# Device-Cloud Collaborative Learning for Recommendation

Jiangchao Yao*†, Feng Wang*†, Kunyang Jia*†, Bo Han‡, Jingren Zhou†, Hongxia Yang†
†DAMO Academy, Alibaba Group, Hang Zhou, China; ‡Hong Kong Baptist University, Hong Kong, China;
†{jiangchao.yjc, wf135777, kunyang.jky, jingren.zhou, yang.yhx}@alibaba-inc.com, ‡bhanml@comp.hkbu.edu.hk

## ABSTRACT

With the rapid development of storage and computing power on mobile devices, it becomes critical and popular to deploy models on devices to save onerous communication latencies and to capture real-time features. While quite a lot of works have explored to facilitate on-device learning and inference, most of them focus on dealing with response delay or privacy protection. Little has been done to model the collaboration between the device and the cloud modeling and benefit both sides jointly. To bridge this gap, we are among the first attempts to study the Device-Cloud Collaborative Learning (DCCL) framework. Specifically, we propose a novel *MetaPatch* learning approach on the device side to efficiently achieve "thousands of people with thousands of models" given a centralized cloud model. Then, with billions of updated personalized device models, we propose a "model-over-models" distillation algorithm, namely *MoMoDistill*, to update the centralized cloud model. Our extensive experiments over a range of datasets with different settings demonstrate the effectiveness of such collaboration on both cloud and devices, especially its superiority to model long-tailed users.

## CCS CONCEPTS

• **Information systems → Recommender systems**.

## KEYWORDS

On-device Intelligence, Cloud Computing, Recommender Systems

**ACM Reference Format:**
Jiangchao Yao*†, Feng Wang*†, Kunyang Jia*†, Bo Han‡, Jingren Zhou†, Hongxia Yang†. 2021. Device-Cloud Collaborative Learning for Recommendation. In *Proceedings of the 27th ACM SIGKDD Conference on Knowledge Discovery and Data Mining (KDD '21), August 14–18, 2021, Virtual Event, Singapore.* ACM, New York, NY, USA, 10 pages. https://doi.org/10.1145/3447548.3467097

## 1 INTRODUCTION

Nascent applications for mobile computing and the Internet of Things (IoTs) are driving computing toward dispersion [38]. Specially, the evolving capacity of mobile devices makes it possible to consider the intelligence services, e.g., online recommendation, from cloud to device modeling. Recently, several works in different perspectives like privacy [2, 22, 23], efficiency [4, 14], applications [8, 11, 42] have explored this pervasive computing advantages. Mobile recommender systems with the on-device engines *e.g.,* TFLite[1] and CoreML[2] hence attract more and more attention.

Previous research in this area of recommender systems can be summarized into two lines, *on-device inference* and *on-device learning* given the centralized model. The former deploys the pretrained model to the devices, and executes the inference to save the onerous communication latencies and capture real-time features. This mainly solves the computational efficiency problem, yielding the exploration of the device-friendly model inference. For example, Sun *et al.* [41] proposed CpRec to shrink the sequential recommender system, which compress the size of input and output matrices as well as the parameters of middle layers via adaptive decomposition and parameter sharing. In a broader area, Cai *et al.* [3] introduced a generalized network pruning method, progressive shrinking, to reduce the model size across dimensions for efficient deployment. Gong *et al.* [11] explored a split deployment across cloud and device to reduce the inference cost of on-device components.

The latter line of works aggregate the temporal on-device training pieces into a centralized model to overcome the privacy constraint, namely Federated recommender systems [44]. The gradients of the local copy from the centralized deep model is first executed on plenty of devices, and then are collected to the server to update the model parameters by federated averaging (FedAvg) or its variants [22, 23, 29]. For example, Qi *et al.* [33] applied Federated Learning and differential privacy for news recommendation and achieved a balance between recommendation performance and privacy protection. Lin *et al.* [27] introduced a MetaMF to account for the storage, energy and communication bandwidths in mobile environments, which distributes the gradient computation across cloud and devices. Niu *et al.* [30] followed a similar split in gradient computation but designed a tunable privacy to the local submodel.

Different from the above two lines of works, we focus on how to leverage the advantages of the device modeling and the cloud modeling jointly to benefit both sides. The intuition behind this direction is two-fold: on the one hand, the centralized cloud model usually ignores or even sacrifices the experience of long-tailed samples, users or items, to maximize the global revenue. And long-tailed and non-i.i.d (independent and identically distributed) characteristics in recommendation samples inevitably induce the model bias to the minorities [31]. One possible solution is to personalize the cloud model with local data on each device. On the other hand, training samples on each device alone may be quite limited suffering from local optimization [10]. In contrast, the centralized cloud models

*These authors contributed equally to this research.


*KDD '21, August 14–18, 2021, Virtual Event, Singapore*

ACM ISBN 978-1-4503-8332-5/21/08…$15.00
https://doi.org/10.1145/3447548.3467097

[1] https://www.tensorflow.org/lite
[2] https://developer.apple.com/documentation/coreml

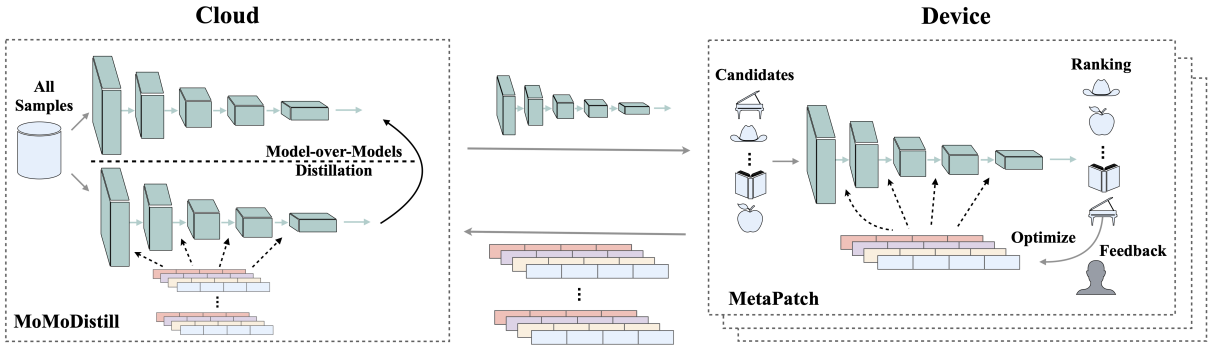


**Figure 1: The general DCCL framework for recommendation. The cloud side is responsible to learn the centralized cloud model via the model-over-models distillation from the personalized on-device models. The device receives the centralized cloud model to conduct the on-device personalization. We propose *MoMoDistill* and *MetaPatch* to instantiate each side respectively.**

are updated with data from all devices and able to avoid this problem. This motivates us to propose a comprehensive Device-Cloud Collaborative Learning (DCCL) framework.

We summarize our contributions as follows:

- Compared to existing works that either only consider the cloud modeling, or on-device inference, or the aggregation of the temporal on-device training pieces to handle the privacy constraint, we formally propose a Device-Cloud Collaborative Learning framework (DCCL) to benefit both sides.
- We propose two novel methods *MetaPatch* and *MoMoDistill* to instantiate each side in DCCL, which consider the sparsity challenge for on-device personalization, and enhance the centralized cloud model via the "model-over-models" *MoMoDistill* distillation instead of the conventional "model-over-data" paradigm.
- Extensive experiments on a range of datasets demonstrate that DCCL outperforms the state-of-the-art methods, and especially, we provide a comprehensive analysis about its advantage to long-tailed users and the inter-loops between computing interactions of cloud and device.

