Everyone's Preference Changes Differently: A Weighted Multi-Interest Model for Retrieval

Hui Shi [1] **Yupeng Gu** [2] **Yitong Zhou** [2] **Bo Zhao** [2] **Sicun Gao** [1] **Jishen Zhao** [1]

Abstract

User embeddings (vectorized representations of a user) are essential in recommendation systems. Numerous approaches have been proposed to construct a representation for the user in order to find similar items for retrieval tasks, and they have been proven effective in industrial recommendation systems. Recently people have discovered the power of using multiple embeddings to represent a user, with the hope that each embedding represents the user's interest in a certain topic. With multi-interest representation, it's important to model the user's preference over the different topics and how the preference changes with time. However, existing approaches either fail to estimate the user's affinity to each interest or unreasonably assume every interest of every user fades at an equal rate with time, thus hurting the performance of candidate retrieval. In this paper, we propose the Multi-Interest Preference (MIP) model, an approach that not only produces multi-interest for users by using the user's sequential engagement more effectively but also automatically learns a set of weights to represent the preference over each embedding so that the candidates can be retrieved from each interest proportionally. Extensive experiments have been done on various industrial-scale datasets to demonstrate the effectiveness of our approach. [1]

1. Introduction

Today, the recommendation system is widely used in online platforms to help users discover relevant items and deliver a positive user experience. In industrial recommendation systems, there are usually billions of entries in the item catalog, which makes it impossible to calculate the similarity between a user and every item. The common approach is, illustrated in Figure 1, retrieving only hundreds or thousands of candidate items based on their similarity to the user embedding on an approximate level (*e.g.* inverted indexes, locality-sensitive hashing) without consuming too much computational power, and then sending the retrieved candidates to the more nuanced ranking models. Thus, finding effective user embedding is fundamental to the recommendation quality.

The user representations learned from the neural networks are proven to work well on large-scale online platforms, such as Google (Cheng et al., 2016), YouTube (Covington et al., 2016), and Alibaba (Wang et al., 2018). Mostly, the user embeddings are learned by aggregating the item embeddings from the user engagement history, via sequential models (Hidasi et al., 2015; Quadrana et al., 2017; Kang & McAuley, 2018; You et al., 2019). These works usually rely on the sequential model, *e.g.* a Recurrent Neural Network (RNN) model or an attention mechanism, to produce a single embedding that summarizes the user's one or more interests from recent and former actions.

Recently researchers (Epasto & Perozzi, 2019; Weston et al., 2013; Pal et al., 2020; Li et al., 2005) have discovered the importance of having multiple embeddings for an individual, especially in the retrieval phase, with the hope that they can capture a user's multiple interests. The intuition is quite clear: if multiple interests of a user are collapsed into a single embedding, though this embedding could be similar to and can be decoded to all the true interests of the user, directly using the single collapsed embedding to retrieve the closest items might result in items that the user is not quite interested in, as illustrated in Figure 1.

Though, conventional sequential models like RNN or the Transformer network do not naturally produce multiple sequence-level embeddings as desired in the multi-interest

[1]Department of Computer Science and Engineering, University of California San Diego, La Jolla, United States, *{hshi, jzhao, sicung}@ucsd.edu* [2]Pinterest, San Francisco, United States, {yupeng, yzhou, bozhao}@pinterest.com. Correspondence to: Hui Shi <hshi@ucsd.edu>.

Proceedings of the 40^{th} International Conference on Machine Learning, Honolulu, Hawaii, USA. PMLR 202, 2023.

[1]The code is available at: https://github.com/shihui2010/MIP

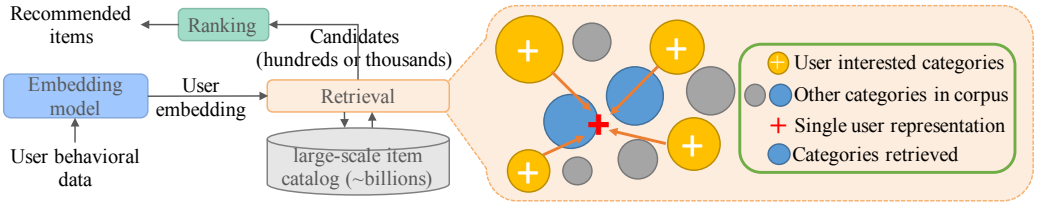


Figure 1. Mis-representation with single user embedding in the retrieve-then-rank framework.

user representation. Existing solutions fall into two directions: 1) **split-by-cluster** approaches first cluster the items in the user engagement history by category labels (Li et al., 2019) or item embedding vectors (Pal et al., 2020) and then compute a representation embedding per cluster; 2) **split-by-attention** models adopt transformer-like architecture with two modifications. The query vectors in the attention are learnable vectors instead of the projections from the input and the results of each attention head are directly taken as multiple embeddings (Zhuang et al., 2020; Cen et al., 2020). The limitations of the two approaches are obvious: the split-by-cluster method works best with dense item feature (Xue et al., 2005); and split-by-attention models bias towards the popular categories owing to its shared query vector among all the users and are inflexible to adjust the number of interests, which is fixed in the training phase as the number of attention heads.

Moreover, the existing multi-interest works ignore one important aspect: the weights of each embedding. In the retrieval stage, given the limited number of items to return, retrieving items from each embedding uniformly will cause a recall problem when the user clearly indicates a high affinity towards one or two categories. Some existing approaches, *e.g.* PinnerSage (Pal et al., 2020), use exponentially decayed weights to assign a higher score to interests that have more frequent and recent engagements. However, the methods still assume that in the same period, regardless of whether the interest is enduring or ephemeral, the level of interest decays equally for any user. Furthermore, these works also assume the number of embeddings to be fixed across all users. Not only is this hyperparameter costly to find, but also the assumption that all users have the same number of interests is questionable. Some dormant users can be well represented using one or two vectors, while others might have a far more diverse set of niche interests that requires tens of embeddings to represent.

In this paper, we propose Multi-Interest Preference (MIP) model that learns user embeddings on multiple interest dimensions with personalized and context-aware interest weights. The MIP model consists of a clustering-enhanced multi-head attention module to compute multiple interests and a feed-forward network to predict the weights for each embedding from the interest embedding as well as the temporal patterns of the interest. The clustering-enhanced attention overcomes the aforementioned shortcomings from two aspects: the query, key, and value vectors are projected from the user's engaged items, thus the output of the attention is personalized and minimized the bias toward globally popular categories; moreover, the clustering module can be applied before or after the multi-head attention, releasing the assumption that item features are pre-computed or the item-category labels are available. The main contribution of this paper and the experimental evidence can be summarized as follows:

- We propose a multi-interest user representation model that minimizes the bias towards popular categories and is applicable no matter if the item embeddings are pre-computed. MIP is successful in various industry-scale datasets (Section 4.1, 4.2); Appendix A.1 reveals the bias resulting from the global query vector and the error resulting from a fixed number of clusters in the split-by-attention approaches, in comparison to MIP.
- In addition to the multi-facet vector representations of a user, MIP assigns weights to each embedding, which are automatically customized for each user interest. This approach improves the recall of candidate generation by retrieving more candidates from the most representative embedding. (Section 4.3).
- Although if the clustering algorithms require, MIP still asks for a number of clusters during the training phase, the number of clusters in MIP in the inference phase can be trivially increased or decreased without re-training of the model. And the experimental results (Appendix A.2) show that re-configuring the number of clusters has an insignificant impact on the retrieval performance, thus allowing the system to trade off the storage and computation cost for better performance. Thus, MIP does not require prior knowledge of the number of interests of users during the model training phase.

2. Related Work

This work relates to two important aspects of existing recommendation systems: sequential models and the multi-interest framework.

