Controllable Multi-Interest Framework for Recommendation

Yukuo Cen[†], Jianwei Zhang[‡], Xu Zou[†], Chang Zhou[‡], Hongxia Yang[‡*], Jie Tang[†*]
[†] Department of Computer Science and Technology, Tsinghua University
[‡] DAMO Academy, Alibaba Group
{cyk18,zoux18}@mails.tsinghua.edu.cn
{zhangjianwei.zjw,ericzhou.zc,yang.yhx}@alibaba-inc.com
jietang@tsinghua.edu.cn

ABSTRACT

Recently, neural networks have been widely used in e-commerce recommender systems, owing to the rapid development of deep learning. We formalize the recommender system as a sequential recommendation problem, intending to predict the next items that the user might be interacted with. Recent works usually give an overall embedding from a user's behavior sequence. However, a unified user embedding cannot reflect the user's multiple interests during a period. In this paper, we propose a novel ***co***ntrollable ***m***ulti-***i***nterest framework for the sequential recommendation, called ComiRec. Our multi-interest module captures multiple interests from user behavior sequences, which can be exploited for retrieving candidate items from the large-scale item pool. These items are then fed into an aggregation module to obtain the overall recommendation. The aggregation module leverages a controllable factor to balance the recommendation accuracy and diversity. We conduct experiments for the sequential recommendation on two real-world datasets, Amazon and Taobao. Experimental results demonstrate that our framework achieves significant improvements over state-of-the-art models[1]. Our framework has also been successfully deployed on the offline Alibaba distributed cloud platform.

CCS CONCEPTS

• **Information systems** → **Recommender systems**; • **Computing methodologies** → **Neural networks**.

KEYWORDS

recommender system; sequential recommendation; multi-interest framework

ACM Reference Format:
Yukuo Cen, Jianwei Zhang, Xu Zou, Chang Zhou, Hongxia Yang and Jie Tang. 2020. Controllable Multi-Interest Framework for Recommendation. In *The 26th ACM SIGKDD Conference on Knowledge Discovery and Data Mining (KDD '20), August 22-27, 2020, San Diego, California, USA.* ACM, New York, NY, USA, 10 pages. https://doi.org/10.1145/nnnnnnn.nnnnnnn

[*] Hongxia Yang and Jie Tang are the corresponding authors.
[1] Code is available at https://github.com/cenyk1230/ComiRec

KDD '20, August 22-27, 2020, San Diego, California, USA

ACM ISBN 978-1-4503-XXXX-X/18/06…$15.00
https://doi.org/10.1145/nnnnnnn.nnnnnnn

1 INTRODUCTION

The development of e-commerce revolutionized our shopping styles in recent years. Recommender systems play a fundamental role in e-commerce companies. Traditional recommendation methods mainly use collaborative filtering methods [47, 48] to predict scores between users and items. Recently, neural networks have been widely used in e-commerce recommender systems, owing to the rapid development of deep learning. Neural recommender systems generate representations for users and items and outperform traditional recommendation methods. However, due to the large-scale e-commerce users and items, it is hard to use deep models to directly give the click-through rate (CTR) prediction between each pair of users and items. Current industrial practice is to use fast K nearest neighbors (e.g., Faiss [25]) to generate the candidate items and then use a deep model (e.g., xDeepFM [33]) to integrate the attributes of users and items to optimize the business metrics such as CTR.

Some recent works leverage graph embedding methods to obtain representations for users and items, which can be used for downstream applications. For example, PinSage [56] builds on GraphSAGE [15] and has applied graph convolutional network based methods to production-scale data with billions of nodes and edges. GATNE [6] considers different user behavior types and leverages a heterogeneous graph embedding method to learn representations for users and items. However, this kind of method ignores the sequential information in the user behaviors and cannot capture the correlations between adjacent user behaviors.

Recent researches [7, 27, 36] formalize the recommender system as a sequential recommendation problem. With a user's behavior history, the sequential recommendation task is to predict the next item he/she might be interested in. This task reflects the real-world recommendation situation. Many recent models can give an overall embedding for each user from his/her behavior sequence. However, a unified user embedding is hard to represent multiple interests. For example, in Figure 1, the click sequence shows three different interests of Emma. As a modern girl, Emma is interested in jewelry, handbags, and make-ups. Therefore, she may click items of the three categories during this period of time.

In this paper, we propose a novel controllable multi-interest framework, called ComiRec. Our multi-interest module can capture the multiple interests of users, which can be exploited for retrieving candidate items. Our aggregation module combines these items from different interests and outputs the overall recommendation. Figure 1 shows a motivating example of our multi-interest framework. We conduct experiments for the sequential recommendation, which is similar to our online situation. The experimental results

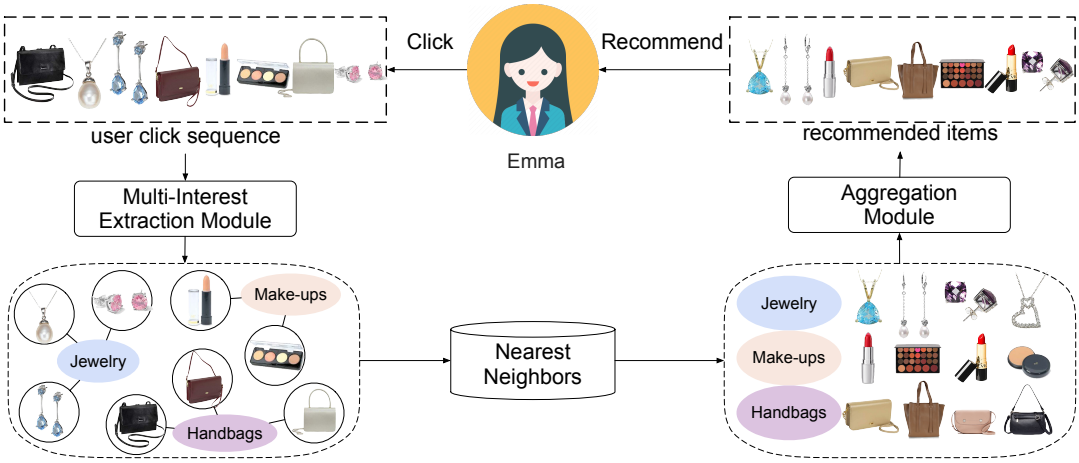

Figure 1: A motivating example of our proposed framework. An e-commerce platform user, Emma, has multiple interests including jewelry, handbags, and make-ups. Our multi-interest extraction module can capture these three interests from her click sequence. Each interest retrieves items from the large-scale item pool based on the interest embedding independently. An aggregation module combines items from different interests and outputs the overall top-N recommended items for Emma.

show that our framework outperforms other state-of-the-art models. Our framework has also been successfully deployed on the Alibaba distributed cloud platform. Results on the billion-scale industrial dataset further confirm the effectiveness and efficiency of our model in practice.

To summarize, the main contributions of this paper are:

- We propose a comprehensive framework that integrates the controllability and multi-interest components in a unified recommender system.
- We investigate the role of controllability on personalized systems by implementing and studying in an online recommendation scenario.
- Our framework achieves state-of-the-art performance on two real-world challenging datasets for the sequential recommendation.

Organization The rest of the paper is organized as follows. Section 2 summarizes related work. Section 3 formulates the sequential recommendation problem and introduces our proposed framework in detail. In Section 4, we conduct extensive experiments and case studies. Finally, we conclude in Section 5.