## 2 RELATED WORKS

### 2.1 Recommender System

Recommender systems [36] have been widely studied in the last decade and become an indispensable infrastructure of web services in the era of cloud computing and big data. The related recommendation methods are gradually improved with the development of collaborative filtering, deep learning and sequential modeling. The early stage mainly focused on the user-based collaborative filtering [48], the item-based collaborative filtering [37] and matrix factorization [24, 34]. As deep learning achieved a great success in computer vision, several variants of collaborative filtering combined with deep neural networks were proposed [5–7, 12, 16, 45]. They leveraged the non-linear transformation of deep neural networks to activate high-level semantics for more accurate recommendation. Sequential modeling as another perspective to model the user interests has been successfully applied to recommender systems [39]. With the optimization in architectures, several methods based on GRU [19], Attention [21, 43, 47, 49] have achieved the remarkable performance in recommender systems.

### 2.2 On-device Inference

On-device inference as an important part of Edge AI [40] greatly reduces the onerous latency and incorporates rich features in recommendation. This line of works critically depend on the device capacity [1], efficient neural network architectures [15], the model compression technique [14] and some split deployment strategies. Recently, several hardware efficient architectures like MobileNets [18] were proposed, which made the model size and the computational budget for on-device inference lightweight. Another effective perspective is the model compression based on the network pruning or quantization [3, 14], which reduces the units, the channels or the value accuracy of the parameters. In recommender systems, CpRec [41] considered to construct a compressed counterpart of the sequential models by introducing a block-wise adaptive decomposition and parameter sharing scheme. Lee *et al.* [26] described how to leverage the mobile GPU to run deep neural network tasks, which considers the special limited computing power, thermal constraints, and energy consumption. Dai *et al.* [8] presented an app to demonstrate the potential usage of on-device inference for mobile health applications. Some other works [11] also explored the divide-and-conquer deployment to save the running time on devices, in which the computational prohibitive modules can be moved to the cloud. Nevertheless, all these works still share a centralized model.

### 2.3 On-device Learning

On-device learning is more time-consuming than on-device inference as it requires the extra computation regarding gradients [9]. Current methods are mainly for Federated Learning [23], a distributed learning framework to keep the data in the local and only communicate gradients or parameters. It is important as more and more users consider the privacy protection on the Internet as General Data Protection Regulation (GDPR)[3] claims in Europe. In Federated Learning, on-device learning is used to compute the temporal training pieces and send to the cloud for averaging, and several

[3]https://gdpr-info.eu/

works applied Federated Learning to the recommender systems. Qi *et al.* [33] applied Federated Learning and differential privacy to News Recommendation, which achieved a promising trade-off between recommendation performance and privacy protection. Lin *et al.* [27] introduced a MetaMF to account for the storage, energy and communication bandwidth in mobile environments, which distributed the gradient computation of Federated Learning across cloud and devices. Niu *et al.* [30] followed a similar split in gradient computation and designed a tunable privacy to the local submodel. Nevertheless, we argue that privacy protection is an exemplar usage of conceivable on-device learning. How to explore the collaboration between the cloud modeling and the device modeling for mutual benefit of two sides is also meaningful in practice [25]. One previous work [28] that has the similar intuition differs from our method in the way to handle the sparsity issue and their method has not considered the frequent checkpoint loading issue.

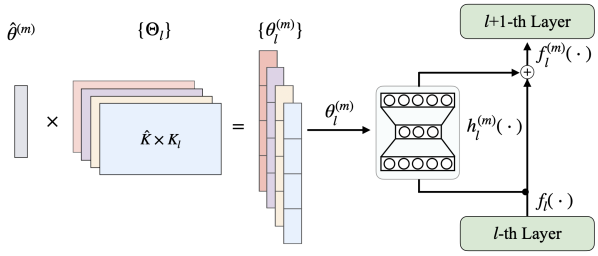


**Figure 2: The proposed *MetaPatch* to reduce the parameter space on the device side. It uses the global parameter basis $\Theta_l$ with the metapatch parameter $\hat{\theta}^{(m)}$ to generate all parameters $\{\theta_l^{(m)}\}$, and parameterize $h_l^{(m)}(\cdot)$ correspondingly. The dimension of patch parameters is reduced to $\hat{K}$ from $\sum_l K_l$.**

## 3 THE PROPOSED FRAMEWORK

### 3.1 Preliminary

Given a recommendation dataset $\{(x_n, y_n)\}_{n=1,\dots,N}$, we target to learn a mapping function $f : x_n \in \mathbb{R}^D \longrightarrow y_n \in \mathbb{R}$ on the cloud side. Here, $x_n$ is the input $D$-dimensional feature that concatenates all available candidate features and user context, $y_n$ is the user implicit feedback (click or not) to the corresponding candidate and $N$ is the sample number. On the device side, each device (indexed by $m$) has its own local dataset, $\left\{(x_n^{(m)}, y_n^{(m)})\right\}_{n=1,\dots,N^{(m)}}$. We add a few parameter-efficient patches [46] to the cloud model $f$ (freezing its parameters on the device side) for each device to construct a new function $f^{(m)} : x_n^{(m)} \in \mathbb{R}^D \longrightarrow y_n^{(m)} \in \mathbb{R}$. Note that, only the parameters related to patches are used for the model personalization and updated based on the local dataset.

As summarized in previous sections, the works regarding *on-device inference* and those related to *on-device learning for the centralized model* have been explored in an independent perspective by means of the pervasive computing advantages. There are few works that consider how to make the device modeling and the cloud modeling to benefit both sides jointly in the recommendation scenarios. However, this is critical and meaningful, since the conventional centralized cloud model follows the "model-over-data" paradigm and is prone to Matthew effect [32]. On the other side, personalized models updated with relatively limited samples on the device side suffers from the local optimum [10], which can be calibrated by the centralized cloud model. We introduce the DCCL to bridge the gap, which customizes the personalization on the device side and enhances the centralized model with the "model-over-models" distillation over billions of personalized device models. Figure 1 gives the general illustration of the DCCL framework, however, real-world challenges make it not straightforward for implementation. In the following, we will discuss the practical challenges and deployment of each side in DCCL.

### 3.2 MetaPatch for On-device Personalization

Although the device hardware has been greatly improved in the recent years, compared to the rich computing power, energy and storage in the cloud, it is still resource-constrained to learn a complete big model on the device. Considering the parameter scale of the centralized cloud model $f$ and the storage of the intermediate activations, it is an impractical choice to adapt the whole network on the device side. Meanwhile, only finetuning last few layers is performance-limited due to the feature basis of the pretrained layers. Fortunately, some previous works have demonstrated that it is possible to achieve the comparable performance as the whole network finetuning via patch learning [4, 17, 46]. Inspired by these works, we insert the model patches on basis of the cloud model $f$ for on-device personalization. Formally, the output of the $l$-th layer attached with one patch on the $m$-th device is expressed as

$$f_l^{(m)}(\cdot) = f_l(\cdot) + \underbrace{h_l^{(m)}(\cdot)}_{\text{patch}} \circ f_l(\cdot), \tag{1}$$

where LHS of Eq.(1) is the sum of the original $f_l(\cdot)$ and the patch response of $f_l(\cdot)$. Here, $h_l^{(m)}(\cdot)$ is the trainable patch function and $\circ$ denotes the function composition that treats the output of the previous function as the input. This construction facilitates that we can set a manual gate to mask the patch response and degenerate to the on-device inference with the centralized cloud model, anytime we want to ease the adaptation effect. Note that, the model patch could have different neural architectures. Here, we do not explore its variants but specify the same bottleneck architecture like [17].