Sequential models. A basic consensus in the recommendation system is that user embeddings should be inferred from

Name	User Embedding	Sequential Model	Additional Input	Preference Weight
GRU4Rec (Hidasi et al., 2015)	single	RNN	interaction session	N/A
TiSASRec (Li et al., 2020)	single	time-aware self-attention	timestamps	N/A
BERT4Rec (Sun et al., 2019)	single	self-attention	–	N/A
MIND(Li et al., 2019)	multiple	label-aware self-attention	category labels	✗
ComiRec (Cen et al., 2020)	multiple	global-query attention	–	✗
PinText2 (Zhuang et al., 2020)	multiple	shared global-query attention	–	✗
PinnerSage (Pal et al., 2020)	multiple	N/A	–	heuristic
MIP	multiple	time-aware self-attention	timestamps	learned

Table 1. Comparison of MIP to existing recommendation models.

the user's historical behavior, and thus the sequential models have been at the heart of recommendation models. A typical and classical sequential model is the Markov Chain (Rendle et al., 2010; He & McAuley, 2016). While Markov Chain captures short-term patterns of engagement sequence well, it fails to make the recommendation that requires memorizing long sequences. With stronger representation power on long sequences, Recurrent Neural Networks (RNNs) have been adopted for learning user embedding from arbitrarily long sequences, *e.g.* GRU4Rec (Hidasi et al., 2015) and others (Xu et al., 2019; Devooght & Bersini, 2017). Besides the standard RNN models, specialized recurrent units are proposed to meet the special need of incorporating certain information, *e.g.* user demographic information (Donkers et al., 2017), global context (Xia et al., 2017), interest drifts with time (Chen et al., 2019), and interaction session (Hidasi et al., 2015). Recently, the success of the Transformer network (Vaswani et al., 2017) has brought revolution to sequential modeling tasks(Shi et al., 2022; Pérez et al., 2019) and has been soon adapted to the recommendation models, *e.g.* ComiRec (Cen et al., 2020), BERT4Rec (Sun et al., 2019), TiSASRec (Li et al., 2020), SASRec (Kang & McAuley, 2018), MIND (Li et al., 2019), PinText2 (Zhuang et al., 2020), and also our MIP.

Multi-interest user representation. Representing users by multiple embeddings greatly improves the recommendation quality, but not every existing recommendation model can easily extend to a multi-interest framework. Classical collaborative filtering and matrix factorization methods do not naturally produce multiple user embeddings, and so do RNNs and attention-based models. To discover multiple interests from user engagement history, heuristic methods (Jiang et al., 2020; Yue & Xiang, 2012) and unsupervised learning methods like clustering (Pal et al., 2020; Wandabwa et al., 2020) and community mining (Wang, 2007; Yu, 2008) have been adopted. Besides, researchers have made efforts to modify the existing neural networks to produce multiple results, for instance, the capsule network (Li et al., 2019; Sabour et al., 2017; Cen et al., 2020) and multi-head attention models (Li et al., 2020; Cen et al., 2020; Zhou et al., 2018). However, they require an estimation of the number of interests of users as a hyperparameter and do not learn the weight of interests. Therefore, unlike MIP, they produce an equal number of clusters for every user and treat each interest with uniform importance.

Relationship to previous works. The motivation of MIP is to acquire weighted multiple user embeddings with standard self-attention but without explicit item-category labels. ComiRec and PinText2 use global-query attention to produce multiple embeddings, which introduces a bias toward frequent items or popular categories and the phenomenon is shown in Appendix A.1. Furthermore, they also predefine a number of interests that is uniform for all the users. TiSASRec and BERT4Rec adopt self-attention but can not learn multiple embeddings. MIND relies on the category labels to produce multiple embeddings from self-attention and capsule networks. However the category labels are sometimes unavailable or vague in other applications, *e.g.* YouTube and Pinterest. PinnerSage produces multiple embeddings without category labels, but requires pre-computed item embeddings. The comparison are summarized in Table 1.

3. Methodology

In this section, we formulate the recommendation problem and the neural architecture to model the multiple user interests with preference weights in detail.

3.1. Problem Statement

Let $\mathcal{I}$ denote the collection of items and $\mathcal{U}$ denote the set of users. The interaction sequence of a user $u \in \mathcal{U}$ is represented as $\mathcal{S}^u$ with a list of item IDs $(v^u_{t_1}, v^u_{t_2}, ..., v^u_{t_{l_u}})$ and timestamps $(t^u_1, t^u_2, ..., t^u_{|l_u|})$. Each item $v^u_i \in \mathcal{I}$ is associated with an item embedding $\boldsymbol{p}^u_i \in \mathbb{R}^d$ and l_u is the length of the interaction sequence.

The objective is to learn a set of user embeddings $\boldsymbol{z}^u_\lambda \in \mathbb{R}^d$ and their weights w^u_λ $(\lambda = 1, ..., \Lambda)$ for each user u. Since the user representation is learned only from the history of

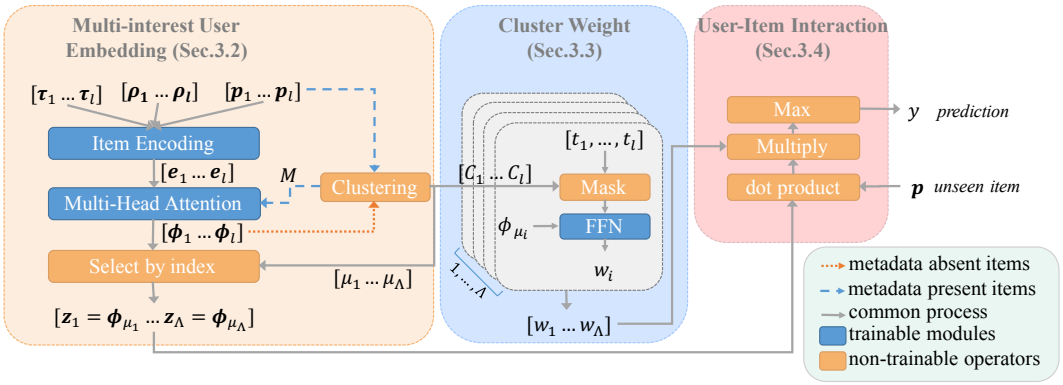


Figure 2. An overview of MIP architecture. The input is the user engagement history containing item embeddings $[\boldsymbol{p}_1 ... \boldsymbol{p}_l]$), temporal encoding $[\boldsymbol{\tau}_1 ... \boldsymbol{\tau}_l]$, and positional encoding ($[\boldsymbol{\rho}_1 ... \boldsymbol{\rho}_l]$). The multi-interest user embedding module produces Λ embeddings, where Λ is decided by the clustering method or as a hyperparameter. With clustering and multi-interest representation, the cluster weight module will then estimate the cluster weights for each cluster. Finally, the multi-interest embeddings with corresponding weights are combined to predict the user's interest in an unseen item $\boldsymbol{p}$. The processes with item metadata absent/present are shown with different arrows.

Notations	Description
$\mathcal{I}, \mathcal{U}$	Item set and user set
$\mathcal{S}$	Abbreviation of user engagement history $\mathcal{S}^u$
l	Abbreviation of l_u, length of $\mathcal{S}$
d	The item embedding dimension
d_{model}	Projected key/query vector dimension
h	Attention head superscript
$\boldsymbol{p}$	An item embedding, $\boldsymbol{p} \in \mathbb{R}^d$
$\boldsymbol{p}_j, t_j$	Short form of the $\boldsymbol{p}^u_{t_j}$ and t^u_j
$W^h_q, \boldsymbol{b}^h_q$	Query projection weights and bias
$W^h_k, \boldsymbol{b}^h_k$	Key projection weights and bias
$\boldsymbol{M}$	Attention mask matrix
Λ	Maximum number of interests per user
$\mathcal{C}$	Cluster assignment, $\mathcal{C} \in \mathbb{R}^l$. $\mathcal{C}_{[i]} = \lambda$ if i-th item belongs to λ-th cluster
$\boldsymbol{L}_\lambda$	The set of indices of items in the λ-th cluster.
$\boldsymbol{z}_\lambda, Z$	User embedding vector(s)
$\mathbb{1}_{[cond]}$	1 if $cond$ is true, 0 otherwise
$\boldsymbol{\tau}_{[m]}$	m-th digit in the vector $\boldsymbol{\tau}$
$[;]$	Vector concatenation operator

Table 2. Notations

that user, hereinafter, we omit the superscript for u in both the input and output sides for simplicity. Notations are summarized in Table 2.

Item embedding $\boldsymbol{p}_j$ can be either represented by item metadata features or treated as model parameters and learned from the data. When items have dense metadata features (*e.g.* text embeddings), we will use them as the item embedding and focus on learning the user representation. When items are only represented by an ID and do not have other metadata, we will learn the item embedding table. Our model is able to handle both cases and eventually learn users' multi-interest embeddings and their weights.

