historical information. We will detail it in this subsection.

Model-based CF algorithms, such as PMF, LDA, etc.【】, can obtain representations of existing mashups and services. To this end, we use node2vec,【】 a typical graph embedding method, to process the graph transformed from the historical invocation matrix MS. Compared to traditional matrix factorization (MF) methods, node2vec is capable of capturing more complex non-linear relationships.

Note that the new mashup is built online and its representation cannot be obtained by the model trained offline. Inspired by the User-based CF method, we first look for some mashups existing in the rep repository and similar to the new mashup (i.e., neighbor mashups). Then we can obtain the feature representation of the new mashup using that of its neighbor mashups and the similarities between them.

We adopt the method proposed by 【】to calculate the similarity between the new mashup and an existing mashup. The content information of Mreq and the information of the newly selected services are exploited to get more accurate similarity.

We first extract topics from the content information by Latent Dirichlet Allocation (LDA) and use top3 topics to represent content of a mashup or service. Then a heterogeneous information network (HIN) containing two kinds of objects (mashup and service) is constructed. ~~In this network, each mashup or service has its topics and tags, and the service also has provider information. Mashups and services are linked because of their invocation.~~ We use the meta-path-based method to calculate six types of associations or similarities between two mashups: having same topics, labelled by same tags, co-invoke one service, co-invoke similar services that have same topics, same tags or same provider. Different weights are assigned to each similarity, and the weighted sum of these six similarities are calculated to get an overall similarity between two mashups. We set the weights to the values pre-trained by PasRec【】. The whole process can be expressed as：

|  |  |
| --- | --- |
|  | (11) |

Our similarity calculation method take into account the information of selected services. When developers select new services, the method can find more accurate neighbor mashups for the mashup and get a better picture of it. The meta-path-based method to calculate mashup similarity is efficient. We can adopt some pruning strategies, that is, set some rules to reduce the size of candidate mashups, to improve the efficiency of looking for neighbor mashups.

Then most similar mashups of are selected as its neighbor mashups in terms of their overall similarities.

Finally we can get the weighted representation of mashup :

|  |  |  |
| --- | --- | --- |
|  |  | (12) |

where is a neighbor mashup of , denotes the similarity between and , and is the representation of obtained by node2vec.

Up to now, we have obtained the representations of MReq, context\_services, and s in the same feature space based on historical information. Then an MLP is utilized to model their interaction in this space.

Hybrid model

The above two examples learn different forms of interaction among Mreq, selected services and s based on content information and historical information respectively. But the prediction is made based on only one kind of information and is of limited effectiveness. Below, we propose a hybrid example that integrates multiple interactions and show its architecture is in Figure 2.

Our hybrid model consists of multiple underlying components, an integration layer and a predict layer. Each underlying component is actually a model instance. It receives a kind of information and learns a unique kind of interaction among Mreq, selected services and s.

We first input different kinds of information into multiple model instances. Then, the integration layer [incorporate](https://www.dictionary.com/browse/incorporate) the interactions learned from different components with an MLP. Finally the output layer make a more accurate prediction.

This means that our model is extensible. If a new form of information is available, we can first learn a new interaction based on the information, and then easily integrate it into the original model to improve the final prediction.

## Offline Model Learning

Given content information and historical invocations between existing mashups and services, our model predicts the possibility of a mashup m choosing a candidate service s as next service when it has selected some services. Obviously, the predicted value should approximate 1 for positive samples and be close to 0 for negative samples, so the likelihood function of the model is:

(19)

where θ denotes model parameters and info is the information used in the model. is the predicted probability of m invoking s next step.

To maximize the above likelihood is equivalent to minimize the following loss function:

(20)

Based on the loss function, we employ Adam algorithm [] to update model parameters on each mini-batch.

Since MISR is a hierarchical model with several nested MLPs, directly updating all parameters may result in slow convergence. So we first train each underlying component or model instance. Then we train the integration layer and the output layer by integrating the learned interactions and mapping the result to target output. On this basis, we fine-tune the whole model.

When SRS has helped developers build enough mature mashups or periodically (e.g. every other week), we update the model based on the training samples generated from these new mashups and their component services.

Online serving

our model can efficiently serve for developers online.

内容信息经过特征提取器的处理，被转化为密集向量表示。存在的Mashup和服务的embedding通过离线训练的node2ve得到。一旦找到新mashup的近邻mashup，便可以得到新mashup和服务的表示。在隔离的交互层中，待选服务首先和已选择服务发生交互，然后三者之间通过一个MLP相互交互。在整合层，多种交互被整合，最终输出层输出预测。所有操作都是高效的，因此线上serving的效率很高。