Nevertheless, we empirically find that the parameter space of multiple patches is still relatively too large to fit the sparse local samples. To overcome this issue, we propose a novel method *MetaPatch* to reduce the parameter space. It is a kind of meta learning methods to generate parameters [13, 20], which shares the global parameter basis to reduce the parameters to be learned, thus more suits the on-device personalization. Concretely, assume the parameters of each patch are denoted by $\theta_l^{(m)} \in \mathbb{R}^{K_l}$ (flatten all parameters in the patch into a vector). Then, we can deduce the following decomposition

$$\theta_l^{(m)} = \Theta_l * \hat{\theta}^{(m)}, \tag{2}$$

where $\Theta_l \in \mathbb{R}^{K_l \times \hat{K}}$ is the globally shared parameter basis (freezing it on the device and learned in the cloud) and $\hat{\theta}^{(m)} \in \mathbb{R}^{\hat{K}}$ is the

surrogate tunable parameter vector to generate each patch parameter $\theta_l^{(m)}$ in the device-model $f^{(m)}$. To facilitate the understanding, we term $\hat{\theta}^{(m)}$ as the metapatch parameter. In this paper, we keep $\sum_l K_l \gg \hat{K}$ so that the metapatch parameters to be learned for personalization are greatly reduced. Figure 2 illustrates the idea of *MetaPatch*. Note that, regarding the pretraining of $\Theta_l$, we leave the discussion in the following section to avoid the clutter, since it is learned on the cloud side. According to Eq. (2), we implement the patch parameter generation via the metapatch parameter $\hat{\theta}^{(m)}$ instead of directly learning $\theta^{(m)}$. To learn the metapatch parameter, we can leverage the local dataset to minimize the following loss.

$$\min_{\hat{\theta}^{(m)}} \ell(y, \tilde{y})\big|_{\tilde{y}=f^{(m)}(x)}, \tag{3}$$

where $\ell$ is the pointwise cross-entropy loss, $f^{(m)}(\cdot) = f_L^{(m)}(\cdot) \circ \cdots f_l^{(m)}(\cdot) \cdots \circ f_1^{(m)}(\cdot)$ and $L$ is the number of total layers. After training the device specific parameter $\hat{\theta}^{(m)}$ by Eq.(3), we can use Eq. (2) to generate all patches, and then insert them into the cloud network $f$ via Eq.(1) to get the final personalized model $f^{(m)}$, which will provide the on-device personalized recommendation.

### 3.3 MoMoDistill to Enhance the Cloud Modeling

The conventional incremental training of the centralized cloud model follows the "model-over-data" paradigm. That is, when the new training samples are collected from devices, we directly perform the incremental learning based on the model trained in the early sample collection. The objective function is formulated as

$$\min_{\mathbf{W}_f} \ell(y, \hat{y})\big|_{\hat{y}=f(x)}, \tag{4}$$

where $\mathbf{W}_f$ is the network parameter of the cloud model $f$ to be trained. This is an independent perspective without considering the device modeling. However, the on-device personalization actually can be more powerful than the centralized cloud model to handle the corresponding local samples. Thus, the guidance from the on-device models could be a meaningful prior to help the cloud modeling. Inspired by this, we propose a "model-over-models" paradigm to simultaneously learn from data and aggregate the knowledge from on-device models, to enhance the training of the centralized cloud model. Besides, considering that on-device learning in a long term suffers from local optimization [10], this step also helps calibrate the global model $f$ for the better on-device personalization. Formally, the objective with the distillation procedure on the samples from all devices is defined as,

$$\min_{\mathbf{W}_f} \ell(y, \hat{y}) + \beta\, \mathrm{KL}(\tilde{y}, \hat{y})\big|_{\hat{y}=f(x), \tilde{y}=f^{(m)}(x)}, \tag{5}$$

where $\beta$ is the hyperparameter to balance the distillation and "model-over-data" learning. Note that, the feasibility of the distillation in Eq. (5) critically depends on the patch mechanism in the previous section, since it allows us to input the metapatch parameters like features with only loading the other model parameters of $f^{(m)}$ in one time. Otherwise, we will suffer from the engineering issue of reloading numerous checkpoints frequently, which is almost impossible for current open source frameworks like Tensorflow.

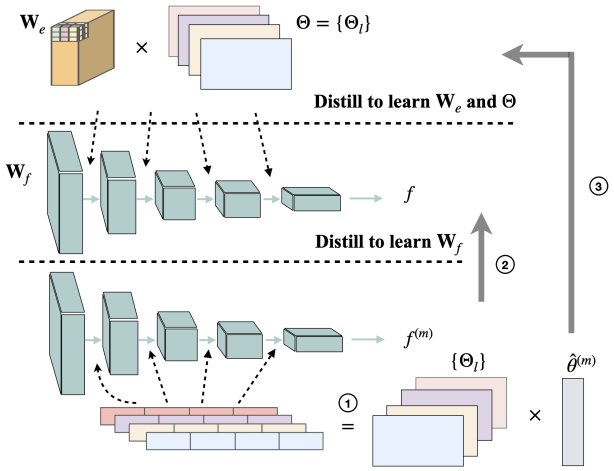


**Figure 3: The proposed *MoMoDistill* to enhance the cloud modeling. ① It leverages Eq. (2) to compute the patch parameters and then constructs $\{f^{(m)}\}$. ② Following Eq. (5), we learn the centralized cloud model $f$ by model-over-models distillation from the on-device models $\{f^{(m)}\}$. ③ Fixing $f$, optimize Eq. (7) to learn the global parameter basis $\Theta$.**

In *MetaPatch*, we introduce the global parameter basis $\{\Theta_l\}$ (simplified by $\Theta$) to reduce the parameter space on the device. Regarding its training, we empirically find that coupled learning with $\mathbf{W_f}$ easily falls into undesirable local optimal, since they play different roles in terms of their semantics. Therefore, we resort to a progressive optimization strategy, that is, first optimize f based on Eq. (5), and then distill the knowledge for the parameter basis $\Theta$ with the learned $f$. For the second step, we design an auxiliary component by considering the heterogeneous characteristics of the metapatches from all devices and the cold-start issue at the beginning. Concretely, given the dataset $\{(x, y, u^{(I(x))}, \hat{\theta}^{(I(x))})\}_{n=1,\ldots,N}$, where $I$ maps the sample index to the device index and $u \subset x$ is the user profile features (*e.g.*, age, gender, purchase level, etc) of the corresponding device, we define the following auxiliary encoder,

$$U(\hat{\theta}, u) = \mathbf{W}^{(1)} \tanh(\mathbf{W}^{(2)} \hat{\theta} + \mathbf{W}^{(3)} u), \tag{6}$$

where $\mathbf{W}^{(1)} \in \mathbb{R}^{\hat{K}_l \times \hat{K}_l}$, $\mathbf{W}^{(2)} \in \mathbb{R}^{\hat{K}_l \times \hat{K}_l}$, $\mathbf{W}^{(3)} \in \mathbb{R}^{\hat{K}_l \times d_u}$ are tunable projection matrices, $d_u$ is the dimension of user profile features. Here, we use $\mathbf{W_e}$ denotes the collection $\{\mathbf{W}^{(1)}, \mathbf{W}^{(2)}, \mathbf{W}^{(3)}\}$ for simplicity. To learn the global parameter basis, we replace $\hat{\theta}$ by $U(\hat{\theta}, u)$ to simulate Eq. (2) to generate the model patch, *i.e.*, $\Theta * U(\hat{\theta}, u)$, since actually $\hat{\theta}$ is too heterogeneous to be directly used. Then, combining $\Theta * U(\hat{\theta}, u)$ with $f$ learned in the first distillation step, we can form a new proxy device model $\hat{f}^{(m)}$ (different from $f^{(m)}$ in the patch generation). Here, we leverage such a proxy $\hat{f}^{(m)}$ to directly distill the knowledge from the true $f^{(m)}$ from devices, which optimizes $\Theta$ and $\mathbf{W_e}$ of the auxiliary encoder as follows,

$$\min_{(\Theta, \mathbf{W}_e)} \ell(y, \hat{y}) + \beta\, \mathrm{KL}(\tilde{y}, \hat{y})\big|_{\hat{y}=\hat{f}^{(m)}(x), \tilde{y}=f^{(m)}(x)}, \tag{7}$$

