MASCOT: A Quantization Framework for Efficient Matrix Factorization in Recommender Systems

Yunyong Ko*[1], Jae-Seo Yu*[1], Hong-Kyun Bae[1], Yongjun Park[1], Dongwon Lee[2], and Sang-Wook Kim†[1]
[1]*Hanyang University*, Seoul, Republic of Korea
{koyunyong, wotj08090, hongkyun, yongjunpark, wook}@hanyang.ac.kr
[2]*The Pennsylvania State University*, University Park, PA, USA
dongwon@psu.edu

***Abstract*—In recent years, quantization methods have successfully accelerated the training of large deep neural network (DNN) models by reducing the level of precision in computing operations (e.g., forward/backward passes) without sacrificing its accuracy. In this work, therefore, we attempt to apply such a quantization idea to the popular Matrix factorization (MF) methods to deal with the growing scale of models and datasets in recommender systems. However, to our dismay, we observe that the state-of-the-art quantization methods are not effective in the training of MF models, unlike their successes in the training of DNN models. To this phenomenon, we posit that two distinctive features in training MF models could explain the difference: (i) the training of MF models is much more memory-intensive than that of DNN models, and (ii) the quantization errors across users and items in recommendation are not uniform. From these observations, we develop a quantization framework for MF models, named MASCOT, employing novel strategies (i.e., *m*-quantization and *g*-switching) to successfully address the aforementioned limitations of quantization in the training of MF models. The comprehensive evaluation using four real-world datasets demonstrates that MASCOT improves the training performance of MF models by about 45%, compared to the training without quantization, while maintaining low model errors, and the strategies and implementation optimizations of MASCOT are quite effective in the training of MF models. For the detailed information about MASCOT, we release the code of MASCOT and the datasets at:** https://github.com/Yujaeseo/ICDM-2021_MASCOT.

***Index Terms*—quantization, matrix factorization, precision switching, recommender systems**

I. Introduction

Quantization is a widely adopted technique to accelerate the model training in a machine learning research field. To efficiently perform the computing operations (e.g., forward/backward passes) in the model training, quantization converts the parameter values represented by a 32-bit precision (i.e., FP32 or single precision) into a lower precision (e.g., FP16 or half precision). As training progresses, however, the computing results (e.g., gradients at the backward pass) may become too small to be represented under the quantized precision, which may lead to the increase of model errors. To address this problem, a *precision switching* method has been widely applied [1], [2]. The training with precision switching is processed as follows: it starts with low precision (thus, all computing operations are performed with low precision); as the training progresses, the error caused by low precision (i.e., quantization error) is measured; if the error exceeds the pre-defined threshold, precision switching is applied where low precision for computing operations is switched back to high precision (e.g., FP32) to prevent the loss of model accuracy. Many existing quantization methods have attempted to improve the training performance of deep neural network (DNN) models that require massive computational costs in computer vision. For example, [1], [2] improve the training performance of popular convolutional neural network (CNN) models (e.g., AlexNet, ResNet, and GoogleLeNet) by about 30% to 64%, compared to the training with FP32.

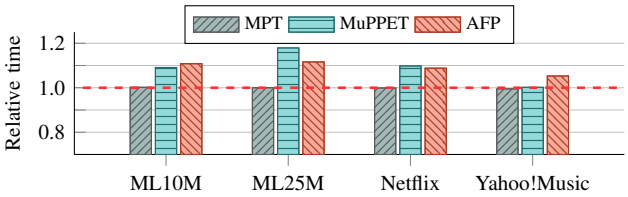


Fig. 1: Poor effects of three state-of-the-art quantization methods when applied to train a MF model (The relative time of 1 in the *Y*-axis indicates the baseline performance of FP32).

On the other hand, *collaborative filtering* (CF) [3]–[5] is one of the most widely adopted techniques in *recommender systems* that accurately recommends favorable items to users by analyzing their feedback (e.g., star rating or browsing history) on items [6]–[8]. In particular, matrix factorization (MF) [9] is a class of popular CF algorithms. Recently, a growing scale of users/items and complicated model architectures make the size of the model significantly increasing, which greatly slows down the training of a recommendation model. As such, it becomes a very important issue to be able to efficiently train the recommendation models, especially MF models, to deal with the growing scale of models and datasets in recommender systems [10], [11].