3.2. Multi-Interest User Modeling

Item Representation. For the j-th item in the user engagement history, we represent the item by $\boldsymbol{p}_j$ and it's either learned (when item metadata is absent) or copied from the item metadata feature (when item metadata is present). In addition to $\boldsymbol{p}_j$, the sequential order and relative timestamps of the interactions are represented by positional encoding and temporal encoding respectively as $\boldsymbol{\rho}$ and $\boldsymbol{\tau}$. Following Vaswani et al. (2017), the odd digits $(2m+1)$ and even digits $(2m)$ on the encoding vectors are given by:

$$\begin{aligned}
\boldsymbol{\tau}_{[2m]}(t_j) &= sin\big(t_j/(\tau_{max})^{2m/m_t}\big) \\
\boldsymbol{\tau}_{[2m+1]}(t_j) &= cos\big(t_j/(\tau_{max})^{2m/m_t}\big) \\
\boldsymbol{\rho}_{[2m]}(j) &= sin\big(j/(\rho_{max})^{2m/m_p}\big) \\
\boldsymbol{\rho}_{[2m+1]}(j) &= cos\big(j/(\rho_{max})^{2m/m_p}\big)
\end{aligned} \tag{1}$$

The hyper-parameters are set as $\tau_{max} = \rho_{max} = 1 \times 10^4$, and $m_t = m_p = 1$. The timestamps unit is *day*. In the Section 4.4, other encodings forms are compared, and show that the design used in Equation 1 has a slight advantage.

Finally, item ID embedding is concatenated with the positional and temporal encodings to produce the final representation of an interaction:

$$\mathbf{e}_j := [\boldsymbol{p}_j; \boldsymbol{\tau}(t_j); \boldsymbol{\rho}(j)] \tag{2}$$

User Representation. Our user representation is built upon the multi-head self-attention module (Vaswani et al., 2017). Using the interaction embedding as above, for each attention head h, we define the attention weight between interaction i and j as

$$a^h_{i,j} = \frac{\exp(s^h_{i,j})}{\sum_{k=1}^{l} \exp(s^h_{i,k})} \tag{3}$$

where $s_{i,j}^h$ is the dot product between the projected query of j and the projected key of i:

$$s_{i,j}^h = \left((W_q^h \boldsymbol{e}_j + \boldsymbol{b}_q^h)^\top \cdot (W_k^h \boldsymbol{e}_i + \boldsymbol{b}_k^h)\right) / \sqrt{d_{model}} \quad (4)$$

In order to build the user's multi-interest embedding, we need to cluster the items in the user sequence. When we use item metadata features as $\boldsymbol{p}$, we pre-compute the item clusters $L_1, \cdots, L_\Lambda$ where each L_λ is the set of item IDs that belong to λ-th cluster. Let $\mathcal{C}$ denote the mapping from item ID to cluster assignment, *i.e.* $\mathcal{C}_{[j]} = \lambda$ if $j \in L_\lambda$. Given the cluster information, we have the advantage of summarizing similar items into a single representation. Specifically, when the context vector only attends to items within the same cluster as the current item, we force that vector to contain only the information from that cluster. Naturally, a mask is introduced to enforce such constraint: let $M \in \{0,1\}^{l \times l}$ be the mask matrix where $M_{i,j} = \mathbb{1}_{[\mathcal{C}_{[i]} = \mathcal{C}_{[j]}]}$ (and 0 otherwise). Each attention head h produces the context vector at position j by aggregating the sequence as:

$$\boldsymbol{\phi}_j^h = \sum_{i=1}^{l} a_{i,j}^h M_{i,j} \boldsymbol{p}_i \quad (5)$$

To process the context vector from all attention heads, a dropout layer and a feed-forward network (FFN) are applied, and the output vector is computed as

$$\boldsymbol{\phi}_j = FFN(Dropout([\boldsymbol{\phi}_j^1; ...; \boldsymbol{\phi}_j^H])), \quad j = 1, \cdots, l \quad (6)$$

The $FFN()$ consists of two fully-connected linear layers with a hyperbolic tangent activation function after the first layer, *i.e.* $FFN(x) = W(tanh(W'x + b')) + b$.

When item metadata is absent and $\boldsymbol{p}$ need to be learned, the mask M will be an all-ones matrix, and output vectors are still computed according to Equation 5 and 6. Then the item clusters are computed from $\boldsymbol{\phi}$ instead of from $\boldsymbol{p}$. We still use $\mathcal{C}$ to represent such cluster assignment, and it will be used to define the user's multi-interest embedding as below.

So far, the multi-head attention module has produced l vectors $\boldsymbol{\phi}_1, ..., \boldsymbol{\phi}_l$, and each $\boldsymbol{\phi}_j$ uses $\boldsymbol{p}_j$ as the (unprojected) query. We will build the multi-interest user embedding by selecting the Λ context vectors that represent each cluster. Denote the position of the last item in each cluster λ as μ_λ (*i.e.* $\mu_\lambda = \arg\max_j(\mathcal{C}_{[j]} = \lambda)$), we will take context vector at that position to represent the λ-th user embedding. In sum, the multi-interest user embedding is

$$Z = [\mathbf{z}_1^\top; ...; \mathbf{z}_\Lambda^\top] = [\boldsymbol{\phi}_{\mu_1}^\top; ...; \boldsymbol{\phi}_{\mu_\Lambda}^\top] \in \mathbb{R}^{\Lambda \times d} \quad (7)$$

Each $\boldsymbol{z}_\lambda$ attends to only items that belong to the same cluster as the item on position μ_λ.

3.3. Cluster Weight Modeling

Besides the multi-interest user embedding, it's also likely that a user favors each interest unequally. As mentioned earlier, ranking these interests correctly can greatly benefit the candidate generation task given its limited budget. PinnerSage (Pal et al., 2020) uses an exponential-decay heuristic function to represent the weight for a cluster, following the assumption that more recent interactions should contribute more to the cluster weight. While we believe that the intuition is generally true, it would be best for the model to automatically learn the role of timestamps from the data. We also incorporate the user embeddings on that cluster's dimension ($\boldsymbol{z}_\lambda$) into cluster weight modeling, since certain categories of interest can have different impacts on the cluster weights as well. Therefore we design the cluster weights to be a function of the recency of the interaction and user embeddings. We model the weight of cluster λ as

$$w_\lambda = FFN([\boldsymbol{z}_\lambda;\ \mathbb{1}_{[C_{[1]}=\lambda]} \cdot \boldsymbol{\tau}_1; ...; \mathbb{1}_{[C_{[l]}=L_\lambda]} \cdot \boldsymbol{\tau}_l]) \quad (8)$$

The first part inside the FFN is the user's embedding on cluster λ. The second part inside the FFN serves as a mask that retains only the timestamps of relevant items that belong to cluster λ. We will discuss how to learn these weights in the next subsection.

3.4. User-Item Interaction Modeling

On the item level, intuitively, a user will engage with an item as long as the item matches *at least one* of his/her interests (not *all*). For example, a user who is interested in both running and home decor purchased a lawn mower, and this behavior will be explained by the user's embedding of the "home decor" cluster (*i.e.* $\boldsymbol{z}_{home}$) and has nothing to do with the embedding of the "running" cluster (*i.e.* $\boldsymbol{z}_{running}$). In other words, the similarity of $\boldsymbol{z}_{home}$ and $\boldsymbol{p}_{lawn\ mower}$ should be high, and the similarity of $\boldsymbol{z}_{running}$ and $\boldsymbol{p}_{lawn\ mower}$ should not even matter. Therefore, when measuring the user-item affinity, we should consider the one user embedding that is the most similar (*e.g.* highest cosine similarity) to the item.

On the cluster level, we need another factor to explain a user's behavior towards different clusters. This can no longer be represented simply by the semantic similarity in the embedding space any more. When the user purchases 20 items in the "home decor" cluster and 5 items in the "running" cluster, it does not indicate that the similarities between $\boldsymbol{z}_{home}$ and these 20 items are higher than the similarities between $\boldsymbol{z}_{running}$ and these 5 items (in fact, all of the similarities should be as high as possible). Therefore, we will multiply the user-item affinity by the cluster weight here to represent a user's intensity towards different clusters.

Considering the arguments above, we propose the likelihood that a user (represented by $Z = [\mathbf{z}_1^\top; ...; \mathbf{z}_\Lambda^\top]$) interacts with an item (represented by $\boldsymbol{p}$) as follow:

$$y = \max\{w_\lambda \cdot (\boldsymbol{z}_\lambda \cdot \boldsymbol{p})\}_{\lambda=1}^{\Lambda} \quad (9)$$

	Amazon	MovieLens	Taobao
# Items	425,582	15,243	823,971
# Interactions	51M	20M	100M
# Training Seq	57,165	127,212	343,171
# Test Seq	5,000	5,000	10,000
# Validation Seq	5,000	5,000	10,000

Table 3. Dataset statistics.