**Algorithm 1:** Device-Cloud Collaborative Learning

Pretrain the cloud model $f$, and then learn the global parameter basis $\Theta$ based on Eq. (7) by setting $\hat{\theta}$ as 0.
**while** *lifecycle* **do**
  Send $f$ and $\Theta$ to devices.
  **Device**($f$, $\Theta$): ▷ *MetaPatch*
    1) Accumulate the local data into batches
    2) On-device personalization via Eq.(3)
    3) If time > threshold: upload $\hat{\theta}^{(m)}$
    4) Else: return step 1)
  Recycle all model patches $\{\hat{\theta}^{(m)}\}$.
  **Cloud**($\{\hat{\theta}^{(m)}\}$): ▷ *MoMoDistill*
    1) Optimize the cloud model $f$ based on Eq.(5)
    2) Learn the parameter basis $\Theta$ by Eq.(7)
**end**

Eq. (5) and Eq. (7) progressively help learn the centralized cloud model and the global parameter basis. We specially term this progressive distillation mechanism as *MoMoDistill* to emphasize our "model-over-models" paradigm different from the conventional "model-over-data" incremental training on the cloud side, and gives the illustration in Figure 3. Finally, in Algorithm 1, we summarize the complete procedure of DCCL with *MetaPatch* and *MoMoDistill*.

## 4 EXPERIMENTS

In this section, we will conduct a range of recommendation experiments to demonstrate the effectiveness of the proposed framework. To be specific, we will answer the following questions about DCCL.

(1) **RQ1:** Whether learning through the proposed framework can achieve a better performance compared with the conventional centralized incremental training in the cloud? The challenges come into two folds. First, we need to train the personalized models for each device. The detailed results of this part will be analysed in RQ2. Second, it is challenging to aggregate the knowledge from plenty of weakly heterogeneous on-device model "experts", which has never been explored before.
(2) **RQ2:** Whether on-device model personalization can achieve further improvement than the centralized cloud model? The sparse data flow on the local devices can be very challenging to learn and easy to overfit. To our best knowledge, this has never been investigated on the large-scale industrial recommendation datasets at device granularity.
(3) **RQ3:** How is the convergence property regarding the their inter-loops between two modules and is it capable to maintain a cyclical process which iterates and optimizes between the cloud and the devices? It is important to characterize the long-term performance of our method, as once deployed, this framework will continually maintain such a learning cycle, which is a novel trial in the area of recommender systems.

### 4.1 Experimental Setup

*4.1.1 Datasets.* Our experiments are implemented on three recommendation datasets Amazon, Movielens-1M and Taobao. The statistics of all the above datasets are shown in Table 1. Generally, all these three datasets are user interactive histroy in sequence format , and the last user interacted item is cut out as test sample. For each last interacted item, we randomly sample 100 items that didn't appear in user's history. Detailed description about the data generation process is given in Appendix section.

**Table 1: Statistics of datasets used in this paper.**

| Dataset | Amazon | MovieLens-1M | Taobao |
|---|---|---|---|
| Users | 360,828 | 6,040 | 17,018,570 |
| Goods | 68,000 | 3,900 | 20,000,000 |
| Categories | 257 | 21 | 14,641 |
| Samples | 3,087,820 | 1,000,209 | ~9 billions |

*4.1.2 Baselines & Our methods.* We compare DCCL with some classical cloud models in different perspectives, namely, the conventional methods, deep learning-based methods and sequence-based methods.

- **Conventional methods:**
  - **MF** [24]: The matrix factorization approach to model user-item interactions, which decomposes the observation as the product of the user embedding and the item embedding.
  - **FM** [35]: Factorization Machines (FM) maps real-valued features into a low-dimensional latent space, and models all interactions between variables using factorized parameters.
- **Deep learning-based methods:**
  - **NeuMF** [16]: It generalizes the classifical MF into a deep-learning counterpart. It uses multiple neural layers to build a more expressive distance measure instead of inner product to capture the non-linearity in the implicit feedbacks.
  - **DeepFM** [12]: A wide & deep architecture that incorporates the factorization machine to automatically extract the wide features instead of the previous hand-crafted effort.
- **Sequence-based methods:**
  - **SASRec** [21]: A self-attention based sequential model, which utilizes the attention mechanism to adaptively assign weights to previous items for the next-item prediction.
  - **DIN** [49]: A target-attention based user interest model, which takes the candidates as the query *w.r.t.* the user historical behaviors to learn a user representation for prediction.

For the whole experiments, we implement our model on the basis of **DIN**, where we insert the model patches in the last second fully-connected layer and the first two fully-connected layers after the feature embedding layer. In all comparisons, we term *MetaPatch* as **DCCL-e**, and *MoMoDistill* as **DCCL-m**, since the whole framework resembles EM[4] iterations. The default method to compare the baselines is named **DCCL**, which indicates that it goes through both on-device personalization and the "model-over-models" distillation.

[4]https://en.wikipedia.org/wiki/Expectation-maximization_algorithm

**Table 2: The top-K recommendation performance of DCCL and baselines on three datasets.**

| Datasets | Metric | MF | FM | NeuMF | DeepFM | SASRec | DIN | DCCL | Improv. |
|---|---|---|---|---|---|---|---|---|---|
| Amazon | HitRate@1 | 23.69 | 21.53 | 26.10 | 25.43 | 26.53 | 26.56 | **26.94** | 1.43% |
| | HitRate@5 | 35.74 | 36.74 | 42.98 | 42.48 | 44.22 | 44.00 | **44.79** | 1.29% |
| | HitRate@10 | 44.38 | 47.90 | 52.32 | 53.51 | 54.94 | 55.43 | **56.59** | 2.09% |
| | NDCG@5 | 29.83 | 29.17 | 34.74 | 34.12 | 35.60 | 35.48 | **36.95** | 3.79% |
| | NDCG@10 | 32.61 | 32.77 | 37.76 | 37.67 | 39.07 | 39.22 | **40.45** | 3.14% |
| MovieLens-1M | HitRate@1 | 14.60 | 14.90 | 16.45 | 15.41 | 34.85 | 37.45 | **38.69** | 3.31% |
| | HitRate@5 | 44.85 | 47.13 | 46.24 | 47.35 | 69.17 | 70.71 | **71.97** | 1.78% |
| | HitRate@10 | 63.54 | 64.40 | 65.36 | 65.46 | 80.69 | 81.25 | **82.23** | 1.21% |
| | NDCG@5 | 29.87 | 30.27 | 31.71 | 31.82 | 53.18 | 55.22 | **56.43** | 2.19% |
| | NDCG@10 | 35.89 | 36.49 | 37.90 | 37.68 | 56.94 | 58.65 | **59.77** | 1.91% |
| Taobao | HitRate@1 | 24.88 | 25.29 | 29.11 | 33.28 | 35.19 | 52.17 | **55.71** | 6.79% |
| | HitRate@5 | 50.83 | 51.18 | 55.42 | 57.26 | 60.13 | 68.12 | **70.31** | 3.21% |
| | HitRate@10 | 62.28 | 63.96 | 65.78 | 66.09 | 69.30 | 74.80 | **76.70** | 2.54% |
| | NDCG@5 | 38.46 | 38.80 | 43.03 | 46.09 | 48.52 | 60.65 | **63.42** | 4.57% |
| | NDCG@10 | 42.17 | 42.93 | 46.40 | 48.95 | 51.50 | 62.81 | **65.49** | 4.27% |