2 RELATED WORK

In this section, we introduce the related literature about recommender systems and recommendation diversity, as well as capsule networks and the attention mechanism we used in the paper.

Collaborative filtering [47, 48] methods have been proven successful in real-world recommender systems, which find similar users and items and make recommendations on this basis. Matrix factorizaion [30] is the most popular technique in classical recommender research, which maps both users and items to a joint latent factor space, such that user-item interactions are modeled as inner products in that space. Factorization Machines (FMs) [44] model all interactions between variables using factorized parameters and thus can estimate interactions even in problems with huge sparsity like recommender systems.

Neural Recommender Systems. Neural Collaborative Filtering (NCF) [20] uses a neural network architecture to model latent features of users and items. NFM [19] seamlessly combines the linearity of FMs in modeling second-order feature interactions and the non-linearity of neural networks in modeling higher-order feature interactions. DeepFM [14] designs an end-to-end learning model that emphasizes both low-order and high-order feature interactions for CTR prediction. xDeepFM [33] extends DeepFM and can learn specific bounded-degree feature interactions explicitly. Deep Matrix Factorization (DMF) [55] uses a deep structure learning architecture to learn a common low dimensional space for the representations of users and items based on explicit ratings and non-preference implicit feedback. DCN [53] keeps the benefits of a deep model and introduces a novel cross network that is more efficient in learning specific bounded-degree feature interactions. CMN [12] uses deep architecture to unify the two classes of CF models capitalizing on the strengths of the global structure of the latent factor model and local neighborhood-based structure in a nonlinear fashion.

Sequential Recommendation. The sequential recommendation is the crucial problem of recommender systems. Many recent works about recommender systems focus on this problem. FPMC [45] subsumes both a common Markov chain and the normal matrix factorization model for sequential basket data. HRM [52] extends the FPMC model and employs a two-layer structure to construct a hybrid representation over users and items from the last transaction. GRU4Rec [21] first introduces an RNN-based approach to model the whole session for more accurate recommendations. DREAM [57], based on Recurrent Neural Network (RNN), learns a dynamic representation of a user for revealing the user's dynamic interests. Fossil [17] integrates similarity-based methods with Markov Chains smoothly to make personalized sequential predictions on sparse

and long-tailed datasets. TransRec [16] embeds items into a vector space where users are modeled as vectors operating on item sequences for large-scale sequential prediction. RUM [7] uses a memory-augmented neural network integrated with the insights of collaborative filtering for the recommendation. SASRec [27] uses a self-attention based sequential model to capture long-term semantics and uses an attention mechanism to make its predictions based on relatively few actions. DIN [60] designs a local activation unit to adaptively learn the representation of user interests from past behaviors with respect to a certain ad. SDM [36] encodes behavior sequences with a multi-head self-attention module to capture multiple types of interests and a long-short term gated fusion module to incorporate long-term preferences.

Recommendation Diversity. Researchers have realized that following only the most accurate way of recommendation may not result in the best recommendation results, since the highest accuracy results tend to recommend similar items to users, yielding boring recommendation results [41]. To address such problems, the diversity of the recommended items also plays a significant role [49]. In terms of diversity, there is aggregated diversity [1], which refers to the ability to recommend "long-tail items" to users. Many studies focus on improving aggregated diversity of recommendation systems [1, 2, 40, 43]. Other works focus on the diversity of items recommended to individual users, i.e., the individual diversity [1, 11, 26, 58], which refers to the dissimilarity of items recommended to an individual user.

Attention The originality of attention mechanism can be traced back to decades ago in fields of computer vision [5, 50]. However, its popularity in various fields in machine learning comes only in recent years. It is first introduced to machine translation by [3], and later becomes an outbreaking method as *tensor2tensor* [51]. BERT [10] leverages *tensor2tensor* and achieves giant success in natural language processing. The attention mechanism is also adapted to recommender systems [6, 59] and is rather useful on real-world recommendation tasks.

Capsule Network. The concept of "capsules" is first proposed by [22] and has become well-known since the dynamic routing method [46] is proposed. MIND [31] introduces capsules into recommendation areas and uses the capsule network to capture multiple interests of e-commerce users based on dynamic routing mechanism, which is applicable for clustering past behaviors and extracting diverse interests. CARP [32] firstly extracts the viewpoints and aspects from the user and item review documents and derives the representation of each logic unit based on its constituent viewpoint and aspect for rating prediction.

3 METHODOLOGY

In this section, we formulate the problem and introduce the proposed framework in detail, as well as showing the difference between our framework and representative existing methods.

3.1 Problem Formulation

Assume we have a set of users $u \in \mathcal{U}$ and a set of items $i \in \mathcal{I}$. For each user, we have a sequence of user historical behaviors $(e_1^{(u)}, e_2^{(u)}, \cdots, e_n^{(u)})$, sorted by time of the occurance. $e_t^{(u)}$ records the t^{th} item interacted by the user. Given historical interactions, the problem of *sequential recommendation* is to predict the next items that the user might be interacted with. Notations are summarized in Table 1.

Table 1: Notations.

Notation	Description
u	a user
i	an item
e	an interaction
$\mathcal{U}$	the set of users
$\mathcal{I}$	the set of items
$\mathcal{I}_u$	the set of testing items of user u
d	the dimension of user/item embeddings
K	the number of interest embeddings
N	the number of candidate items
$\mathbf{V}_u$	the matrix of interest embeddings of user u
$\delta(\cdot)$	indicator function

In practice, due to the strict requirements of latency and performance, industrial recommender systems usually consist of two stages, the matching stage and the ranking stage. The matching stage corresponds to retrieving top-N candidate items, while the ranking stage is used for sorting the candidate items by more precise scores. Our paper mainly focuses on improving the effectiveness in the matching stage. In the following parts of this section, we will introduce our controllable multi-interest framework and illustrate the significance of our framework for the *sequential recommendation* problem.

3.2 Multi-Interest Framework

As the item pools of industrial recommender systems usually consist of millions or even billions of items, the matching stage plays a crucial role in recommender systems. Specifically, the matching model first computes user embeddings from user historical behaviors and then retrieves a candidate set of items for each user based on the user embedding. With the help of fast K nearest neighbors (KNN) algorithm to select the closest items from the large-scale item pool to generate a candidate set for each user, we mainly focus on the computation of user embeddings. In other words, the decisive factor for the matching stage is the quality of user embeddings computed from user historical behaviors.