Given the set of items with positive label ($\{\mathcal{I}_+^u\}_{u\in\mathcal{U}}$) and negative label ($\{\mathcal{I}_-^u\}_{u\in\mathcal{U}}$), the negative log-likelihood (NLL) loss of our model can be written as:

$$\mathcal{L} = -\frac{\sum_{u\in\mathcal{U}}\left(\sum_{\boldsymbol{p}_i\in\mathcal{I}_+^u}\log(y_i^u) + \sum_{\boldsymbol{p}_i\in\mathcal{I}_-^u}\log(1-y_i^u)\right)}{\sum_{u\in\mathcal{U}}\left(|\mathcal{I}_+^u| + |\mathcal{I}_-^u|\right)} \tag{10}$$

4. Experiments

We conduct an exhaustive analysis to demonstrate the effectiveness of MIP on the data from Pinterest, one of the largest online content discovery platforms, and a few public datasets. We will divide our discussions to two categories: (1) learning item ID embeddings (Section 4.1) and (2) using item metadata features as is (Section 4.2). Finally, an ablation study is done on different components of the model (Section 4.3).

4.1. Learning Item ID Embeddings

We first evaluate MIP on learning from collaborative filtering datasets, where the item features are absent and will be learned from the user-item interactions.

Dataset. Three public datasets are used: Amazon-book [2] (hereinafter, Amazon), Taobao[3], and MovieLens[4]. We adopt a 10-core setting as previous works (Li et al., 2020; Wang et al., 2019) and filter out items that appear less than 10 times in the dataset. We then split each user's engagement history to non-overlapping sequences of length 100, and use the first 50 items to learn the user embeddings and the last 50 items as labels (as used in Cen et al. (2020)). Any sequence shorter than this threshold are discarded. For each sequence, another 50 negative samples are uniformly sampled at random from the items that the user does not interact with. Our goal is to rank the positive items (that users have actually interacted with) higher than the negative items (random). The dataset statistics are listed in Table 3.

Baseline and model configuration. We compare several open-sourced baseline models with MIP. For fair comparison, we set up the configurations as follow: (1) item and user embedding vectors have the same size ($d = 32$); (2) the number of attention heads is the same ($H = 8$) if the model includes a multi-head attention module; (3) the baseline models should have similar or more parameters than MIP. We let the hidden size in GRU4Rec (Hidasi et al., 2015) be 128, the key and query projected dimension (d_{model} in Equation 3) is labeled in place with the results, and if the model contains a position-wise FFN (Equation 6), it will be a two-layered fully-connected structure with a hidden size of 32 each. The BERT4Rec model (Sun et al., 2019) is originally proposed to predict the item directly as a classification task, so we take its last BERT output as the user embedding to compute the similarity between user and item, and train with the NLL loss. We disabled the session-parallel mini-batch in these models since the session information is absent. We also replace the text encoder in the PinText2 (Zhuang et al., 2020) with an item embedding layer since the inputs in our experiments are items instead of texts.

Training setup. All the models are trained for 100 epochs on a NVIDIA Tesla T4 GPU with an early stop strategy that stops the training when validation AUC does not improve for 20 epochs. The clustering method used in MIP is the Ward clustering algorithm Ward Jr (1963).[5] We compare other clustering methods in Appendix A.2.

Training strategy. We adopt a two-stage setting in the model training in order to enhance the model performance. In the first stage, we fix all the cluster weights to be 1 and train the remaining parameters. After the model converges, we no longer freeze the cluster weights and all parameters are trained until converge. Table 5 summarizes the model performance of two-stage training vs. joint training, and we can clearly see that this strategy is working really well in practice. The rationale behind this is the cluster weights tend to converge too quickly before clusters emerge, therefore we deliberately let the clusters form first before learning the cluster weights. We want to point out this maneuver in our training and hope it can benefit the implementation of similar models.

Results and analysis. The performance is summarized in Table 4. MIP has a better performance on Amazon and Taobao datasets and is trivially worse than GRU4Rec and ComiRec in AUC on MovieLens. Intuitively, the purchase behavior on e-commerce websites (Amazon, Taobao) can be largely explained by the user's interest in multiple categories or brands, while movie-watching is driven more by a movie's popularity and quality rather than the category. Since all models have very close performance, MIP is still a competitive approach in applications that do not support the

[2]https://jmcauley.ucsd.edu/data/amazon/

[3]https://tianchi.aliyun.com/dataset/dataDetail?dataId=649

[4]https://www.kaggle.com/grouplens/movielens-20m-dataset

[5]We adopted the scikit-learn implementation. https://scikit-learn.org/stable/modules/generated/sklearn.cluster.AgglomerativeClustering.html, with n_cluster=5, and other default arguments.

Model	Params.	Config		Amazon			Taobao			MovieLens			Latency
		layers	d_{model}	AUC	recall	nDCG	AUC	recall	nDCG	AUC	recall	nDCG	(*ms*)
GRU4Rec	66338	1	–	0.682	0.635	0.678	0.816	0.745	0.794	0.961	0.903	0.934	1.15
BERT4Rec	50242	1	64	0.681	0.632	0.678	0.815	0.745	0.794	0.960	0.901	0.928	38.34
BERT4Rec	55426	2	32	0.721	0.665	0.698	0.815	0.745	0.794	0.960	0.902	0.941	57.53
PinText2	69634	1	256	0.558	0.541	0.608	0.716	0.669	0.691	0.883	0.817	0.782	14.46
TiSASRec	67586	2	64	0.721	0.667	0.704	0.815	0.744	0.794	0.960	0.902	0.928	14.54
ComiRec	67586	1	256	0.717	0.674	0.704	0.709	0.656	0.698	**0.963**	0.907	**0.979**	14.61
MIP	49347	1	32	**0.805**	**0.789**	**0.781**	**0.885**	**0.884**	**0.909**	0.930	**0.933**	0.954	40.05

Table 4. Performance on public datasets. *Params* excludes the parameters in the item embedding table. Recall and nDCG are measured at top-50 items. See Appendix A.3 for latency measure details.

MIP	Amazon			Taobao			MovieLens		
Model	Epoch	AUC	recall@50	Epoch	AUC	recall@50	Epoch	AUC	recall@50
After the first stage	22	0.731	0.667	4	0.821	0.749	34	0.960	0.901
After the second stage	6	0.806	0.789	4	0.885	0.884	6	0.930	0.933
Joint training	18	0.576	0.570	28	0.803	0.802	14	0.924	0.937

Table 5. Comparison of training strategy and the performance of MIP. The columns *Epoch* shows the training epochs when the best validation AUC is achieved.

strong multi-interest assumption.

4.2. Using Item Metadata Features

Dataset. The dataset contains user engagement history collected from Pinterest, an image-sharing and social media service that allows users to share and discover visual content (images and videos). The interactions between a user and an item (also referred to as a *pin*) are categorized into *impression* (pin is shown to the user), *clickthrough* (user clicks the pin), *re-pin* (user saves the pin into their board collection), and *hide* (user manually hides the pin). In total, there are 38 million interactions from 510 thousand users during three weeks of time. Each pin is represented as a 256-dimension feature extracted by the PinSage model (Ying et al., 2018).

User's engagement sequences are processed in a similar way as the public datasets, except that we enforce a one-day gap between the inputs and labels, because adjacent user engagements are usually very similar which makes the prediction task easy. Intuitively, we can use clickthrough and re-pin as the positive label, and hide (which is less often) and impression (without click or re-pin) as the negative label, but since impressions are also recommended to the user at some point, they are likely to be relevant to the user as well, and thus correlate with the positive data. In order to alleviate this bias, we introduce the *random* negative data where pins are sampled from the whole set of pins. The entire negative dataset will consist of 50% *observed* negative data (hide and impression), and 50% *random* negative data.

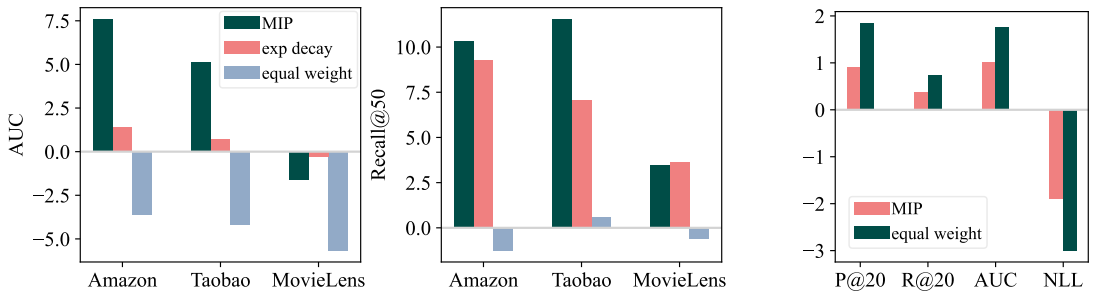
Baselines and model configuration. We compare the multi-interest models PinnerSage and ComiRec, and the single-embedding model TiSASRec with the same setting as in Section 4.1.

	precision@20	recall@20	AUC	NLL
PinnerSage	0.740	0.296	0.815	1.033
TiSASRec	0.798	0.312	0.850	0.478
ComiRec	0.864	0.345	0.875	0.407
MIP	**0.882**	**0.353**	**0.893**	**0.377**

Table 6. Performance on the Pinterest dataset.