*4.1.3 Evaluation Metrics.* The model performance in our experiments are measured by the widely used AUC and NDCG . They are respectively calculated by the following equations.

$$\begin{aligned}
\text{HitRate@}K &= \frac{1}{|\mathcal{U}|}\sum_{u\in\mathcal{U}} \mathbb{1}(R_{u,g_u} \le K),\\
\text{NDCG@}K &= \sum_{u\in\mathcal{U}} \frac{1}{|\mathcal{U}|}\frac{2^{\mathbb{1}(R_{u,g_u}\le K)}-1}{\log_2(\mathbb{1}(R_{u,g_u}\le K)+1)},\\
\text{macro-AUC} &= \sum_{u\in\mathcal{U}} \frac{1}{|\mathcal{U}|}\frac{\sum_{x_0\in D_T^{(u)}}\sum_{x_1\in D_F^{(u)}}\mathbf{1}[f(x_1)<f(x_0)]}{|D_T^{(u)}||D_F^{(u)}|},
\end{aligned} \tag{8}$$

where $\mathcal{U}$ is the user set, $\mathbb{1}(\cdot)$ is the indicator function, $R_{u,g_u}$ is the rank generated by the model for the ground truth item $g_u$ and user $u$, $f$ is the model to be evaluated and $D_T^{(u)}$, $D_F^{(u)}$ is the positive and negative sample sets in testing data.

## 4.2 Experiments and Results

*4.2.1 RQ1.* To demonstrate the effectiveness of DCCL, we conduct the experiments on Amazon, Movielens and Taobao to compare to a range of baselines. Aligned with the popular experimental settings [16, 49], the last interactive item of each user on three datasets is left for evaluation and all items before the last one are used for training. For DCCL, we split the training data into two parts on average according to the temporal order: one part is for the pretraining of the backbone (DIN) and the other part is for the training of DCCL. In the experiments, we conduct one-round DCCL-e and DCCL-m. Finally, the DCCL-m is compared with the six representative models, whose results are summarized in Table 2.

According to Table 2, we find that the deep learning based methods NeuMF and DeepFM usually outperform the conventional methods MF and FM, and the sequence-based methods SASRec and DIN consistently outperform previous non-sequence-based methods. Our DCCL builds upon on the best baseline DIN and further improves its results. Specifically, DCCL shows about 2% or more improvements in terms of NDCG@10, and at least 1% improvements in terms of HitRate@10 on all three datasets. The performance on both small and large datasets confirms the superiority of DCCL.

*4.2.2 RQ2.* In this section, we target to demonstrate that how on-device personalization via *MetaPatch* (abbreviated as DCCl-e) can improve the recommendation performance from different levels of users compared with the centralized cloud model. Considering the data scale and the availability of the context information for visualization, only the Taobao dataset is used to conduct this experiment. To validate the performance of DCCL-e in the fine-grained granularity, we sort the users based on their sample numbers and then partition them into 20 groups on average along the sorted user axis (see the statistic of the sample number *w.r.t.* the user in the appendix). After on-device model personalization, we calculate the performance for each group based on the personalized models. Here, the macro-AUC is used, which equally treats the users in the group instead of the sample-number-aware group AUC [49].

We use DIN as baseline in this experiment and pretrain it on the Taobao Dataset of the first 20 days. Then, we test the model in the data of the remaining 10 days. For DCCL-e, we first pretrain DIN on the Taobao Dataset of the first 10 days, and then insert the patches into the pretrained DIN same as previous settings. Finally, we perform the on-device personalization in the subsequent 10 days. Similarly, we test the DCCL-e on the data of the last 10 days. The evaluation is respectively conducted in the 20 groups and their results are plot in Figure 4. According to the results, we can find that with the increase of the group index number, the performance approximately decreases. This is because the users in the group of larger indices are more like the long-tailed users based on our group partition, and their patterns are easily ignored or even sacrificed by the centralized cloud model. In comparison, DCCL-e shows the consistent improvement over DIN on all groups, and especially achieve the large improvement in the long-tailed user groups.

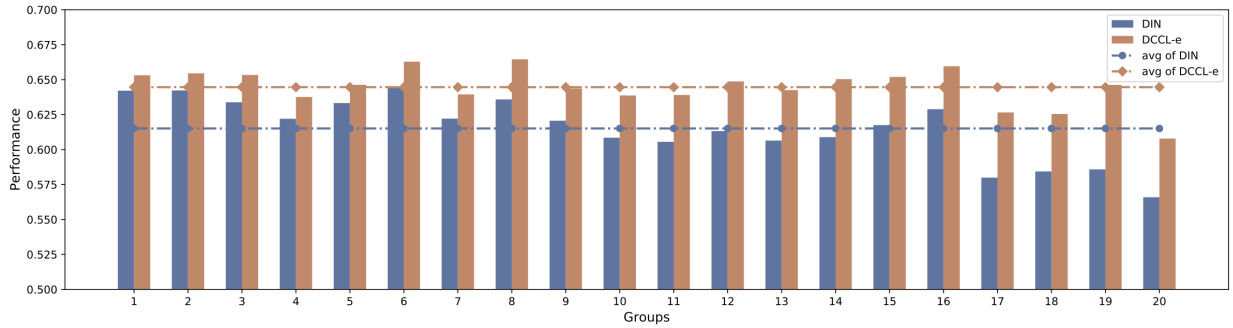


**Figure 4: The macro-AUC of on-device personalization *v.s.* DIN in all user groups. The performance of DIN in the head user groups are approximately better than that in the long-tailed user groups. In comparsion, DCCL-e performs better than DIN in all groups and specifically, achieves the larger improvement in the long-tailed user groups.**

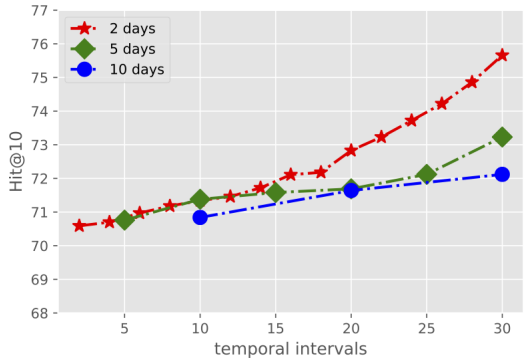


**Figure 5: The convergence property of DCCL in three different temporal intervals of each round on the Taobao dataset.**

*4.2.3 RQ3.* To illustrate the convergence property of DCCL, we conduct the experiments on the Taobao dataset in different device-cloud interaction temporal intervals. Concretely, we specify every 2, 5, 10 days interactions between device and cloud, and respectively trace the performance of each round evaluated on the last click of each user. Figure 5 illustrates the convergence procedure of DCCL in different intervals. According to Figure 5, we observe that frequent interactions achieve much better performance than the infrequent counterparts. We speculate that, as *MeatPatch* and *MoMoDistill* could promote each other at every round, the advantages in performance have been continuously strengthened with more frequent interactions. However, the side effect is we have to frequently update the on-device models, which may introduce other uncertain crash risks. Thus, in the real-world scenarios, we still need to make a trade-off between performance and the interaction interval in the cold-start.