Existing matching models usually use RNN[21, 54] to compute embeddings for users, but most of them only generate a single embedding vector for each user. This suffers from the lack of expressiveness of a single embedding since real-world customers usually have several kinds of items in their minds and these items are often for different uses and vary a lot in categories. Such behaviors of real-world customers highlight the need to use multiple vectors to represent their multiple interests. Based on the observations, we propose a multi-interest framework for the sequential recommendation. The input of our framework is a user behavior sequence, which contains a list of item IDs representing the user's interactions with items in time order. The item IDs are fed into an

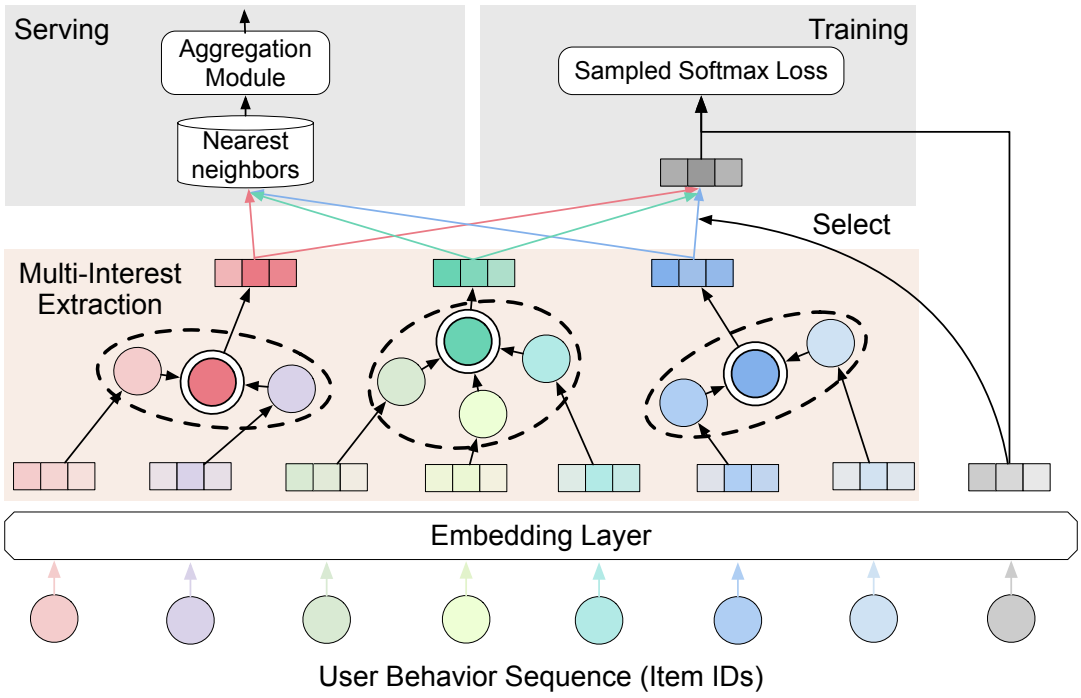

Figure 2: An overview of our model for the sequential recommendation. The input of our model is a user behavior sequence, which contains a list of item IDs. The item IDs are fed into the embedding layer and transformed into the item embeddings. Interest embeddings are generated through the multi-interest extraction module and can be then used for model training and serving. For model training, the nearest interest embedding to the target embedding will be chosen to compute the sampled softmax loss. For serving, each interest embedding will independently retrieve top-N nearest items, which are then fed into the aggregation module. The aggregation module generates the overall top-N items by a controllable procedure that balances the recommendation accuracy and diversity.

embedding layer and transformed into item embeddings. A multi-interest extraction module receives item embeddings and generates multiple interests for each user.

To build a multi-interest extraction module, there are many optional methods. In this paper, we explore two methods, dynamic routing method and self-attentive method, as our multi-interest extraction module. Our framework using a dynamic routing method or self-attentive method is named as ComiRec-DR or ComiRec-SA, respectively.

Dynamic Routing. We utilize a dynamic routing method as a multi-interest extraction module for user behavior sequences. The item embeddings of the user sequence can be viewed as primary capsules, and the multiple user interests can be seen as interest capsules. We use the dynamic routing method from CapsNet [46]. We briefly introduce dynamic routing for computing vector inputs and outputs of capsules. A capsule is a group of neurons whose activity vectors represent the instantiation parameters of a specific type of entity such as an object or an object part [46]. The length of the output vector of a capsule represents the probability that the entity represented by the capsule is in the current input. Let $\mathbf{e}_i$ be the capsule i of the primary layer. We then give the computation of the capsule j of the next layer based on primary capsules. We first compute the prediction vector as

$$\hat{\mathbf{e}}_{j|i} = \mathbf{W}_{ij}\mathbf{e}_i, \tag{1}$$

where $\mathbf{W}_{ij}$ is a transformation matrix. Then the total input to the capsule j is the weighted sum over all prediction vectors $\hat{\mathbf{e}}_{j|i}$ as

$$\mathbf{s}_j = \sum_i c_{ij}\hat{\mathbf{e}}_{j|i}, \tag{2}$$

where c_{ij} are the coupling coefficients that are determined by the iterative dynamic routing process. The coupling coefficients between capsule i and all the capsules in the next layer should sum to 1. We use "routing softmax" to calculate the coupling coefficients using initial logits b_{ij} as

$$c_{ij} = \frac{\exp(b_{ij})}{\sum_k \exp(b_{ik})}, \tag{3}$$

where b_{ij} represents the log prior probability that capsule i should be coupled to capsule j. A non-linear "squashing" function [46] is proposed to ensure short vectors to get shrunk to almost zero length and long vectors to get shrunk to a length slightly below 1. Then the vector of of capsule j is computed by

$$\mathbf{v}_j = \text{squash}(\mathbf{s}_j) = \frac{\|\mathbf{s}_j\|^2}{1+\|\mathbf{s}_j\|^2}\frac{\mathbf{s}_j}{\|\mathbf{s}_j\|}, \tag{4}$$

Algorithm 1: Dynamic Routing

Input: primary capsules $\mathbf{e}_i$, iteration times r, number of interest capsules K
Output: interest capsules $\{\mathbf{v}_j, j = 1, ..., K\}$

for each primary capsule i and interest capsule j: initialize $b_{ij} = 0$.
for $iter = 1, \cdots, r$ **do**
  for each primary capsule i: $\mathbf{c}_i = \text{softmax}(\mathbf{b}_i)$.
  for each interest capsule j: $\mathbf{s}_j = \sum_i c_{ij}\mathbf{W}_{ij}\mathbf{e}_i$.
  for each interest capsule j: $\mathbf{v}_j = \text{squash}(\mathbf{s}_j)$.
  for each primary capsule i and interest capsule j: $b_{ij} = b_{ij} + \mathbf{v}_j^\top \mathbf{W}_{ij}\mathbf{e}_i$.
return $\{\mathbf{v}_j, j = 1, ..., K\}$

where $\mathbf{s}_j$ is the total input of capsule j. To calculate the output capsules $\mathbf{v}_j$, we need to calculate the probability distribution based on the inner production of $\mathbf{v}_j$ and $\mathbf{e}_i$. The calculation of $\mathbf{v}_j$ relies on itself; thus, dynamic routing method is proposed to solve this problem. The whole dynamic routing process is listed in Algorithm 1. The output interest capsules of the user u are then formed as a matrix $\mathbf{V}_u = [\mathbf{v}_1, ..., \mathbf{v}_K] \in \mathbb{R}^{d \times K}$ for downstream tasks.

Self-Attentive Method. The self-attentive method [35] can also be applied to our multi-interest extraction module. Given the embeddings of user behaviors, $\mathbf{H} \in \mathbb{R}^{d \times n}$, where n is the length of the user sequence, we use the self-attention mechanism to obtain a vector of weights $\mathbf{a} \in \mathbb{R}^n$:

$$\mathbf{a} = \text{softmax}(\mathbf{w}_2^\top \tanh(\mathbf{W}_1\mathbf{H}))^\top, \tag{5}$$

where $\mathbf{w}_2$ and $\mathbf{W}_1$ are trainable parameters with size d_a and $d_a \times d$, respectively. The superscript $\top$ denotes the transpose of the vector or the matrix. The vector $\mathbf{a}$ with size n represents the attention weight of user behaviors. When we sum up the embeddings of user behaviors according to the attention weight, we can obtain a vector representation $\mathbf{v}_u = \mathbf{H}\mathbf{a}$ for the user. For the self-attentive method to make use of the order of user sequences, we add trainable positional embeddings [51] to the input embeddings. The positional embeddings have the same dimension d as the item embeddings and the two can be directly summed.