Results and analysis. As shown in Table 6, MIP outperforms all the state-of-the-art multi-interest sequential models.

- PinnerSage shares the same clustering algorithm with MIP, but differs in that 1) each cluster embedding is represented by the medoid of all of its item embeddings; and 2) the cluster weights are heuristic-based (not learned from the model). Instead, MIP learns the cluster representations and weights collectively from data, and thus has a clear advantage over PinnerSage.
- TiSASRec has a similar attention module as MIP, except only using the single last attention output as the user embedding. The comparison confirms the necessity of multi-interest representation, as in ComiRec and MIP.
- Compared to ComiRec, MIP interestingly shows that self-attention has stronger representation power than attention with global query. In Appendix A.1, we further use synthetic data to illustrate the fundamental difference between the two types of models.



(a) Public datasets (b) Pinterest dataset

Figure 3. Cluster weight variants performance (difference to the best baseline models, in the unit of 10^{-2})

NLL		Triplet (α =0.2)		Triplet (α =0.5)		Triplet (α =0.8)	
AUC	R@50	AUC	R@50	AUC	R@50	AUC	R@50
0.930	0.933	0.885	0.882	0.903	0.901	0.906	0.906

Table 7. MIP on MovieLens with different loss functions.

4.3. Ablation Study

This section focuses on the claim that 1) the user's preference should be dynamically learned from the temporal pattern by demonstrating the effectiveness of the learned personalized cluster weights module of MIP; 2) The NLL loss is sufficient to reach good performance without the need to sweep the hyperparameters required in other losses, *e.g.* triplet loss; 3) effectiveness of positional and temporal encoding (Section 4.4). We refer the readers to Appendix for a comprehensive ablation study on attention mechanism (A.1), re-configuration and comparison of the clustering method post-training (A.2). We will focus on the ablation study on interest weights and loss function in this section.

Interest weights. To validate the assumption that the preference trends (weights of multiple interests) change from user to user, we compare the MIP with two variants that disable the cluster weight module. The first one (referred to as *MIP (Equal Weight)*) constantly assigns 1 to the cluster weights, leading to an equally weighted multi-interest user representation. Another one (referred to as *MIP (Exp Decay)*) uses an exponential-decay heuristic weights given by $w_\lambda = \sum_{i:C_i \in \lambda} exp(-\epsilon(t_{now} - t_i))$(Pal et al., 2020). We let t_{now} be the last user engagement time t_{last_item}, since in practice it's unrealistic to update the weights in real-time. According to Pal et al. (2020), we also set $\epsilon = 0.01$, which balances well between emphasizing recently engaged items and accentuating frequently engaged categories. We let the number of clusters be $\Lambda = 5$. We calculate performance on these model variants and show the relative percentage difference from the best baseline model in Figure 3. Note that smaller NLL means better performance. In all experiments, the AUC and recall can benefit from learning cluster weights (MIP). The second best variant is usually having an equal weight for all clusters.

Loss function. Beside the NLL loss function in Equation 10, triplet loss is also widely used in recommendation system and contrastive learning. Let the y^+ be the similarity prediction between the user representation and a positive item, and y^- be the predicted similarity between user representation and a negative item. The triplet loss is given by:

$$\mathcal{L}_\alpha = \sum (y^+ - y^- + \alpha)_{\mu \in \mathcal{U}} \tag{11}$$

where α is a hyperparameter of the positive-negative margin. With triplet loss, we added a linear factor β (*learned*) in Equation 9, (*i.e.* $y = max\{\beta w_\lambda \cdot (\boldsymbol{z}_\lambda \cdot \boldsymbol{p})\}_{\lambda=1}^{\Lambda}$) to re-scale the similarity since y is unbounded and makes the choice of α to be hard. We let the MIP train on the MovieLens dataset to investigate the impact on the performance of loss function choice. The results in Table 7 illustrate that the triplet loss marginally underperforms the NLL loss we used.

4.4. Positional and Temporal Encoding

In Equation 2, the sequential (positional) and temporal information are encoded and included in the self-attention module to produce the multi-interest representations. The motivation is that given items from the same category, the recent ones might better represent the user's current interest than the obsolete items. We verify 1) if the incorporation of positional and temporal encoding is critical to the performance; 2) how the encoding (Equation 1) method affects the performance.

Configuration. The MIP are configured on the two set of choices: the Equation 2 can be configured alternatively:

- item embedding only: $\boldsymbol{e}_j = \boldsymbol{p}_j$.
- + positional: $\boldsymbol{e}_j = [\boldsymbol{p}_j; \boldsymbol{\rho}(j)]$, where $\boldsymbol{\rho}(j)$ is given by Equation 1.

Dataset	Configuration	AUC	Recall
Amazon	item embedding	76.34	73.75
	+positional	76.19	74.28
	+temporal	78.59	76.94
	+both	**79.31**	**78.22**
Taobao	item embedding	82.06	81.75
	+positional	86.54	**86.22**
	+temporal	**86.72**	86.56
	+both	86.59	85.83
MovieLens	item embedding	**95.26**	**94.45**
	+positional	95.01	94.31
	+temporal	94.96	94.19
	+ both	94.61	94.12

Table 8. Ablation study on the positional and temporal encoding on public datasets. (in 10^{-2})

- + temporal: $\boldsymbol{e}_j = [\boldsymbol{p}_j; \boldsymbol{\tau}(t_j)]$, where $\boldsymbol{\tau}(t_j)$ is given by Equation 1.
- + positional and temporal: Equation 2 and Equation 1

and there are several other choices of temporal encodings other than the sinusoidal form in the Equation 1:

- One-hot (Zhou et al., 2018): Create exponential buckets $[0, b)$, $[b, b^2)$, $\cdots$, $[b^{k-1}, \infty)$ with base b, and encode the timestamp as an one-hot vector, *i.e.* $\tau_i = lookup(buckets(t))$.
- Two-hot (Shi et al., 2021): Similarly create exponential boundaries $\{0, b, b^2, \cdots, b^{k-1}, \infty\}$, and encode the timestamp as $\boldsymbol{\tau}_i = log_b(t) - i$ and $\boldsymbol{\tau}_{i+1} = i + 1 - log_b(t)$, where $b^i \leq t < b^{i+1}$.

Dataset and Results. The options to include positional and temporal information are evaluated on all the dataset (Table 8), and the encodings methods are compared exhaustively on Pinterest dataset (Table 9).

Analysis. Two conclusions can be made from Table 8 and Table 9: 1) including both temporal and positional information is a safe option, which has the best performance on Amazon and Pinterest and marginally (< 0.01) worse performance on Taobao and MovieLens; and 2) the model is insensitive to encoding methods.

5. Conclusions

In this paper, we study the problem of multi-interest user embedding for recommendation systems. We follow the recent findings on representing users with multiple embeddings, which has been proven helpful over the single user representation. In addition, we illustrate that in industrial recommendation systems, it is important to have a set of weights for these multiple embeddings for a more efficient candidate generation process due to its budget on the number of items returned. More specifically, we define the likelihood of an engagement based on the *closest* user embedding to the item embedding and update the weight for the corresponding cluster. Moreover, the case studies on multiple real-world datasets have demonstrated our advantage over state-of-the-art approaches. Finally, the ablation study on the cluster weight module demonstrated our intuition that simple heuristic does not work as well as personalized model-learned interest weights. We refer readers to the appendix for extensive study on other model design choices.

Configuration	AUC	NLL
item embedding	0.8923	**0.377**
+ positional	0.8846	0.386
+ temporal (one-hot)	0.8850	0.388
+ temporal (two-hot)	0.8846	0.385
+ temporal (sinusoid)	0.8921	**0.377**
+ both (one-hot)	0.8861	0.382
+ both (two-hot)	0.8852	0.387
+ both (sinusoid)	**0.8926**	**0.377**

Table 9. Influence of temporal and positional encoding in attention on the performance in MIP

References

Cen, Y., Zhang, J., Zou, X., Zhou, C., Yang, H., and Tang, J. Controllable multi-interest framework for recommendation. In *Proceedings of the 26th ACM SIGKDD International Conference on Knowledge Discovery & Data Mining*, pp. 2942–2951, 2020.