*4.2.4 Ablation and Case Study.* In this section, we present more analysis of DCCL about *MoMoDistill* and *MetaPatch*. Then, we will exhibit a case study to illustrate the difference of recommendation results between DCCL and DIN for the long-tailed users.

For the first study, we give the results of *MetaPatch* and *MoMoDistill* in one-round DCCL on the Taobao dataset and compare with DIN in Table 3. From the results, we can observe the progressive improvement after DCCL-e and DCCL-m, and DCCL-e acquires more benefit than DCCL-m in terms of the improvement. The revenue behind DCCL-e is *MetaPatch* customizes a peronsalized model for each user to improve their recommendation experience once new behavior logs are collected on device, without the delayed update from the centralized cloud server. Regarding DCCL-m, as stated before, the local samples could be scarce and noisy. Therefore, the lengthy on-device personalization will be easy to be trapped by the limited knowledge of each user and result in the local optimal for the recommendation model. The further improvements from DCCL-m confirm the necessity of *MoMoDistill* to re-calibarate the backone and the parameter basis. However, if we conduct the experiments without our two modules, the model performance is as DIN, which is not better than both DCCL-e and DCCL-m.

**Table 3: The results of one-round DCCL compared to DIN.**

| Method | DIN | DCCL-e | DCCL-m |
|---|---|---|---|
| HitRate@1 | 52.17 | 55.03 | **55.71** |
| HitRate@5 | 68.12 | 70.03 | **70.31** |
| HitRate@10 | 74.80 | 76.46 | **76.70** |
| NDCG@5 | 60.65 | 62.99 | **63.42** |
| NDCG@10 | 62.81 | 65.07 | **65.49** |

**Table 4: One-round DCCL with different positional patches.**

| Position | 1st Junction | 2nd Junction | 3rd Junction |
|---|---|---|---|
| HitRate@1 | 53.26 | **54.10** | 52.36 |
| HitRate@5 | 68.89 | **69.36** | 68.14 |
| HitRate@10 | 75.54 | **75.86** | 74.85 |
| NDCG@5 | 61.56 | **62.19** | 60.74 |
| NDCG@10 | 63.71 | **64.29** | 62.92 |

For the second ablation study, we explore the effect of the model patches in different layer junctions. As claimed in previous sections, we insert two patches (1st Junction, 2nd Junction) in the two fully-connected layers respectively after the feature embedding layer,


**Figure 6: The case study to compare the recommendation of DIN, DCCL-e and DCCL-m for one long-tailed user. The left hand side presents five items that user 1 clicked and the right hand side gives the top-5 recommendation from three methods.**

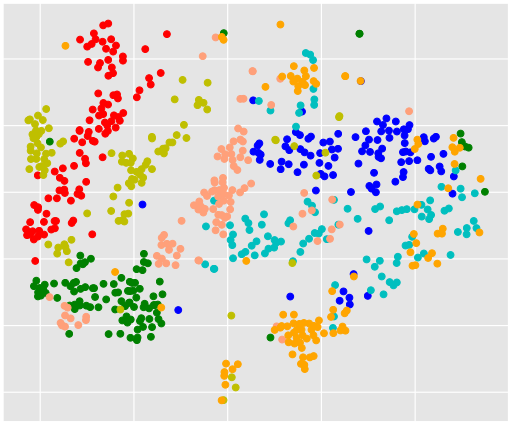

**Figure 7: T-SNE visualization of the metapatches. The color is the representative Level-1 category to train metapatches.**

and one patch (3rd Junction) to the layer before the last softmax transformation layer. In this experiment, we validate their effectiveness by only keep each of them in one-round DCCL. Their results on the Taobao dataset are summarized in Table 4. Compared with the full model in Table 3, we can find that removing the model patch would decrease the performance. And the results suggest that the model patches in the 1st and 2nd juncitons are more effective than the higher layer (the 3rd junction). The intuition is that the model patches in low layers more easily adjust the item contributions, which highlights or downweights items according to the user interests, resulting in a more expressive personalization.

To visualize the metapatches learned from on-device personalization, we project their parameter vectors $\{\hat{\theta}^{(m)}\}$ into the 2-D space via t-SNE[5] as shown in Figure 7. For a better visualization, we assign each point with the color representing the most frequent level-1 category in the local samples to train the metapatch. Note that, it does not mean the true label but we conjecture that the optimization direction of the metapatch is possibly correlated with the sample categories. From Figure 7, we can find that the points belonging to the same color approximately form into a cluster. This might indicate they share similar on-device models to some extent, which inherently keeps the consistency with motivation of learning the global parameter basis $\Theta$ to reduce the parameter space.

To visualize the difference between the centralized cloud model and DCCL, we illustrate one case study based on DIN, DCCL-e and DCCL-m in Figure 6. Specifically, one long-tailed user is chosen to validate their effectiveness. As shown in Figure 6, although the user has clicked many items from the bed categories, DIN has not recommend the expected items related to his history. It is because the bed category is a minority compared to other daily supplies, and the centralized cloud model are optimized bias to this pattern, which is prone to recall other supplies. In comparison, DCCL-e that has adapted to the historical behaviors of this user recommends more relevant items of this category. Nevertheless, it is not perfect to ease the bias effect of the pretrained cloud model and also introduces some bad cases like the last sofa item. In comparison, DCCL-m that refines the global parameter basis further alleviates this issue and recommends all items to the user historical behaviors.

## 5 CONCLUSION

In this paper, we focus on the general mobile-cloud collaboration scenarios, and propose a Device-Cloud Collaborative Learning framework to explore mutual benefit of the mobile and the cloud modeling. Specifically, we introduce a novel *MetaPatch* approach to efficiently achieve the model personalization for each device, and simultaneously introduce a *MoMoDistill* enhance the cloud modeling via the knowledge distillation from the personalized on-device models. Extensive experiments on a range of datasets demonstrate the superiority of DCCL over state-of-the-art methods. However, as an initial exploration about the collaboration between the device and the cloud modeling, more works need to be contributed to on-device intelligence and solving the general challenges about the data sparsity, noise and other computational constraints.

## REFERENCES

[1] Guneet Bedi, Ganesh Kumar Venayagamoorthy, Rajendra Singh, Richard R Brooks, and Kuang-Ching Wang. 2018. Review of Internet of Things (IoT) in electric power and energy systems. *IEEE Internet of Things Journal* 5, 2 (2018), 847–870.

[2] Ilai Bistritz, Ariana Mann, and Nicholas Bambos. 2020. Distributed Distillation for On-Device Learning. *Advances in Neural Information Processing Systems* 33 (2020).

[3] Han Cai, Chuang Gan, Tianzhe Wang, Zhekai Zhang, and Song Han. 2020. Once for All: Train One Network and Specialize it for Efficient Deployment. In *ICLR*.

[5] https://en.wikipedia.org/wiki/T-distributed_stochastic_neighbor_embedding

[4] Han Cai, Chuang Gan, Ligeng Zhu, and Song Han. 2020. TinyTL: Reduce Memory, Not Parameters for Efficient On-Device Learning. *Advances in Neural Information Processing Systems* 33 (2020).

[5] Xu Chen, Ya Zhang, Ivor Tsang, Yuangang Pan, and Jingchao Su. 2020. Towards Equivalent Transformation of User Preferences in Cross Domain Recommendation. *arXiv preprint arXiv:2009.06884* (2020).