Recently, DNN models have achieved great successes in many applications including recommender systems [12]–[14]. However, MF models have also been widely used as a way to implement CF. In particular, MF models have various advantages over DNN models. First, MF models provide competitive recommendation accuracy, comparable to or even better than that of more-complex DNN models. MF models predict a

*The first two authors have equally contributed to this work.
† Corresponding author.

user's preference to an item based on the similarity between the user and the item, where the similarity is computed by the dot-product of two latent vectors of the user and item. Recently, [15] showed that this dot-product based similarity could be a better choice than that of DNN models in recommender systems. [15]–[17] empirically reported that well-designed MF models have achieved accuracies higher than those of more-complex DNN models. Second, both training and inference of MF models require less time than those of DNN models since the amount of model parameters and computational overhead in MF models are often smaller than that in DNN models. As such, MF is an excellent recommendation model to attempt to improve its training scalability by means of quantization.

Then, a natural question to raise is "*Can the proven quantization idea for DNN models be adopted to scale up MF models in recommender systems?*" To answer this question, we evaluated the effects of three state-of-the-art quantization methods (i.e., MPT [18], MuPPET [1], and AFP [2]), proposed to improve the training of DNN models, on the training of a MF model using four real-world datasets. Figure 1 shows the results, where the *y*-axis represents the relative training time of existing quantization methods, compared to the training with single precision (i.e., FP32). The results show that *all existing quantization methods are rarely effective or even degrade the training performance*, which is completely different from their reported effects on the training of DNN models. This unexpected result implies that the training of MF models has distinct features unique from that of DNN models.

From this motivation, we conducted an in-depth analysis on the training of both MF and DNN models, and observed two unique features in the training of MF models: (Observation 1) *the training of a MF model is much more memory-intensive than that of a DNN model*, and (Observation 2) in the training of MF models with recommendation datasets, *the quantization error of each user/item is different from each other*, depending on the number of ratings that each user/item has. Based on these observations, we propose a quantization framework for efficient training of MF models in recommender systems, named **M**emory qu**A**ntization and group ba**S**ed pre**C**isi**O**n swi**T**ching (**MASCOT**). MASCOT employs two novel strategies to address the unique features of the training of MF models: (i) a quantization strategy (**m-quantization**) to improve the memory access operations and (ii) a group-based precision switching strategy (**g-switching**) to reflect the difference among the quantization errors across users/items.

To further improve the training of MF models, we adopt a heuristic method to efficiently and accurately estimate the quantization error for each group, and carefully implement MASCOT to efficiently handle the additional overhead during the training by fully exploiting the computing resource of the GPU such as caches and registers.

To the best of our knowledge, this is the first work to scale up the training of MF models using "quantization" with precision switching. The main contributions are as follows:

- Discovering that existing state-of-the-art quantization techniques are rarely effective in the training of MF models through the experiments with four real-world datasets.
- Observing two unique features of the training of MF models: (i) the training of MF models is much more memory-intensive than that of DNN models and (ii) the quantization error of each user/item differs, depending on the number of ratings.
- Proposing a quantization framework for efficient training of MF models in recommender systems, named MASCOT, by employing two novel strategies to address the unique features of the training MF models.
- Comprehensive evaluation verifying the effectiveness of MASCOT in the training MF models, improving the training performance by about 45% on average (almost ideal), compared to the training without quantization.

II. Related work

Recently, various quantization methods have been studied and successfully accelerated the training of large DNN models by reducing (i.e., quantizing) the level of precision in computing operations [1], [2], [18]–[26]. The process of the model training with a quantization method is as follows: (i) reading the model parameters stored with high precision and reducing their precision (e.g., INT8, FP16); (ii) performing the computing operations (forward/backward passes) with low precision; (iii) increasing the precision of gradients computed at the backward pass (e.g., FP32); and, (iv) updating the model parameters by applying the gradients. As such, the performance improvement by quantization becomes larger as the number of computing operations performed with low precision increases, as in (ii). Let us explain two types of quantization approaches: fixed quantization and dynamic quantizaion.