Chen, X., Zhang, Y., and Qin, Z. Dynamic explainable recommendation based on neural attentive models. In *Proceedings of the AAAI Conference on Artificial Intelligence*, volume 33, pp. 53–60, 2019.

Cheng, H.-T., Koc, L., Harmsen, J., Shaked, T., Chandra, T., Aradhye, H., Anderson, G., Corrado, G., Chai, W., Ispir, M., et al. Wide & deep learning for recommender systems. In *Proceedings of the 1st workshop on deep learning for recommender systems*, pp. 7–10, 2016.

Covington, P., Adkan, J., and Sargin, E. Deep neural networks for youtube recommendations. In *Proceedings of the 10th ACM conference on recommender systems*, pp. 191–198, 2016.

Devooght, R. and Bersini, H. Long and short-term recommendations with recurrent neural networks. In *Proceedings of the 25th Conference on User Modeling, Adaptation and Personalization*, pp. 13–21, 2017.

Donkers, T., Loepp, B., and Ziegler, J. Sequential user-based recurrent neural network recommendations. In *Proceed-*

ings of the Eleventh ACM Conference on Recommender Systems, pp. 152–160, 2017.

Epasto, A. and Perozzi, B. Is a single embedding enough? learning node representations that capture multiple social contexts. In *The World Wide Web Conference*, pp. 394–404, 2019.

Ester, M., Kriegel, H.-P., Sander, J., Xu, X., et al. A density-based algorithm for discovering clusters in large spatial databases with noise. In *kdd*, volume 96, pp. 226–231, 1996.

He, R. and McAuley, J. Fusing similarity models with markov chains for sparse sequential recommendation. In *2016 IEEE 16th International Conference on Data Mining (ICDM)*, pp. 191–200. IEEE, 2016.

Hidasi, B., Karatzoglou, A., Baltrunas, L., and Tikk, D. Session-based recommendations with recurrent neural networks. *arXiv preprint arXiv:1511.06939*, 2015.

Jiang, H., Wang, W., Wei, Y., Gao, Z., Wang, Y., and Nie, L. What aspect do you like: Multi-scale time-aware user interest modeling for micro-video recommendation. In *Proceedings of the 28th ACM International Conference on Multimedia*, pp. 3487–3495, 2020.

Kang, W.-C. and McAuley, J. Self-attentive sequential recommendation. In *2018 IEEE International Conference on Data Mining (ICDM)*, pp. 197–206. IEEE, 2018.

Li, C., Liu, Z., Wu, M., Xu, Y., Zhao, H., Huang, P., Kang, G., Chen, Q., Li, W., and Lee, D. L. Multi-interest network with dynamic routing for recommendation at tmall. In *Proceedings of the 28th ACM International Conference on Information and Knowledge Management*, pp. 2615–2623, 2019.

Li, J., Wang, Y., and McAuley, J. Time interval aware self-attention for sequential recommendation. In *Proceedings of the 13th International Conference on Web Search and Data Mining*, pp. 322–330, 2020.

Li, Y., Lu, L., and Xuefeng, L. A hybrid collaborative filtering method for multiple-interests and multiple-content recommendation in e-commerce. *Expert systems with applications*, 28(1):67–77, 2005.

Pal, A., Eksombatchai, C., Zhou, Y., Zhao, B., Rosenberg, C., and Leskovec, J. Pinnersage: Multi-modal user embedding framework for recommendations at pinterest. In *Proceedings of the 26th ACM SIGKDD International Conference on Knowledge Discovery & Data Mining*, pp. 2311–2320, 2020.

Pérez, J., Marinković, J., and Barceló, P. On the turing completeness of modern neural network architectures. *arXiv preprint arXiv:1901.03429*, 2019.

Quadrana, M., Karatzoglou, A., Hidasi, B., and Cremonesi, P. Personalizing session-based recommendations with hierarchical recurrent neural networks. In *proceedings of the Eleventh ACM Conference on Recommender Systems*, pp. 130–137, 2017.

Rendle, S., Freudenthaler, C., and Schmidt-Thieme, L. Factorizing personalized markov chains for next-basket recommendation. In *Proceedings of the 19th international conference on World wide web*, pp. 811–820, 2010.

Sabour, S., Frosst, N., and Hinton, G. E. Dynamic routing between capsules. *arXiv preprint arXiv:1710.09829*, 2017.

Shi, H., Zhang, Y., Wu, H., Chang, S., Qian, K., Hasegawa-Johnson, M., and Zhao, J. Continuous cnn for nonuniform time series. In *ICASSP 2021-2021 IEEE International Conference on Acoustics, Speech and Signal Processing (ICASSP)*, pp. 3550–3554. IEEE, 2021.

Shi, H., Gao, S., Tian, Y., Chen, X., and Zhao, J. Learning bounded context-free-grammar via lstm and the transformer: Difference and the explanations. In *Proceedings of the AAAI Conference on Artificial Intelligence*, volume 36, pp. 8267–8276, 2022.

Shi, J. and Malik, J. Normalized cuts and image segmentation. *IEEE Transactions on pattern analysis and machine intelligence*, 22(8):888–905, 2000.

Sun, F., Liu, J., Wu, J., Pei, C., Lin, X., Ou, W., and Jiang, P. Bert4rec: Sequential recommendation with bidirectional encoder representations from transformer. In *Proceedings of the 28th ACM international conference on information and knowledge management*, pp. 1441–1450, 2019.

Vaswani, A., Shazeer, N., Parmar, N., Uszkoreit, J., Jones, L., Gomez, A. N., Kaiser, L., and Polosukhin, I. Attention is all you need. *arXiv preprint arXiv:1706.03762*, 2017.

Wandabwa, H., Naeem, M. A., Mirza, F., Pears, R., and Nguyen, A. Multi-interest user profiling in short text microblogs. In *International Conference on Design Science Research in Information Systems and Technology*, pp. 154–168. Springer, 2020.

Wang, F. Multi-interest communities and community-based recommendation. 2007.

Wang, J., Huang, P., Zhao, H., Zhang, Z., Zhao, B., and Lee, D. L. Billion-scale commodity embedding for e-commerce recommendation in alibaba. In *Proceedings of the 24th ACM SIGKDD International Conference on Knowledge Discovery & Data Mining*, pp. 839–848, 2018.

Wang, X., He, X., Wang, M., Feng, F., and Chua, T.-S. Neural graph collaborative filtering. In *Proceedings of the 42nd international ACM SIGIR conference on Research and development in Information Retrieval*, pp. 165–174, 2019.

Ward Jr, J. H. Hierarchical grouping to optimize an objective function. *Journal of the American statistical association*, 58(301):236–244, 1963.

Weston, J., Weiss, R. J., and Yee, H. Nonlinear latent factorization by embedding multiple user interests. In *Proceedings of the 7th ACM conference on Recommender systems*, pp. 65–68, 2013.

Xia, B., Li, Y., Li, Q., and Li, T. Attention-based recurrent neural network for location recommendation. In *2017 12th International Conference on Intelligent Systems and Knowledge Engineering (ISKE)*, pp. 1–6. IEEE, 2017.

Xu, C., Zhao, P., Liu, Y., Xu, J., S. Sheng, V. S. S., Cui, Z., Zhou, X., and Xiong, H. Recurrent convolutional neural network for sequential recommendation. In *The World Wide Web Conference*, pp. 3398–3404, 2019.

Xue, G.-R., Lin, C., Yang, Q., Xi, W., Zeng, H.-J., Yu, Y., and Chen, Z. Scalable collaborative filtering using cluster-based smoothing. In *Proceedings of the 28th annual international ACM SIGIR conference on Research and development in information retrieval*, pp. 114–121, 2005.

Ying, R., He, R., Chen, K., Eksombatchai, P., Hamilton, W. L., and Leskovec, J. Graph convolutional neural networks for web-scale recommender systems. In *Proceedings of the 24th ACM SIGKDD International Conference on Knowledge Discovery & Data Mining*, pp. 974–983, 2018.

You, J., Wang, Y., Pal, A., Eksombatchai, P., Rosenburg, C., and Leskovec, J. Hierarchical temporal convolutional networks for dynamic recommender systems. In *The world wide web conference*, pp. 2236–2246, 2019.

Yu, L. Using ontology to enhance collaborative recommendation based on community. In *2008 The Ninth International Conference on Web-Age Information Management*, pp. 45–49. IEEE, 2008.

Yue, W. and Xiang, C. A multi-interests model of recommendation system based on customer life cycle. In *2012 Fifth International Conference on Intelligent Computation Technology and Automation*, pp. 22–25. IEEE, 2012.