[6] Heng-Tze Cheng, Levent Koc, Jeremiah Harmsen, Tal Shaked, Tushar Chandra, Hrishi Aradhye, Glen Anderson, Greg Corrado, Wei Chai, Mustafa Ispir, et al. 2016. Wide & deep learning for recommender systems. In *Proceedings of the 1st workshop on deep learning for recommender systems*. 7–10.

[7] Kenan Cui, Xu Chen, Jiangchao Yao, and Ya Zhang. 2018. Variational collaborative learning for user probabilistic representation. *Workshop of the AAAI conference on artificial intelligence* (2018).

[8] Xiangfeng Dai, Irena Spasić, Bradley Meyer, Samuel Chapman, and Frederic Andres. 2019. Machine learning on mobile: An on-device inference app for skin cancer detection. In *2019 Fourth International Conference on Fog and Mobile Edge Computing (FMEC)*. IEEE, 301–305.

[9] Amir Erfan Eshratifar, Mohammad Saeed Abrishami, and Massoud Pedram. 2019. JointDNN: an efficient training and inference engine for intelligent mobile cloud computing services. *IEEE Transactions on Mobile Computing* (2019).

[10] Chelsea Finn, Pieter Abbeel, and Sergey Levine. 2017. Model-agnostic meta-learning for fast adaptation of deep networks. *ICML* (2017).

[11] Yu Gong, Ziwen Jiang, Yufei Feng, Binbin Hu, Kaiqi Zhao, Qingwen Liu, and Wenwu Ou. 2020. EdgeRec: Recommender System on Edge in Mobile Taobao. In *Proceedings of the 29th ACM International Conference on Information & Knowledge Management*. 2477–2484.

[12] Huifeng Guo, Ruiming Tang, Yunming Ye, Zhenguo Li, and Xiuqiang He. 2017. DeepFM: a factorization-machine based neural network for CTR prediction. In *Proceedings of the 26th International Joint Conference on Artificial Intelligence*. 1725–1731.

[13] David Ha, Andrew Dai, and Quoc V Le. 2017. Hypernetworks. *ICLR* (2017).

[14] Song Han, Huizi Mao, and William J Dally. 2016. Deep compression: Compressing deep neural networks with pruning, trained quantization and huffman coding. *ICLR* (2016).

[15] Song Han, Jeff Pool, John Tran, and William Dally. 2015. Learning both weights and connections for efficient neural network. *Advances in neural information processing systems* 28 (2015), 1135–1143.

[16] Xiangnan He, Lizi Liao, Hanwang Zhang, Liqiang Nie, Xia Hu, and Tat-Seng Chua. 2017. Neural collaborative filtering. In *Proceedings of the 26th international conference on world wide web*. 173–182.

[17] Neil Houlsby, Andrei Giurgiu, Stanislaw Jastrzebski, Bruna Morrone, Quentin De Laroussilhe, Andrea Gesmundo, Mona Attariyan, and Sylvain Gelly. 2019. Parameter-efficient transfer learning for NLP. In *International Conference on Machine Learning*. PMLR, 2790–2799.

[18] Andrew G Howard, Menglong Zhu, Bo Chen, Dmitry Kalenichenko, Weijun Wang, Tobias Weyand, Marco Andreetto, and Hartwig Adam. 2017. Mobilenets: Efficient convolutional neural networks for mobile vision applications. *ICLR* (2017).

[19] Dietmar Jannach and Malte Ludewig. 2017. When recurrent neural networks meet the neighborhood for session-based recommendation. In *Proceedings of the Eleventh ACM Conference on Recommender Systems*. 306–310.

[20] Xu Jia, Bert De Brabandere, Tinne Tuytelaars, and Luc V Gool. 2016. Dynamic filter networks. *Advances in neural information processing systems* 29 (2016), 667–675.

[21] Wang-Cheng Kang and Julian McAuley. 2018. Self-attentive sequential recommendation. In *2018 IEEE International Conference on Data Mining (ICDM)*. IEEE, 197–206.

[22] Sai Praneeth Karimireddy, Satyen Kale, Mehryar Mohri, Sashank J Reddi, Sebastian U Stich, and Ananda Theertha Suresh. 2020. Scaffold: Stochastic controlled averaging for on-device federated learning. *ICML* (2020).

[23] Jakub Konečný, H. Brendan McMahan, Felix X. Yu, Peter Richtarik, Ananda Theertha Suresh, and Dave Bacon. 2016. Federated Learning: Strategies for Improving Communication Efficiency. In *ICLR*.

[24] Yehuda Koren, Robert Bell, and Chris Volinsky. 2009. Matrix factorization techniques for recommender systems. *Computer* 42, 8 (2009), 30–37.

[25] Viraj Kulkarni, Milind Kulkarni, and Aniruddha Pant. 2020. Survey of Personalization Techniques for Federated Learning. *arXiv preprint arXiv:2003.08673* (2020).

[26] Juhyun Lee, Nikolay Chirkov, Ekaterina Ignasheva, Yury Pisarchyk, Mogan Shieh, Fabio Riccardi, Raman Sarokin, Andrei Kulik, and Matthias Grundmann. 2019. On-device neural net inference with mobile gpus. *arXiv preprint arXiv:1907.01989* (2019).

[27] Yujie Lin, Pengjie Ren, Zhumin Chen, Zhaochun Ren, Dongxiao Yu, Jun Ma, Maarten de Rijke, and Xiuzhen Cheng. 2020. Meta Matrix Factorization for Federated Rating Predictions. In *Proceedings of the 43rd International ACM SIGIR Conference on Research and Development in Information Retrieval*. 981–990.

[28] Yan Lu, Yuanchao Shu, Xu Tan, Yunxin Liu, Mengyu Zhou, Qi Chen, and Dan Pei. 2019. Collaborative learning between cloud and end devices: an empirical study on location prediction. In *Proceedings of the 4th ACM/IEEE Symposium on Edge Computing*. 139–151.

[29] Khalil Muhammad, Qinqin Wang, Diarmuid O'Reilly-Morgan, Elias Tragos, Barry Smyth, Neil Hurley, James Geraci, and Aonghus Lawlor. 2020. Fedfast: Going beyond average for faster training of federated recommender systems. In *Proceedings of the 26th ACM SIGKDD International Conference on Knowledge Discovery & Data Mining*. 1234–1242.

[30] Chaoyue Niu, Fan Wu, Shaojie Tang, Lifeng Hua, Rongfei Jia, Chengfei Lv, Zhihua Wu, and Guihai Chen. 2020. Billion-scale federated learning on mobile clients: A submodel design with tunable privacy. In *Proceedings of the 26th Annual International Conference on Mobile Computing and Networking*. 1–14.

[31] Yoon-Joo Park and Alexander Tuzhilin. 2008. The long tail of recommender systems and how to leverage it. In *Proceedings of the 2008 ACM conference on Recommender systems*. 11–18.

[32] Matjaž Perc. 2014. The Matthew effect in empirical data. *Journal of The Royal Society Interface* 11, 98 (2014), 20140378.

[33] Tao Qi, Fangzhao Wu, Chuhan Wu, Yongfeng Huang, and Xing Xie. 2020. Privacy-Preserving News Recommendation Model Learning. In *Proceedings of the 2020 Conference on Empirical Methods in Natural Language Processing: Findings*. 1423–1432.