Fixed quantization. In the *fixed quantization methods* [18], [23], [24], [26]–[28], a specific low precision is *fixed* for computing operations (e.g., forward/backward passes) and used during the entire training. [27], [28] use low precision only for the operations at the forward pass (i.e., not at backward pass). Thus, the computing operations at the backward pass are performed in high precision, thereby computing gradients more accurately. [18] and [26] employ FP16 and FP8 for forward and backward passes, respectively. Furthermore, [23] and [24] consider that the precisions required to maintain model accuracy are different according to the types of parameters (i.e., activation, activation gradient, weight gradient). [23] uses FP16 and FP8 for the operations with respect to weight gradient and activation gradient, respectively. Via the different precision for each type of parameters, [23] achieves more performance improvement than [18] that only uses FP16, and maintains model errors lower than [26] that only uses FP8.

Dynamic quantization. Fixed quantization, however, has a limitation in which the error caused by performing the computing operations in low precision (i.e., quantization error) increases as training progresses, which may lead to a significant loss in the model accuracy. In order to address this, *dynamic quantization methods* have been proposed in [1],

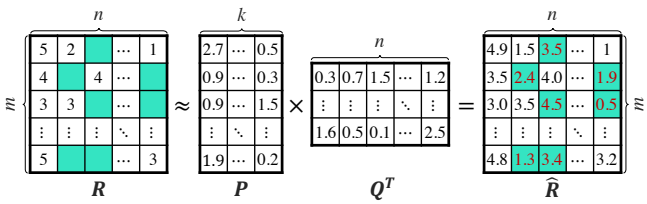


Fig. 2: The architecture of a matrix factorization model.

[2]. These methods measure the quantization error as the training progresses. Then, if the quantization error exceeds the pre-defined threshold, they dynamically switch the low precision for computing operations to higher precision (e.g., FP32) in the middle of the training. Specifically, [1] uses the *gradient diversity* [29] as a metric for the quantization error. Gradient diversity [29] quantifies the degree to which individual gradients are different from each other (i.e., dissimilarity among gradients), which can effectively detect small gradients considered as zeros with low precision. On the other hand, [2] uses the mean difference of gradients before and after applying the quantization as the metric for the quantization error.

III. The Proposed Framework: MASCOT

In this section, we identify the unique features in the training of MF models in recommender systems. To address the unique features, we propose a novel quantization framework for efficient matrix factorization, named MASCOT.

A. Matrix Factorization

We review a MF model [9] that we focus on. As illustrated in Figure 2, a MF model maps users and items to latent space of dimensionality k, where each user u is associated with a vector $p_u \in \mathbb{R}^k$ and each item i is associated with a vector $q_i \in \mathbb{R}^k$. The resulting dot product, $p_u \cdot q_i^\top$, represents user u's overall interest in item i's features, approximating user u's rating of item i (i.e., predicted rating, $\hat{r}_{u,i} = p_u \cdot q_i^\top$). The goal of the MF is to obtain the latent feature matrices P and Q, satisfying $R \approx P \times Q^\top$, given the rating matrix R and the dimensionality of the latent space k. To this end, the objective function $\mathcal{L}$ of a MF model is defined as represented in Eq. 1.

$$\mathcal{L}(P,Q) = \sum_{(u,i)\in R} (r_{u,i} - p_u q_i^\top)^2 + \lambda_P \|p_u\|_F^2 + \lambda_Q \|q_i\|_F^2 \quad (1)$$

where $r_{u,i}$ is the observed rating, λ_P and λ_Q are regularization parameters for users and items, respectively, and $\|\cdot\|_F$ denotes the Frobenius norm.

Stochastic gradient descent (SGD) optimization is widely used to train MF models [30], [31]. The training of a MF model with SGD is processed as follows. For each rating $r_{u,i}$, the system estimates $\hat{r}_{u,i}$ through a dot product of the latent vectors p_u and $q_i^\top$, and compute the prediction error (i.e., $e_{u,i} = r_{u,i} - \hat{r}_{u,i}$). Then, it computes the gradients with respect to p_u and q_i. Finally, it updates p_u and q_i in the opposite direction of the gradients as represented in Eq 2 and Eq 3.

$$p_u \leftarrow p_u + \eta \cdot (e_{u,i} \cdot q_i - \lambda_P \cdot p_u) \quad (2)$$