Zhang, T., Ramakrishnan, R., and Livny, M. Birch: an efficient data clustering method for very large databases. *ACM sigmod record*, 25(2):103–114, 1996.

Zhou, C., Bai, J., Song, J., Liu, X., Zhao, Z., Chen, X., and Gao, J. Atrank: An attention-based user behavior modeling framework for recommendation. In *Proceedings of the AAAI Conference on Artificial Intelligence*, volume 32, 2018.

Zhuang, J., Zhao, J., Subramanian, Srinivas, A., Lin, Y., Krishnapuram, B., and Zwol, R. v. Pintext 2: Attentive bag of annotations embedding. In *Proceedings of DLP-KDD 2020*, 2020.

Model	Global query		Self-attention
	Non-shared	Shared	
Key vector $\boldsymbol{k}_j$	$(W_r^h \boldsymbol{p}_j + \boldsymbol{b}^h)$		
Query vector	$\boldsymbol{q}^h$	$\boldsymbol{q}$	$\boldsymbol{q}_i^h = W_q^h \boldsymbol{p}_i + \boldsymbol{b}_q^h$
$\boldsymbol{q}$ shared between sequences	Yes		No
$\boldsymbol{q}$ shared between att. heads	No	Yes	No
Dot-product	$e_j^h = \boldsymbol{q}^{h\top} \cdot \boldsymbol{k}_j^h$	$e_j^h = \boldsymbol{q}^\top \cdot \boldsymbol{k}_j^h$	$e_{i,j}^h = \boldsymbol{q}_i^{h\top} \cdot \boldsymbol{k}_j^h$

Table 10. Variations of multi-head attention.

A. Appendix

A.1. Variants of Attention Mechanism

To interpret the performance improvement of our models against other attention models that have been applied in the recommendation system, we further construct a synthetic dataset and visualize the internal attention and the user representations aggregated from different attentions.

Synthetic dataset. Without loss of generality, we assume there are G global clusters in the corpus, representing different global categories, each of which is a d-dimensional Gaussian distribution. Each user is interested in up to k ($< G$) categories, referred to as user clusters. We generate the oracle user interest model by sampling no more than k clusters from G global clusters following a multinomial distribution. Then each of the items in the user engagement history is sampled in two steps: uniformly sample one cluster from user clusters, then sample from the d-dimensional Gaussian distribution. Note that in the synthetic model, the item-to-cluster affinity is measured in Euclidean distance, while in the recommendation model, the affinity is decided by cosine distance. To eliminate this discrepancy, we force the Gaussian distributions to center on the unit sphere, so that the rankings by cosine distance and Euclidean distance are consistent.

We use a 2D dataset ($d = 2$) for visualization purposes and another high-dimensional dataset for quantitative evaluation. For the 2D dataset, we set a relatively small $G = 8$ and $k = 4$ in order to have a clear boundary between clusters. Since there are only 162 distinct subsets[6] with $G = 8, k = 4$, we use 100 of them for training, 31 for validation, and the remaining 31 for testing. For the high-dimensional dataset, we set $G = 1024$ and $k = 8$, and let $d = 16, 32, 64, 128$. We generate 10000 users for training, 1000 for validation, and another 1000 for testing.

Attention models. We focus on comparing our attention model (i.e. *self-attention*), the attention model utilized in ComiRec (i.e. *Non-shared query*), and the one used in PinText2 (i.e. *Shared query*). The comparison of the attentions is in Table 10.

For simplicity, we remove the temporal and positional encoding from the computation of attentions, skip the Ward clustering step from MIP, and directly represent user as Equation 7. Also, the dropout layer is removed in order to eliminate randomness in visualization.

Metrics. We visualize the intermediate results and user representations learned from the 2D dataset for qualitative evaluation. For high-dimensional data, we evaluate the performance by AUC and normalized discounted cumulative gain (nDCG).

Qualitative results and interpretations. Figure 4 shows the learned user representations given the engagement history. There are three observations. 1) When $H = 1$, global query attention fails to capture all the user interests, while the self-attention model is free from the limitation. 2) Viewing from the third row, the self-attention model is more accurate in learning cluster representations than global query models. The latter is systematically biased due to the global query as shown in global query models in Figure 6. 3) All the models learn super-clusters, depending on the bias in the dataset. For the example shown in Figure 4, the two adjacent clusters on the top side of the unit circle are often represented to be a super-cluster.

We also visualize the internal attention scores and self-attention models (Figure 5). Some attention heads show highly similar attention patterns because their queries are close to each other, which can be verified from Figure 6. Figure 5 compares the ground truth attention model with the learned attention. The learned attention shows clear boundaries between clusters in the heatmap. Note that the ground truth ignores the adjacency of clusters but the self-attention model considers the similarity between clusters, so Figure 5(a) is block diagonal while Figure 5(b-d) has dark blocks off the diagonal.

[6]Number of ways to select no more than 4 clusters from a pool of 8 clusters: $162 = \binom{8}{1} + \binom{8}{2} + \binom{8}{3} + \binom{8}{4}$

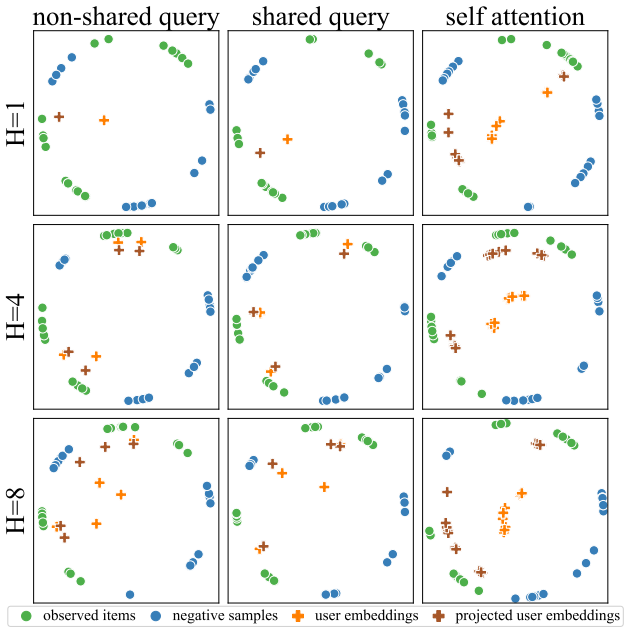


Figure 4. Learned user representations with different attention mechanisms. Non-shared and shared global query results might miss some user interest or are close to the negative categories, while self-attention results are comprehensive and accurate.

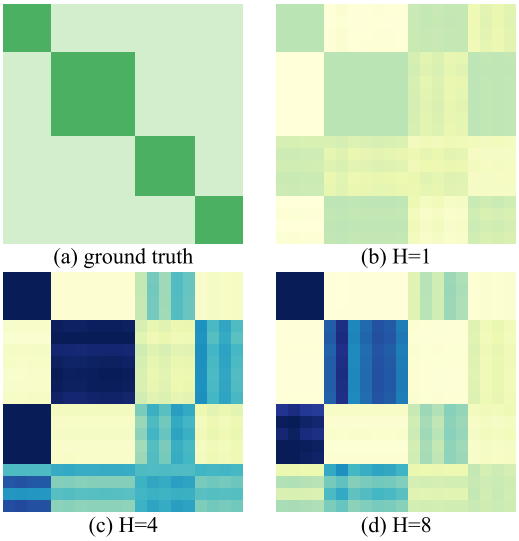


Figure 5. Learned attention scores in self-attention model. Darker color represents higher values. (a) the indicator function $\mathbb{1}_{[c_i=c_j]}$, (b-d) $a_{i,j}$. The input sequence is re-ordered for better visualization.

Quantitative results. Previous results show the intuitive comparison between global query models and the self-attention model, and the quantitative results further confirm the consistency of performance gain of self-attention. Experiments are repeated on the dataset for feature dimension $d = 16, 32, 64, 128$ and number of attention heads $H = 4, 8$. Figure 7 shows that the MIP model constantly and significantly outperforms global query models. As illustrated in the 2D dataset, the performance gain benefits from the personalized user representation, rather than matching to the globally popular clusters. Another observation from the result is that for global query models, $H = 4$ under-performs $H = 8$ models, as the number of attention heads decides the number of global clusters the model can learn; however, for self-attention model, $H = 4$ performs even better than $H = 8$. The explanation is that the self-attention model does not require a growing number of attention heads with respect to the number of global clusters, and $H = 4$ could be already enough for capturing user interest but easier than $H = 8$ to train.