[34] Zhiwei Rao, Jiangchao Yao, Ya Zhang, and Rui Zhang. 2016. Preference aware recommendation based on categorical information. In *2016 15th IEEE International Conference on Machine Learning and Applications (ICMLA)*. IEEE, 865–870.

[35] Steffen Rendle. 2010. Factorization machines. In *2010 IEEE International Conference on Data Mining*. IEEE, 995–1000.

[36] Paul Resnick and Hal R Varian. 1997. Recommender systems. *Commun. ACM* 40, 3 (1997), 56–58.

[37] Badrul Sarwar, George Karypis, Joseph Konstan, and John Riedl. 2001. Item-based collaborative filtering recommendation algorithms. In *Proceedings of the 10th international conference on World Wide Web*. 285–295.

[38] Mahadev Satyanarayanan. 2017. The emergence of edge computing. *Computer* 50, 1 (2017), 30–39.

[39] Guy Shani, David Heckerman, and Ronen I Brafman. 2005. An MDP-based recommender system. *Journal of Machine Learning Research* 6, Sep (2005), 1265–1295.

[40] Ion Stoica, Dawn Song, Raluca Ada Popa, David Patterson, Michael W Mahoney, Randy Katz, Anthony D Joseph, Michael Jordan, Joseph M Hellerstein, Joseph E Gonzalez, et al. 2017. A berkeley view of systems challenges for ai. *arXiv preprint arXiv:1712.05855* (2017).

[41] Yang Sun, Fajie Yuan, Ming Yang, Guoao Wei, Zhou Zhao, and Duo Liu. 2020. A Generic Network Compression Framework for Sequential Recommender Systems. *SIGIR* (2020).

[42] Prahalathan Sundaramoorthy, Gautham Krishna Gudur, Manav Rajiv Moorthy, R Nidhi Bhandari, and Vineeth Vijayaraghavan. 2018. Harnet: Towards on-device incremental learning using deep ensembles on constrained devices. In *Proceedings of the 2nd International Workshop on Embedded and Mobile Deep Learning*. 31–36.

[43] Qiaoyu Tan, Jianwei Zhang, Jiangchao Yao, Ninghao Liu, Jingren Zhou, Hongxia Yang, and Xia Hu. 2021. Sparse-interest network for sequential recommendation. In *Proceedings of the 14th ACM International Conference on Web Search and Data Mining*. 598–606.

[44] Qiang Yang, Yang Liu, Tianjian Chen, and Yongxin Tong. 2019. Federated machine learning: Concept and applications. *ACM Transactions on Intelligent Systems and Technology (TIST)* 10, 2 (2019), 1–19.

[45] Jiangchao Yao, Ya Zhang, Ivor Tsang, and Jun Sun. 2017. Discovering user interests from social images. In *International Conference on Multimedia Modeling*. Springer, 160–172.

[46] Fajie Yuan, Xiangnan He, Alexandros Karatzoglou, and Liguang Zhang. 2020. Parameter-efficient transfer from sequential behaviors for user modeling and recommendation. In *Proceedings of the 43rd International ACM SIGIR Conference on Research and Development in Information Retrieval*. 1469–1478.

[47] Shengyu Zhang, Dong Yao, Zhou Zhao, Tat-Seng Chua, and Fei Wu. 2021. CauseRec: Counterfactual User Sequence Synthesis for Sequential Recommendation. In *Proceedings of the 44th International ACM SIGIR Conference on Research and Development in Information Retrieval*.

[48] Zhi-Dan Zhao and Ming-Sheng Shang. 2010. User-based collaborative-filtering recommendation algorithms on hadoop. In *2010 Third International Conference on Knowledge Discovery and Data Mining*. IEEE, 478–481.

[49] 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 *Proceedings of the 24th ACM SIGKDD International Conference on Knowledge Discovery & Data Mining*. 1059–1068.

# A REPRODUCIBILITY DETAILS

## A.1 Dataset Generation

**Amazon**[6] and **Movielens-1M**[7] are public datasets containing user-item interactions. We collect **Taobao** dataset from Mobile Taobao App[8], containing historical behaviors of randomly sampled users in one month. We show the line plot of sorted number of user behaviors in descending order in Figure 8. One can observe the long-tail effect of data distribution, which indicates the necessity of personalized model.

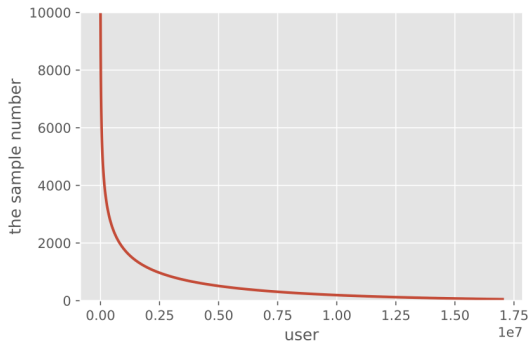


**Figure 8: The Taobao sample number *w.r.t.* the user index.**

We preprocess the datasets to guarantee that users and items have been interacted at least 8 times in Amazon dataset, 20 times in Movielens-1M dataset and 50 times in Taobao dataset.

To train one-round DCCL, each dataset is divided into three disjoint parts according to the log timestamps, including pre-training phase, device-cloud collaborative learning phase and test phase. Pre-training phase is used to learn the initial cloud model $f$ and globally shared parameter basis $\Theta$. Then, *MetaPatch* and *MoMoDistill* are performed successively on device-cloud collaborative learning phase. Finally the model is evaluated on the test phase dataset. For multi-round DCCL in RQ3, the device-cloud collaborative learning phase is further equally divided into multiple parts, each of which corresponding to one round.

## A.2 Parameter Settings

Table 5 shows the hyper-parameter setting of DCCL method for three datasets.

## A.3 Experiment Environment

All experiments are conducted on workstations equipped with GPUs (Tesla V100). All baselines and DCCL method are implemented with TensorFlow. Software versions of Python and TensorFlow are 2.7 and 1.12 respectively.

[6]https://nijianmo.github.io/amazon/index.html
[7]https://grouplens.org/datasets/movielens/1m/
[8]https://market.m.taobao.com/app/fdilab/download-page/main/index.html

**Table 5: Hyper parameters of DCCL in Amazon, Movielens, Taobao dataset**

| Dataset | Parameters | Setting |
|---|---|---|
| Amazon | user embedding dimension | 15 |
| | item embedding dimension | 10 |
| | brand embedding dimension | 10 |
| | category embedding dimension | 10 |
| | learning rate | 1e-3 |
| | batch size | 512 |
| | num_attention_layers | 2 |
| | attention dimension(Q,K,V) | 32 |
| | feed_forward_layer_dimension | 32,16 |
| | patch dimension | 32,16,32 |
| | optimizer | Adam |
| | $\beta$ | 0.01 |
| Movielens-1M | user embedding | 8 |
| | item embedding | 8 |
| | learning rate | 1e-3 |
| | optimizer | Adam |
| | batch size | 256 |
| | encoder layer | 2 |
| | encoder size | 32 |
| | attention dimension(Q,K,V) | 32 |
| | classifer dimension | 32 |
| | patch dimension | 32 ,16, 32 |
| | User Profile Dimension | 8 |
| | $\beta$ | 0.01 |
| Taobao | user embedding | 8 |
| | item embedding | 8 |
| | learning rate | 1e-3 |
| | optimizer | Adam |
| | batch size | 1024 |
| | encoder layer | 2 |
| | encoder size | 32 |
| | attention dimension(Q,K,V) | 32 |
| | classifer dimension | 32 |
| | patch dimension | 32 ,16, 32 |
| | User Profile Dimension | 8 |
| | $\beta$ | 0.01 |