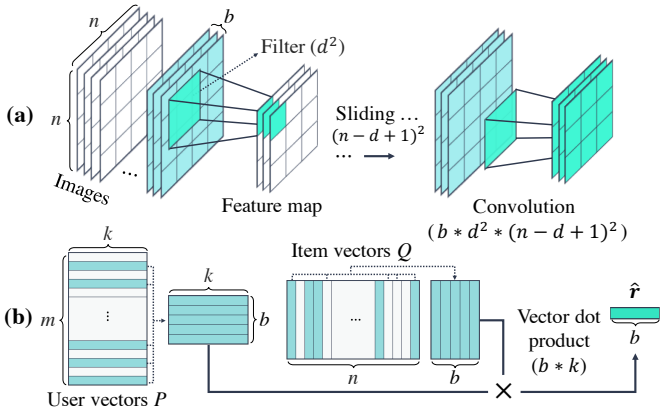


Fig. 3: (Observation 1): the training of MF models (a) is much more memory-intensive than that of DNN models (b).

$$q_i \leftarrow q_i + \eta \cdot (e_{u,i} \cdot p_u - \lambda_Q \cdot q_i) \quad (3)$$

where η is the learning rate. These steps are repeated through all ratings in the training set. We note that for each rating $r_{u,i}$, *only the corresponding latent vectors, p_u and q_i, are updated*, not all latent vectors in the model. After the training is over, given a target user u, the model predicts ratings for the items that have not been rated by the user, using the two trained latent matrices P and Q. Based on the predicted ratings, the most favorable items are recommended to the target user u.

B. Quantization for Memory Access: m-quantization

As described in Section II, the performance improvement by the quantization method can increase as the more number of computing operations are performed in low precision. As shown in Figure 1, however, existing state-of-the-art quantization methods [1], [2], [18] are rarely effective in the training of MF models or even degrade the training performance (See Figure 1). Now, let us figure out the distinct features of the MF model training, causing the unexpected result.

To identify the cause of the unexpected result in Figure 1, we compare the training of MF and DNN models. We, first, analyze the training process of a CNN model, which is one of the most popular DNN models and the target of existing quantization methods. Figure 3(a) shows the process of a convolution layer in the CNN model. where we assume that the batch size is b, the image size is $n \times n$, the filter size is $d \times d$, and the stride is 1. First, b training images and the parameters of the filter are read, and then the filter is applied to each image with stride=1. At this time, the required memory access cost is $b * n^2 + d^2$ for reading the training images and the parameters of the filter, while the computational cost is $b * d^2 * (n-d+1)^2$. Thus, the ratio of the computational cost to memory cost is $\frac{b*d^2*(n-d+1)^2}{b*n^2+d^2}$. For example, when the image size is 32×32 and the filter size is 3×3, the ratio is 7.84, and when the image size is 224×224 and the filter size is 5×5, the ratio is 24.10. Therefore, as the size of data (image) and parameters (filter) increases, the DNN model training is getting more *computation-intensive*, which indicates that the room for the performance improvement by quantization is sufficient.

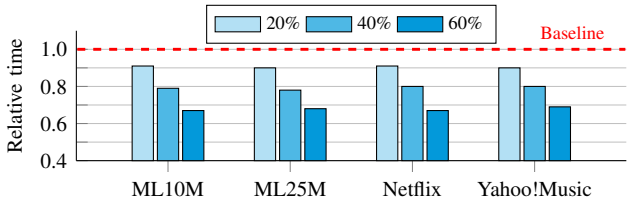


Fig. 4: The potential of the *m*-quantization strategy for improving the training performance of the MF model.

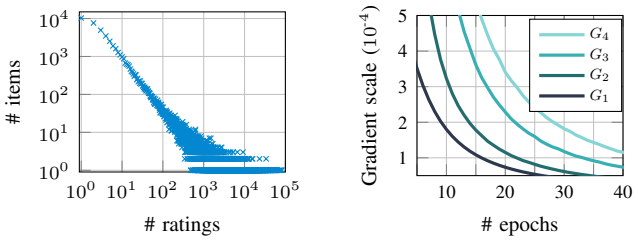
Next, we analyze the process of MF model training. Figure 3(b) shows the dot products of latent vectors of users and items to compute the predicted ratings, where the batch size is b and the dimensionality of the latent vector is k. First, given b ratings, it reads b user and item vectors corresponding to the ratings from P and Q, respectively, and then compute the predicted ratings by taking dot products of the latent vectors. Here, the required memory access cost is $b * 2k$, while the computational cost is only $b * k$. The ratio of the computational cost to memory cost is $\frac{b*k}{b*2k} = 0.5$, which means that in the training of MF models, the memory cost is always larger than the computational cost regardless of the batch size and the dimensionality of the latent space. Therefore, MF model training is *memory-intensive*, and there is little room for performance improvement by quantization.