This vector representation focuses on and reflects a specific interest of the user u. To represent the overall interests of the user, we need multiple $\mathbf{v}_u$ from the user behaviors that focus on different interests. Thus we need to perform multiple times of attention. We extend the $\mathbf{w}_2$ into a d_a-by-K matrix as $\mathbf{W}_2$. Then the attention vector $\mathbf{a}$ becomes an attention matrix $\mathbf{A}$ as

$$\mathbf{A} = \text{softmax}(\mathbf{W}_2^\top \tanh(\mathbf{W}_1\mathbf{H}))^\top. \tag{6}$$

The final matrix of user interests $\mathbf{V}_u$ can be computed by

$$\mathbf{V}_u = \mathbf{H}\mathbf{A}. \tag{7}$$

Model Training. After computing the interest embeddings from user behaviors through the multi-interest extraction module, we use an *argmax* operator to choose a corresponding user embedding vector for a target item i:

Algorithm 2: Greedy Inference

Input: Candidate item set $\mathcal{M}$, number of output items N
Output: Output item set $\mathcal{S}$

$\mathcal{S} = \emptyset$
for $iter = 1, \cdots, N$ **do**
  $j = \text{argmax}_{i \in \mathcal{M} \setminus \mathcal{S}} \left(f(u, i) + \lambda \sum_{k \in \mathcal{S}} g(i, k) \right)$
  $\mathcal{S} = \mathcal{S} \cup \{j\}$
return $\mathcal{S}$

$$\mathbf{v}_u = \mathbf{V}_u[:, \text{argmax}(\mathbf{V}_u^\top \mathbf{e}_i)], \tag{8}$$

where $\mathbf{e}_i$ denotes the embedding of the target item i, and $\mathbf{V}_u$ is the matrix formed by user interest embeddings.

Given a training sample (u, i) with the user embedding $\mathbf{v}_u$ and the item embedding $\mathbf{e}_i$, we can compute the likelihood of the user u interacting with the item i as

$$P_\theta(i|u) = \frac{\exp(\mathbf{v}_u^\top \mathbf{e}_i)}{\sum_{k \in \mathcal{I}} \exp(\mathbf{v}_u^\top \mathbf{e}_k)}. \tag{9}$$

The objective function of our model is to minimize the following negative log-likelihood

$$loss = \sum_{u \in \mathcal{U}} \sum_{i \in \mathcal{I}_u} -\log P_\theta(i|u). \tag{10}$$

The sum operator of equation (9) is computationally expensive; thus, we use a sampled softmax technique [9, 24] to train our model.

Online Serving. For online serving, we use our multi-interest extraction module to compute multiple interests for each user. Each interest vector of a user can independently retrieve top-N items from the large-scale item pool by the nearest neighbor library such as Faiss [25]. The items retrieved by multiple interests are fed into an aggregation module to determine the overall item candidates. Finally, the items with higher ranking scores will be recommended for users.

3.3 Aggregation Module

After the multi-interest extraction module, we obtain multiple interest embeddings for each user based on his/her past behavior. Each interest embedding can independently retrieve top-N items based on the inner production proximity. But how to aggregate these items from different interests to obtain the overall top-N items? A basic and straightforward way is to merge and filter the items based on their inner production proximity with user interests, which can be formalized as

$$f(u, i) = \max_{1 <= k <= K} (\mathbf{e}_i^\top \mathbf{v}_u^{(k)}), \tag{11}$$

where $\mathbf{v}_u^{(k)}$ is the k-th interest embedding of the user u. This is an effective method for the aggregation process to maximize the recommendation accuracy. However, it is not all about the accuracy of current recommender systems. People are more likely to be recommended with something new or something diverse. The problem can be formulated in the following. Given a set $\mathcal{M}$ with $K \cdot N$ items retrieved from K interests of a user u, find a set $\mathcal{S}$ with

Table 2: Statistics of datasets.

Dataset	# users	# items	# interactions
Amazon Books	459,133	313,966	8,898,041
Taobao	976,779	1,708,530	85,384,110

N items such that a pre-defined value function is maximized. Our framework uses a controllable procedure to solve this problem. We use the following value function $Q(u, S)$ to balance the accuracy and diversity of the recommendation by a controllable factor $\lambda \geq 0$,

$$Q(u, \mathcal{S}) = \sum_{i \in \mathcal{S}} f(u, i) + \lambda \sum_{i \in \mathcal{S}} \sum_{j \in \mathcal{S}} g(i, j). \tag{12}$$

Here $g(i, j)$ is a diversity or dissimilarity function such as

$$g(i, j) = \delta(\text{CATE}(i) \neq \text{CATE}(j)). \tag{13}$$

where CATE(i) means the category of item i and $\delta(\cdot)$ is an indicator function. For the most accurate case, i.e., $\lambda = 0$, we just use the above straightforward method to obtain the overall items. For the most diverse case, i.e., $\lambda = \infty$, the controllable module finds the most diverse items for users. We study the controllable factor in the Section 4.3. We propose a greedy inference algorithm to approximately maximize the value function $Q(u, S)$, which is listed in the Algorithm 2.

3.4 Connections with Existing Models

We make a comparison between our model and existing models.

MIMN. MIMN [42], a recent representative work for the ranking stage of recommendation, uses memory networks to capture user interests from long sequential behavior data. Both MIMN and our model target at the multiple interests of users. For very long sequential behaviors, a memory-based architecture may also be insufficient to capture the long-term interests of users. Compared with MIMN, our model utilizes the multi-interest extraction module to leverage multiple interests of users instead of a complicated memory network with memory utilization regularization and memory induction unit.

MIND. MIND [31], a recent representative work for the matching stage of recommendation, proposes a Behavior-to-Interest (B2I) dynamic routing for adaptively aggregating user's behaviors into interest representation vectors, which differs with our dynamic routing method. Compared with MIND, we follow the original dynamic routing method used by CapsNet [46]. Specifically, MIND uses fully shared transformation matrices, i.e., $\mathbf{W}_{ij} = \mathbf{W}$. In this situation, B2I dynamic routing ignores the item positions and considers the item sequence as an item set. However, the item positions are important for the sequential recommendation.

4 EXPERIMENTS

In this section, we experiment on the sequential recommendation to evaluate the performance of our framework compared with other state-of-the-art methods. Besides, we also report the experimental results of our framework on a billion-scale industrial dataset.

4.1 Experimental Setup

We evaluate the performance of all methods under strong generalization [34, 37, 38]: We split all users into training/validation/test sets by the proportion of 8:1:1. We train models using the entire click sequences of training users. To evaluate, we take the first 80% of the user behaviors from validation and test users to infer user embeddings from trained models and compute metrics by predicting the remaining 20% user behaviors. This setting is more difficult than weak generalization where the users' behavior sequences are used during both training and evaluation processes [34]. In detail, we adopt a common setting of training sequential recommendation models. Let the behavior sequence of user u be $(e_1^{(u)}, e_2^{(u)}, ..., e_k^{(u)}, ..., e_n^{(u)})$. Each training sample uses the first k behaviors of u to predict the $(k+1)$-th behavior, where $k = 1, 2, ..., (n-1)$.