A.2. Clustering Options

The Ward's algorithm is applied to MIP considering its success in PinnerSage(Pal et al., 2020), it's beneficial to explore the selection of the clustering algorithm and the number of clusters on the collaborative filtering dataset. To illustrate the impact, we evaluate MIP with a wide range of clustering algorithms.

Model Configuration and training: MIP models are configured with an attention module that takes both positional and temporal encoding. For unweighted MIP, no clustering method is applied to the encoded user engagement history $\{z_*\}$ (computed from Eq. 6) in the training stage. For weighted MIP, Ward's algorithm is applied to $\{z_*\}$ and the number of clusters is set to 5. To keep the MIP fully differentiable, the cluster embedding is the encoding of the last item in each cluster, instead of the medoid.

Inference: The choice of clustering in the inference phase is independent of its configuration during the training. We explore the inference options on the pre-trained models. Different types of clustering methods are compared:

- Ward: hierarchical clustering method that minimizes the sum of squared distances within all clusters.
- K-Means: an iterative method also minimizes the sum of in-cluster summed squared distances.
- Spectral(Shi & Malik, 2000): performs clustering on the projection of the normalized Laplacian computed from the

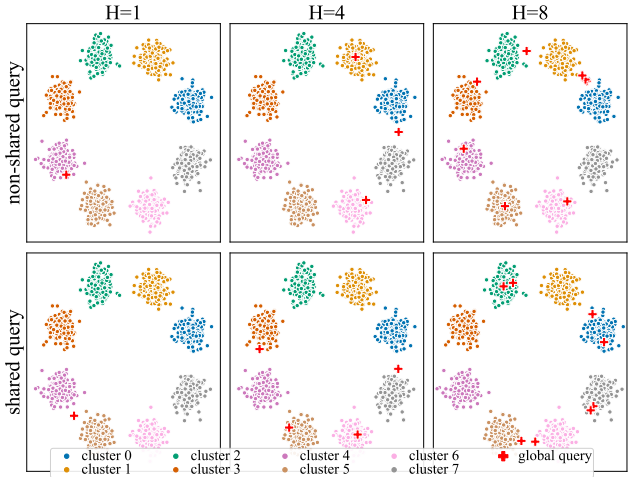


Figure 6. Learned global interests in global query models. The query vector is reversely projected and normalized.

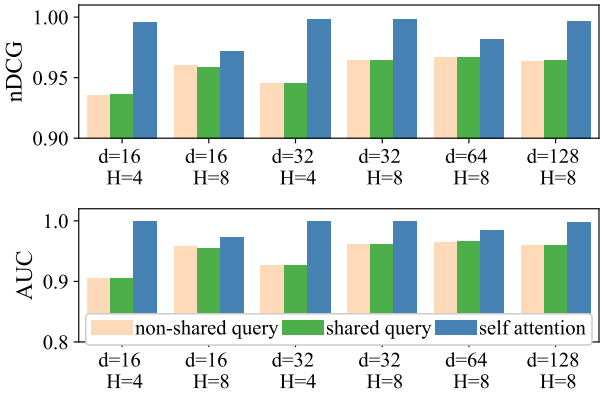


Figure 7. Performance comparison on high dimensional synthetic dataset. d denotes the feature dimension and H is the number of attention heads.

affinity matrix.

- BIRCH(Zhang et al., 1996): another hierarchical method that clusters the points by building the Clustering Feature Tree.
- DBSCAN(Ester et al., 1996): a density-based clustering method that does not require specifying the number of clusters.

The number of clusters is set to 5, 8, and 10 when required. Note that during training, the number of clusters is fixed to 5, however, after training, MIP can produce other numbers of embeddings per user, which gives the system huge flexibility to trade-off between storage/computation cost and recommendation performance.

Result and analysis: There are two observations from Table 11. 1) the choice of clustering algorithm has a marginal impact on the performance. While PinnerSage reported that Ward's algorithm outperforms the K-Means, their result does not conflict with our observation here. Recall that for PinnerSage and our experiment on the Pinterest dataset, the clustering method is applied to the exogenous item embeddings, thus the clustering methods can be influenced by the non-flat geometry and outliers. However, with the collaborative filtering dataset, the clustering method is applied to the encodings produced by multi-head self-attention layers which average the embedding of the items and all other items (Eq. 3). The encodings after the multi-head self-attention should be smoothly distributed, and as a result, any clustering methods work almost equally well on that. 2) Selecting the number of clusters is a non-trivial trade-off. The motivation to decrease the number of clusters is the storage and computation cost which grow linearly as the number of clusters increases. For unweighted MIP, though the non-clustering (each item is a cluster) settings have the best AUC, decreasing the number of user embedding from 50 (non-clustering) to 10 is still acceptable. For weighted MIP, since it's impossible to learn the clustering weights without applying a clustering method, the trade-off can be more complicated: besides the storage concern, when the number of clusters increases the average information to learn the weights of each cluster decreases, and consequently may hurt the overall performance; on the other hand, 10-cluster settings are better than the 5-cluster settings for all the dataset.

A.3. Model Latency Comparison

Seeing the performance gain, another prominent question will be what is the time cost of the performance increase. In this section, we profiled the model latency on a desktop computer with a 12-core Intel i7-8700k CPU, and a single Nvidia GeForce RTX 2080 Ti GPU. The neural network training and inference are on the GPU with vanilla PyTorch framework (version 1.12) without any further optimization on the computation. The clustering algorithm in MIP is performed by CPU with Python's scikit-learn package. We set the batch size to 1 and the dataset to Amazon, then measure and summarize the training and inference latency in Table 12.

There are a few observations from Table 12. First, compared to the neural network inference latency, the clustering step time cost is trivial. PinText2, ComiRec, and TiSASRec has similar training and inference latency, while the performance of

Clustering Method	Inference Clusters	Unweighted MIP			Weighted MIP		
		Amazon	Taobao	MovieLens	Amazon	Taobao	MovieLens
None	-	73.11	82.09	95.97	-	-	-
Ward	5	71.56	80.58	95.53	79.31	8**6.49**	94.61
	8	71.99	80.99	95.72	80.47	87.85	**95.25**
	10	72.16	81.20	**95.78**	80.84	88.42	**95.25**
K-Means	5	71.58	80.62	95.53	79.26	86.18	**94.86**
	8	71.95	81.03	95.71	80.66	88.02	95.17
	10	72.14	81.22	95.77	80.62	**88.61**	95.10
Spectral	5	**72.28**	**80.72**	**95.54**	78.99	85.84	94.46
	8	**72.37**	**81.08**	**95.73**	**80.79**	87.61	94.81
	10	**72.64**	**81.26**	95.78	**81.19**	88.40	95.07
BIRCH	5	71.98	80.63	95.52	**79.39**	86.29	94.61
	8	72.03	81.02	95.71	80.65	**88.03**	95.25
	10	72.44	81.21	95.78	80.91	88.53	**95.25**
DBSCAN	-	71.98	80.63	95.52	70.05	75.58	89.63

Table 11. Comparison of clustering options in AUC (in 10^{-2}). Note that the number of inference clusters is independent of training, i.e. changing the number of inference clusters does not require the re-training of the model. With the same number of clusters, the best performances are bold.

Latency/ Recall	GRU4Rec	BERT4Rec ($L = 1$)	BERT4Rec ($L = 2$)	PinText2	TiSASRec	ComiRec	MIP (total)	MIP (clustering)
Train	**218.57**	925.36	1058.34	555.79	452.94	479.59	998.62	5.86
(std)	(0.81)	(108.96)	(733.33)	(76.96)	(67.07)	(67.50)	(56.31)	(0.76)
Inference	**1.15**	38.34	57.53	14.46	14.54	14.61	40.05	5.93
(std)	(0.20)	(56.60)	(56.23)	(54.99)	(56.34)	(55.70)	(27.08)	(0.72)
R@50	63.50	63.15	66.52	54.13	66.67	67.36	**78.85**	-

Table 12. Latency and performance comparison of the models. Training and inference latencies are measured in *ms*, and brackets show the standard deviations.

them is worse than MIP. BERT4Rec has similar latency as MIP since our sequential model architecture is similar, while the BERT4Rec has worse performance. GRU4Rec has the least inference time. Notice that the standard deviations of the inference latency of PinText2, TiSASRec, and ComiRec are large. It indicates, though on average the three models are faster in inference, MIP inference latency is less possible to be very large while the other latency might be several times longer than average.

Conclusively, MIP, as well as other baselines compared, can all satisfy the latency requirement when applying online, even without further optimization on the computation and serving. MIP has a higher time cost compared to some of the baselines, but the performance increase is also appealing.