Through this analysis of the training processes of the DNN and MF models, therefore, we observe that *the training of MF models is much more memory-intensive than that of the DNN model* (Observation 1). From this observation, we posit that it is likely to be the reason why the existing state-of-the-art quantization methods are *not* effective in the training of MF models. As a solution, then, we propose a novel quantization strategy for memory access (***m*-quantization**). The *m*-quantization *stores and manages the parameters of MF models in low precision* (i.e., FP16). Thus, *m*-quantization improves the performance of the training of MF models by reducing the costs for the memory-accessing operations. In the case of Figure 3(b), given b ratings, the memory cost of the MF model training can be reduced from $2b * k$ to $b * k$.

To verify the potential of *m*-quantization, we conduct a preliminary experiment. We apply *m*-quantization to the randomly selected 20%, 40%, and 60% of latent vectors of users and items, train the MF model (k=128) on four real-world datasets using the three versions, and measure their training time. Figure 4 shows the results where the *y*-axis represents the relative time of the training, compared to the training with single precision (i.e., no quantization). As clearly demonstrated in Figure 4, the higher percentage of latent vectors *m*-quantization is applied to, the shorter the training time becomes. This result indicates that *m*-quantization can successfully improve the training performance of MF models.

C. Group-Based Precision Switching: g-switching

In general, it has been known that the datasets used in recommender systems have a power-law distribution [32], [33], which means that a majority of users/items have a small number of ratings, while a small number of users/items have a very large number of ratings. (See Figure 5(a)). Meanwhile, for each rating, only the latent vectors corresponding to the rating are updated in the MF model training with SGD as we explained in Section III-A. Therefore, the latent vectors of a few users/items with a large number ratings are updated very frequently, while the latent vectors of the majority of users/items with a small number of ratings are updated infrequently. As a result, in the MF model training, the number of updates for the latent vector of each user/item is not uniform, varying per the number of ratings that each user/item has.



(a) The distribution of # of items (b) The scale of gradient

Fig. 5: (Observation 2): in the MF model training, the scale of the gradient for the latent vector of each user (item) differs since recommender datasets follow the power-law distribution.

From this understanding, we posit that in training a MF model using recommender datasets, *the scale of gradient for the latent vector of each user/item varies, depending on the number of ratings of each user/item* (Observation 2). This is because the scale of the gradient tends to decrease as the training progresses [1]. Thus, the scale of the gradient for the latent vector of a user/item with many ratings decreases quickly since the latent vector is updated frequently. On the other hand, the scale of the gradient for the latent vector of a user/item with a small number of ratings decreases slowly since the latent vector is updated infrequently. Our observation is clearly demonstrated in Figure 5(b) where the *X*-axis represents the training epochs and the *Y*-axis represents the scale of gradients. For instance, G_1 is the group with the top 25% of users with many ratings, while, G_4 is the group with the bottom 25% of users.

By the disparity among the gradient scales of the latent vectors of users/items, the quantization error of each user/item is also likely to be different from each other. Most existing precision switching techniques (e.g., [1], [2]), however, measure the quantization error for the entire model, and determine the point of precision switching based on the measured error, which may cause the following limitations. For users/items with many ratings, the precision switching is likely to be applied too late, resulting in the loss of accuracy. While, for users/items with a few ratings, the precision switching is likely to be applied unnecessarily quickly, reducing the performance improvement by the quantization.