Datasets. We conduct experiments on two challenging public datasets. The statistics of the two datasets are shown in Table 2.

- **Amazon**[2] consists of product reviews and metadata from Amazon [18, 39]. In our experiment, we use the *Books* category of the Amazon dataset. Each training sample is truncated at length 20.
- **Taobao**[3] collects user behaviors from Taobao's recommender systems [61]. In our experiment, we only use the click behaviors and sort the behaviors from one user by time. Each training sample is truncated at length 50.

Competitors. We compare our proposed models, ComiRec-SA and ComiRec-DR, with state-of-the-art models. In our experimental setting, models should give the prediction for the unseen users of validation and test sets. Thus factorization-based methods are inappropriate for this setting.

- **MostPopular** is a traditional recommendation method that recommends the most popular items to users.
- **YouTube DNN** [9] is one of the most successful deep learning models for industrial recommender systems.
- **GRU4Rec** [21] is the first work that introduces recurrent neural networks for the recommendation.
- **MIND** [31] is a recent state-of-the-art model related with our model. It designs a multi-interest extractor layer based on the capsule routing mechanism, which is applicable for clustering past behaviors and extracting diverse interests.

Implementation Notes. The code used by our experiments is implemented with TensorFlow[4] 1.14 in Python 3.6.

Parameter Configuration. The number of dimensions d for embeddings is set to 64. The number of samples for sampled softmax loss is set to 10. The number of maximum training iterations is set to 1 million. The number of interest embeddings for multi-interest models is set to 4. We use Adam optimizer [29] with learning rate $lr = 0.001$ for optimization.

Evaluation Metrics. We use the following metrics to evaluate the performance of our proposed model. We use three commonly used evaluation criteria in our experiments.

[2] http://jmcauley.ucsd.edu/data/amazon/
[3] https://tianchi.aliyun.com/dataset/dataDetail?dataId=649&userId=1
[4] https://www.tensorflow.org/

Table 3: Model performance on public datasets. Bolded numbers are the best performance of each column. All the numbers in the table are percentage numbers with '%' omitted.

	Amazon Books						Taobao					
	Metrics@20			Metrics@50			Metrics@20			Metrics@50		
	Recall	NDCG	Hit Rate	Recall	NDCG	Hit Rate	Recall	NDCG	Hit Rate	Recall	NDCG	Hit Rate
MostPopular	1.368	2.259	3.020	2.400	3.936	5.226	0.395	2.065	5.424	0.735	3.603	9.309
YouTube DNN	4.567	7.670	10.285	7.312	12.075	15.894	4.205	14.511	28.785	6.172	20.248	39.108
GRU4Rec	4.057	6.803	8.945	6.501	10.369	13.666	5.884	22.095	35.745	8.494	29.396	46.068
MIND	4.862	7.933	10.618	7.638	12.230	16.145	6.281	20.394	38.119	8.155	25.069	45.846
ComiRec-SA	**5.489**	8.991	11.402	**8.467**	**13.563**	17.202	**6.900**	**24.682**	41.549	9.462	31.278	51.064
ComiRec-DR	5.311	**9.185**	**12.005**	8.106	13.520	**17.583**	6.890	24.007	**41.746**	**9.818**	**31.365**	**52.418**

- **Recall**. We adopt per-user average instead of global average for better interpretability [7, 28].

$$\text{Recall@N} = \frac{1}{|\mathcal{U}|} \sum_{u \in \mathcal{U}} \frac{|\hat{\mathcal{I}}_{u,N} \cap \mathcal{I}_u|}{|\mathcal{I}_u|}, \tag{14}$$

  where $\hat{\mathcal{I}}_{u,N}$ denotes the set of top-N recommended items for user u and $\mathcal{I}_u$ is the set of testing items for user u.
- **Hit Rate**. Hit rate (HR) measures the percentage that recommended items contain at least one correct item interacted by the user, which has been widely used in previous works [7, 28].

$$\text{HR@N} = \frac{1}{|\mathcal{U}|} \sum_{u \in \mathcal{U}} \delta(|\hat{\mathcal{I}}_{u,N} \cap \mathcal{I}_u| > 0), \tag{15}$$

  where $\delta(\cdot)$ is the indicator function.
- **Normalized Discounted Cumulative Gain**. Normalized Discounted Cumulative Gain (NDCG) takes the positions of correct recommended items into consideration [23].

$$\text{NDCG@N} = \frac{1}{Z}\text{DCG@N} = \frac{1}{Z}\frac{1}{|\mathcal{U}|} \sum_{u \in \mathcal{U}} \sum_{k=1}^{N} \frac{\delta(\hat{i}_{u,k} \in \mathcal{I}_u)}{\log_2(k+1)}, \tag{16}$$

  where $\hat{i}_{u,k}$ denotes the k-th recommended item for the user u, and Z is a normalization constant denoting the ideal discounted cumulative gain (IDCG@N), which is the maximum possible value of DCG@N.

4.2 Quantitative Results

To make a fair comparison with other models, we set $\lambda = 0$ in our aggregation module. We give a detailed illustration of retrieving top-N items of our framework. For our framework, each interest of a user independently retrieves top-N candidate items. Thus, our model retrieves a total of $K \cdot N$ items for each user. We sort the items by the inner product of the item embedding and the corresponding interest embedding. After the sorting, top-N items from these $K \cdot N$ items are viewed as the final candidate items of our model. The way of retrieving candidate items is also applied to MIND. The model performance for the sequential recommendation is shown in Table 3. Our models outperform all state-of-the-art models by a wide margin on all the evaluation criteria. GRU4Rec obtains the best performance over other models that only output single embedding for each user. Compared with MIND, ComiRec-DR obtains better performance due to the difference of the dynamic routing method.

Table 4: Model performance of parameter sensitivity. All the numbers in the table are percentage numbers with '%' omitted.

	Amazon Books		Taobao	
Metric@50	Recall	NDCG	Recall	NDCG
ComiRec-SA (K=2)	8.835	**14.273**	9.935	32.873
ComiRec-SA (K=4)	8.467	13.563	9.462	31.278
ComiRec-SA (K=6)	**8.901**	14.167	9.378	31.020
ComiRec-SA (K=8)	8.547	13.631	9.493	31.196
ComiRec-DR (K=2)	7.081	12.068	9.293	30.735
ComiRec-DR (K=4)	8.106	13.520	9.818	31.365
ComiRec-DR (K=6)	7.904	13.219	10.836	**34.048**
ComiRec-DR (K=8)	7.760	12.900	**10.841**	33.895

Table 5: Model performance of Amazon dataset for the controllable study. All the numbers in the table are percentage numbers with '%' omitted.

	ComiRec-SA (K=4)		ComiRec-DR (K=4)	
Metric@50	Recall	Diversity	Recall	Diversity
$\lambda = 0.00$	**8.467**	23.237	**8.106**	19.036
$\lambda = 0.05$	8.347	38.808	7.931	42.915
$\lambda = 0.10$	8.229	46.731	7.850	46.258
$\lambda = 0.15$	8.142	51.135	7.820	46.912
$\lambda = 0.20$	8.086	53.671	7.783	47.581
$\lambda = 0.25$	8.034	**55.100**	7.764	**48.375**

ComiRec-SA shows the strong ability to capture user interests by the self-attention mechanism and gets comparable results with ComiRec-DR.

Parameter Sensitivity. We investigate the sensitivity of the number of interests K of our framework. Table 4 illustrates the performance of our framework when the hyperparameter K changes. Our two models show the different properties of this hyperparameter. For the Amazon dataset, ComiRec-SA obtains the better performance when $K = 2, 6$ and ComiRec-DR gets the best result when $K = 4$. For the Taobao dataset, ComiRec-DR gets better performance when K increases from 2 to 8 but ComiRec-SA obtains the best result when $K = 2$.

Table 6: Statistics of the industrial dataset

Dataset	# users	# items	# interactions
Industrial	145,606,322	22,554,170	4,322,505,616

4.3 Controllable Study

To obtain the final top-N candidate items for each user, we propose a novel module to aggregate the items retrieved by different interests of each user. In addition to aim at achieving high prediction accuracy for the recommendation, some studies suggest the need for diversified recommendations to avoid monotony and improve customers' experience [8, 13].

Recommendation diversity plays a more important role in current recommender systems. Many pieces of research target on improving the recommendation diversity [4, 43]. Our proposed aggregation module can control the balance of recommendation accuracy and diversity. We use the following definition of individual diversity based on item categories:

$$\text{Diversity@N} = \frac{\sum_{j=1}^{N}\sum_{k=j+1}^{N}\delta(\text{CATE}(\hat{i}_{u,j}) \neq \text{CATE}(\hat{i}_{u,k}))}{N \times (N-1)/2}, \tag{17}$$

where $\text{CATE}(i)$ is the category of item i, $\hat{i}_{u,j}$ denotes the j-th recommended item for the user u, and $\delta(\cdot)$ is an indicator function.

Table 5 shows the model performance of the Amazon dataset when we control the factor λ to balance the recommendation quality and diversity. From the table, recommendation diversity increases substantially and recall decreases slightly when the controllable factor λ increases. Our aggregation module can achieve the optimum trade-off between the accuracy and diversity by choosing an appropriate value for the hyperparameter λ.

4.4 Industrial Results

We further experiment on the industrial dataset collected by Mobile Taobao App on February 8th, 2020. The statistics of the industrial dataset are shown in the Table 6. The industrial dataset contains 22 million high-quality items, 145 million users, and 4 billion behaviors between them.

Our framework has been deployed on the Alibaba distributed cloud platform, where every two workers share an NVIDIA Tesla P100 GPU with 16GB memory. We split the users and use the click sequences of training users to train our model. To evaluate, we use our model to compute multiple interests for each user in the test set. Each interest vector of a user independently retrieves top-N items from the large-scale item pool by a fast nearest neighbor method. The items retrieved by different user interests are fed into our aggregation module. After this module, top-N items out of $K \cdot N$ items are the final candidate items and are used to compute the evaluation metric, recall@50.

We conduct an offline experiment between our framework and the state-of-the-art sequential recommendation method, MIND [31], which has shown significant improvement in the recommender system of Alibaba Group. The experimental result demonstrates that our ComiRec-SA and ComiRec-DR improve recall@50 by 1.39% and 8.65% compared with MIND, respectively.

Figure 3: A case study of an e-commerce user. We generate four interest embeddings from the click sequence of a random user by our model. We find that the four interests of the user are about sweets, gift boxes, phone cases, and accessories. We report those items in the click sequence that correspond to the four interests. The right part shows the items retrieved from the industrial item pool by four interest embeddings.

Case Study. From the Figure 3, we can see that our model learns four different interests of the user from her click sequence. It is worth noting that our model only uses item IDs for training and does not use the manually defined category information of items. Despite that, our model still can learn the item categories from user behavior sequences. Each interest learned by our model approximately corresponds to one specific category and can retrieve similar items of the same category from the large-scale industrial item pool.

5 CONCLUSION

In this paper, we propose a novel controllable multi-interest framework for the sequential recommendation. Our framework uses a multi-interest extraction module to generate multiple user interests and uses an aggregation module to obtain the overall top-N items. Experimental results demonstrate that our models can achieve significant improvements over start-of-the-art models on two challenging datasets. Our framework has also been successfully deployed on the Alibaba distributed cloud platform. Results on the billion-scale industrial dataset further confirm the effectiveness and efficiency of our framework in practice. Recommender systems start a new phase owing to the rapid development of deep learning. Traditional recommendation methods cannot meet the requirements of the industry. For the future, we plan to leverage memory networks to capture the evolving interests of users and introduce cognitive theory to make better user modeling.

REFERENCES

[1] Gediminas Adomavicius and YoungOk Kwon. 2011. Improving aggregate recommendation diversity using ranking-based techniques. *TKDE* 24, 5 (2011), 896–911.

[2] Sujoy Bag, Abhijeet Ghadge, and Manoj Kumar Tiwari. 2019. An integrated recommender system for improved accuracy and aggregate diversity. *Computers & Industrial Engineering* 130 (2019), 187–197.

[3] Dzmitry Bahdanau, Kyunghyun Cho, and Yoshua Bengio. 2014. Neural machine translation by jointly learning to align and translate. *arXiv preprint arXiv:1409.0473* (2014).

[4] Keith Bradley and Barry Smyth. 2001. Improving recommendation diversity. In *AICS'01*. Citeseer, 85–94.

[5] Peter J Burt. 1988. Attention mechanisms for vision in a dynamic world. In *ICPR'88*. IEEE, 977–987.

[6] Yukuo Cen, Xu Zou, Jianwei Zhang, Hongxia Yang, Jingren Zhou, and Jie Tang. 2019. Representation learning for attributed multiplex heterogeneous network. In *KDD'19*. 1358–1368.

[7] Xu Chen, Hongteng Xu, Yongfeng Zhang, Jiaxi Tang, Yixin Cao, Zheng Qin, and Hongyuan Zha. 2018. Sequential recommendation with user memory networks. In *WSDM'18*. ACM, 108–116.

[8] Peizhe Cheng, Shuaiqiang Wang, Jun Ma, Jiankai Sun, and Hui Xiong. 2017. Learning to recommend accurate and diverse items. In *WWW'17*. 183–192.

[9] Paul Covington, Jay Adams, and Emre Sargin. 2016. Deep neural networks for youtube recommendations. In *RecSys'16*. ACM, 191–198.

[10] Jacob Devlin, Ming-Wei Chang, Kenton Lee, and Kristina Toutanova. 2018. Bert: Pre-training of deep bidirectional transformers for language understanding. *arXiv preprint arXiv:1810.04805* (2018).

[11] Tommaso Di Noia, Vito Claudio Ostuni, Jessica Rosati, Paolo Tomeo, and Eugenio Di Sciascio. 2014. An analysis of users' propensity toward diversity in recommendations. In *RecSys'14*. 285–288.

[12] Travis Ebesu, Bin Shen, and Yi Fang. 2018. Collaborative memory network for recommendation systems. In *SIGIR'18*. ACM, 515–524.

[13] Anupriya Gogna and Angshul Majumdar. 2017. Balancing accuracy and diversity in recommendations using matrix completion framework. *Knowledge-Based Systems* 125 (2017), 83–95.

[14] Huifeng Guo, Ruiming Tang, Yunming Ye, Zhenguo Li, and Xiuqiang He. 2017. DeepFM: a factorization-machine based neural network for CTR prediction. In *IJCAI'17*.

[15] Will Hamilton, Zhitao Ying, and Jure Leskovec. 2017. Inductive representation learning on large graphs. In *NIPS'17*. 1024–1034.

[16] Ruining He, Wang-Cheng Kang, and Julian McAuley. 2017. Translation-based recommendation. In *RecSys'17*. ACM, 161–169.

[17] Ruining He and Julian McAuley. 2016. Fusing similarity models with markov chains for sparse sequential recommendation. In *ICDM'16*. IEEE, 191–200.

[18] Ruining He and Julian McAuley. 2016. Ups and downs: Modeling the visual evolution of fashion trends with one-class collaborative filtering. In *WWW'16*. International World Wide Web Conferences Steering Committee, 507–517.

[19] Xiangnan He and Tat-Seng Chua. 2017. Neural factorization machines for sparse predictive analytics. In *SIGIR'17*. ACM, 355–364.

[20] Xiangnan He, Lizi Liao, Hanwang Zhang, Liqiang Nie, Xia Hu, and Tat-Seng Chua. 2017. Neural collaborative filtering. In *WWW'17*. International World Wide Web Conferences Steering Committee, 173–182.

[21] Balázs Hidasi, Alexandros Karatzoglou, Linas Baltrunas, and Domonkos Tikk. 2016. Session-based recommendations with recurrent neural networks. In *ICLR'16*.

[22] Geoffrey E Hinton, Alex Krizhevsky, and Sida D Wang. 2011. Transforming auto-encoders. In *ICANN'11*. Springer, 44–51.

[23] Kalervo Järvelin and Jaana Kekäläinen. 2000. IR evaluation methods for retrieving highly relevant documents. In *SIGIR'00*. ACM, 41–48.

[24] Sébastien Jean, Kyunghyun Cho, Roland Memisevic, and Yoshua Bengio. 2015. On using very large target vocabulary for neural machine translation. *ACL'15*.

[25] Jeff Johnson, Matthijs Douze, and Hervé Jégou. 2017. Billion-scale similarity search with GPUs. *arXiv preprint arXiv:1702.08734* (2017).

[26] M Kalaivanan and K Vengatesan. 2013. Recommendation system based on statistical analysis of ranking from user. In *ICICES'13*. IEEE, 479–484.

[27] Wang-Cheng Kang and Julian McAuley. 2018. Self-attentive sequential recommendation. In *ICDM'18*. IEEE, 197–206.

[28] George Karypis. 2001. Evaluation of item-based top-n recommendation algorithms. In *CIKM'01*. ACM, 247–254.

[29] Diederik P Kingma and Jimmy Ba. 2014. Adam: A method for stochastic optimization. *arXiv preprint arXiv:1412.6980* (2014).

[30] Yehuda Koren, Robert Bell, and Chris Volinsky. 2009. Matrix factorization techniques for recommender systems. *Computer* 8 (2009), 30–37.

[31] Chao Li, Zhiyuan Liu, Mengmeng Wu, Yuchi Xu, Pipei Huang, Huan Zhao, Guoliang Kang, Qiwei Chen, Wei Li, and Dik Lun Lee. 2019. Multi-Interest Network with Dynamic Routing for Recommendation at Tmall. *arXiv preprint arXiv:1904.08030* (2019).

[32] Chenliang Li, Cong Quan, Li Peng, Yunwei Qi, Yuming Deng, and Libing Wu. 2019. A Capsule Network for Recommendation and Explaining What You Like and Dislike. In *SIGIR'19*. ACM, 275–284.

[33] Jianxun Lian, Xiaohuan Zhou, Fuzheng Zhang, Zhongxia Chen, Xing Xie, and Guangzhong Sun. 2018. xDeepFM: Combining explicit and implicit feature interactions for recommender systems. In *KDD'18*. ACM, 1754–1763.

[34] Dawen Liang, Rahul G Krishnan, Matthew D Hoffman, and Tony Jebara. 2018. Variational autoencoders for collaborative filtering. In *WWW'18*. 689–698.

[35] Zhouhan Lin, Minwei Feng, Cicero Nogueira dos Santos, Mo Yu, Bing Xiang, Bowen Zhou, and Yoshua Bengio. 2017. A structured self-attentive sentence embedding. In *ICLR'17*.

[36] Fuyu Lv, Taiwei Jin, Changlong Yu, Fei Sun, Quan Lin, Keping Yang, and Wilfred Ng. 2019. SDM: Sequential deep matching model for online large-scale recommender system. In *CIKM'19*. 2635–2643.

[37] Jianxin Ma, Chang Zhou, Peng Cui, Hongxia Yang, and Wenwu Zhu. 2019. Learning disentangled representations for recommendation. In *NIPS'19*. 5712–5723.

[38] Benjamin Marlin. 2004. *Collaborative filtering: A machine learning perspective*. University of Toronto Toronto.

[39] Julian McAuley, Christopher Targett, Qinfeng Shi, and Anton Van Den Hengel. 2015. Image-based recommendations on styles and substitutes. In *SIGIR'15*. ACM, 43–52.

[40] Katja Niemann and Martin Wolpers. 2013. A new collaborative filtering approach for increasing the aggregate diversity of recommender systems. In *KDD'13*. 955–963.

[41] Umberto Panniello, Alexander Tuzhilin, and Michele Gorgoglione. 2014. Comparing context-aware recommender systems in terms of accuracy and diversity. *UMUAI* 24, 1-2 (2014), 35–65.

[42] Qi Pi, Weijie Bian, Guorui Zhou, Xiaoqiang Zhu, and Kun Gai. 2019. Practice on long sequential user behavior modeling for click-through rate prediction. In *KDD'19*. 2671–2679.

[43] Lijing Qin and Xiaoyan Zhu. 2013. Promoting diversity in recommendation by entropy regularizer. In *IJCAI'13*.

[44] Steffen Rendle. 2010. Factorization machines. In *ICDM'10*. IEEE, 995–1000.

[45] Steffen Rendle, Christoph Freudenthaler, and Lars Schmidt-Thieme. 2010. Factorizing personalized markov chains for next-basket recommendation. In *WWW'10*. ACM, 811–820.

[46] Sara Sabour, Nicholas Frosst, and Geoffrey E Hinton. 2017. Dynamic routing between capsules. In *NIPS'17*. 3856–3866.

[47] Badrul Munir Sarwar, George Karypis, Joseph A Konstan, John Riedl, et al. 2001. Item-based collaborative filtering recommendation algorithms. *WWW'01* (2001), 285–295.

[48] J Ben Schafer, Dan Frankowski, Jon Herlocker, and Shilad Sen. 2007. Collaborative filtering recommender systems. In *The adaptive web*. Springer, 291–324.

[49] Malcolm Slaney and William White. 2006. Measuring playlist diversity for recommendation systems. In *AMCMM'06 workshop*. 77–82.

[50] Yaoru Sun and Robert Fisher. 2003. Object-based visual attention for computer vision. *Artificial intelligence* 146, 1 (2003), 77–123.

[51] Ashish Vaswani, Noam Shazeer, Niki Parmar, Jakob Uszkoreit, Llion Jones, Aidan N Gomez, Łukasz Kaiser, and Illia Polosukhin. 2017. Attention is all you need. In *NIPS'17*. 5998–6008.

[52] Pengfei Wang, Jiafeng Guo, Yanyan Lan, Jun Xu, Shengxian Wan, and Xueqi Cheng. 2015. Learning hierarchical representation model for nextbasket recommendation. In *SIGIR'15*. ACM, 403–412.

[53] Ruoxi Wang, Bin Fu, Gang Fu, and Mingliang Wang. 2017. Deep & cross network for ad click predictions. In *ADKDD'17*. ACM, 12.

[54] Chao-Yuan Wu, Amr Ahmed, Alex Beutel, Alexander J Smola, and How Jing. 2017. Recurrent recommender networks. In *WSDM'17*. ACM, 495–503.

[55] Hong-Jian Xue, Xinyu Dai, Jianbing Zhang, Shujian Huang, and Jiajun Chen. 2017. Deep Matrix Factorization Models for Recommender Systems.. In *IJCAI'17*. 3203–3209.

[56] Rex Ying, Ruining He, Kaifeng Chen, Pong Eksombatchai, William L Hamilton, and Jure Leskovec. 2018. Graph convolutional neural networks for web-scale recommender systems. In *KDD'18*. ACM, 974–983.

[57] Feng Yu, Qiang Liu, Shu Wu, Liang Wang, and Tieniu Tan. 2016. A dynamic recurrent model for next basket recommendation. In *SIGIR'16*. ACM, 729–732.

[58] Ting Yu, Junpeng Guo, Wenhua Li, Harry Jiannan Wang, and Ling Fan. 2019. Recommendation with diversity: An adaptive trust-aware model. *Decision Support Systems* 123 (2019), 113073.

[59] Chang Zhou, Jinze Bai, Junshuai Song, Xiaofei Liu, Zhengchao Zhao, Xiusi Chen, and Jun Gao. 2018. Atrank: An attention-based user behavior modeling framework for recommendation. In *AAAI'18*.

[60] Guorui Zhou, Xiaoqiang Zhu, Chenru Song, Ying Fan, Han Zhu, Xiao Ma, Yanghui Yan, Junqi Jin, Han Li, and Kun Gai. 2018. Deep interest network for click-through rate prediction. In *KDD'18*. ACM, 1059–1068.

[61] Han Zhu, Xiang Li, Pengye Zhang, Guozheng Li, Jie He, Han Li, and Kun Gai. 2018. Learning tree-based deep model for recommender systems. In *KDD'18*. ACM, 1079–1088.

A APPENDIX

In the appendix, we give the implementation notes of our proposed models. The details of other models and descriptions of datasets are then given.

A.1 Implementation Notes

Running Environment. The experiments in this paper can be divided into two parts. One is conducted on two public datasets using a single Linux server with 4 Intel(R) Xeon(R) CPU E5-2680 v4 @ 2.40GHz, 256G RAM, and 8 NVIDIA GeForce RTX 2080 Ti. The codes of our proposed models in this part are implemented with TensorFlow[5] 1.14 in Python 3.6. The other part is conducted on the industrial dataset using Alibaba's distributed cloud platform[6] which contains thousands of workers. Every two workers share an NVIDIA Tesla P100 GPU with 16GB memory. Our proposed models are implemented with TensorFlow 1.4 in Python 2.7 in this part.

Implementation Details. Our codes used by a single Linux server can be split into three parts: data iterator, model training, and evaluation. For each training iteration, the data iterator selects random training users with a size of *batch_size*. For each selected user, we randomly select an item in his/her click sequence as the training label and use the items before that item as the training sequence. The training part is implemented following the training loop in the Algorithm 3 based on the Tensorflow 1.x APIs. Our loss function is based on *tf.nn.sampled_softmax_loss*. The evaluation part replies on Faiss[7], a library for efficient similarity search and clustering of dense vectors. We use the *GpuIndexFlatIP* class of Faiss, which implements an exact search for the inner product on GPU. All model parameters are updated and optimized by stochastic gradient descent with Adam updating rule [29]. The distributed version of our proposed models is implemented based on the coding rules of Alibaba's distributed cloud platform in order to maximize the distribution efficiency.

Parameter Configuration. Our user/item embedding dimension d is set to 64. The number of samples for sampled softmax loss is set to 10. The number of maximum training iterations is set to 1 million and all models use early stopping based on the Recall@50 on the validation set. The batch size for the Amazon dataset and Taobao dataset is set to 128 and 256, respectively. The number of iterations for the dynamic routing method is set to 3. The number of interest embeddings K for multi-interest models is set to 4 for a fair comparison. We use the Adam optimizer [29] with learning rate $lr = 0.001$ for optimization.

Code and Dataset Releasing Details. The code of all models and our partition of the two public datasets are available[8].

A.2 Compared Methods

We give the implementation details about all compared methods as follows.

[5]https://www.tensorflow.org/
[6]https://data.aliyun.com/
[7]https://github.com/facebookresearch/faiss
[8]https://github.com/cenyk1230/ComiRec

Algorithm 3: ComiRec

Input: User behavior sequences.
Initialize all the model parameters.
Generate training samples $\{(u, i)\}$ with user click sequences.
while *not converged* **do**
  for *each batch from training samples* **do**
    Compute $\mathbf{V}_u$ using multi-interest extraction module.
    Compute $\mathbf{v}_u$ based on Equation (8).
    Compute sampled softmax loss using Equation (10).
    Update model parameters by the Adam optimizer.

- **MostPopular** is a non-personalized method that recommends the most popular items to users. This method does not need training and we implement it separately.
- **YouTube DNN** is one of the most successful deep learning models for industrial recommender systems. We implement the model in our code based on the original paper.
- **GRU4REC** is the first work that introduces recurrent neural networks for the recommendation. We implement the model by *tf.nn.rnn_cell.GRUCell* and *tf.nn.dynamic_rnn* of TensorFlow in our code.
- **MIND** is a recent state-of-the-art model. We implement the model based on the original paper and an internal version of the code in Alibaba Group.

A.3 Datasets

Our experiments evaluate on three datasets, including two public datasets and a billion-scale industrial dataset. For the two public datasets, we keep users and items with at least 5 behaviors.

- **Amazon**[9] consists of product reviews and metadata from Amazon [18, 39]. In our experiment, we use the *Books* category of the Amazon dataset. For each user u, we sort the reviews from the user by time, and our task is to predict whether the user will write the review for the item based on previous reviews. Each training sample is truncated at length 20.
- **Taobao**[10] collects user behaviors from Taobao's recommender systems [61]. Taobao dataset randomly selects about 1 million users who have behaviors including click, purchase, add-to-cart, and add-to-preference from November 25 to December 03, 2017. Each behavior is represented by five fields, which consist of user ID, item ID, item's category ID, behavior type, and timestamp. In our experiment, we only use the click behaviors and sort the behaviors from one user by time. Each training sample is truncated at length 50.
- **Industrial dataset** collects user click behaviors by Mobile Taobao App on February 8th, 2020. The industrial dataset contains 22 million high-quality items, 145 million users, and 4 billion behaviors between them. Each training sample is truncated at length 40.

[9]http://jmcauley.ucsd.edu/data/amazon/
[10]https://tianchi.aliyun.com/dataset/dataDetail?dataId=649&userId=1