To overcome these limitations, we propose a novel group-based precision switching strategy (***g*-switching**) that groups users/items having a similar number of ratings into the same group, measures the quantization error per each group, and determines the point of precision switching in a group-wise

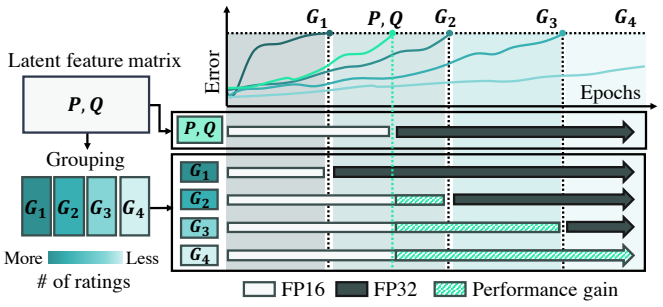


Fig. 6: The performance improvement by the g-switching of MASCOT, compared with the existing precision switching method, in the MF model training.

manner. Figure 6 illustrates how g-switching improves the existing precision switching method. In the training with g-switching, for users/items with many ratings, the precision switching is applied to their group early since the quantization error for the group increases quickly. That is, precision switching is applied *only to the groups that are highly likely to cause the loss of accuracy (e.g., group G_1 in Figure 6)*. On the other hand, for users/items with a small number of ratings, the precision switching is applied to their group late since the quantization error for the group increases slowly. That is, groups unlikely to cause the accuracy loss are trained with quantized precision (i.e., FP16) for a long training period without precision switching, which helps to further improve the overall training performance. Consequently, g-switching can maximize the performance improvement by quantization, while maintaining low model error simultaneously, by applying precision switching only to the groups highly likely to cause the model error. We will empirically verify the effectiveness of each strategy of MASCOT in Section IV-C.

In MASCOT, therefore, users and items are grouped by this intuition: *users/items in the same group have a similar number of ratings*. Note that we manage the groups for users and items, respectively, (i.e., g user groups and g item groups), since the rating distribution of users and items is different. Thus, users/items within the same group have the similar quantization error. As the number of groups gets larger, the quantization error for each group can be estimated more accurately since users/items with different numbers of ratings are divided into groups more finely. However, the overhead required to manage many groups increases as well, causing performance degradation. As an extreme case, if the number of groups is the same as that of the users/items, the precision and quantization error of every user/item has to be managed individually. Thus, we empirically validate the training performance and model error of MASCOT with respect to the number of groups in Section IV-D.

We define the quantization error for the j^{th} user group G_j^U, $q\text{-}error(G_j^U)$, as the inverse form of the gradient diversity [29].

$$q\text{-}error(G_j^U) = \frac{\| \sum_{u \in G_j^U} \nabla p_u \|_2^2}{\sum_{u \in G_j^U} \| \nabla p_u \|_2^2} \quad (4)$$

where ∇p_u is the gradient of user u's latent vector p_u. We note that the quantization error for each item group, $q\text{-}error(G_j^I)$, is computed in the same way as that of the user group. Gradient diversity [29], quantifying the degree to which individual gradients of loss functions are different from each other, is able to effectively detect parameters whose values become too small to be represented by low precision [1]. The quantization error $q\text{-}error(G_j)$ increases as gradients are closer to 0 (i.e., more quantization loss).

Finally, we discuss a heuristic method to efficiently estimate a quantization error. If a quantization error is computed too frequently and all latent vectors of all users/items in each group are used for computing an error (e.g., $\forall u \in G_j^U$), the overhead for the computation can be significant, which may degrade the overall training performance. To address this issue, MASCOT computes a quantization error every π epochs, and uses the sampled $\gamma\%$ latent vectors, where π is the interval of epochs to estimate an error and γ is the sampling ratio for the error estimation. For example, when $\pi = 5$ and $\gamma = 10$, MASCOT estimates a quantization error for each group based on the latent vectors of users/items corresponding to the sampled 10% ratings for every 5 epochs. As π gets larger or γ gets smaller, the overhead for estimating a quantization error decreases but the quality of an error estimation tends to degrade too. Thus, we also empirically evaluate the training performance and model errors of MASCOT with respect to π and γ in Section IV-D.

D. Algorithm and Implementation Details

In this section, we describe the training process of MASCOT and the implementation details. Algorithm 1shows the whole training process of MASCOT. First, MASCOT divides the latent vectors of users and items into g groups, respectively, and then initializes them with half precision (lines 1-2 in Algorithm 1). Next, for each rating $r_{u,i}$, the corresponding latent vectors p_u and q_i are read and updated (lines 5-8). As explained in Section III-C, MASCOT estimates the quantization error for each group based on the sampled ratings. Thus, MASCOT determines whether to select each rating for the error estimation with the probability $\gamma\%$, and the gradients of the latent vectors corresponding to the selected rating are stored in the separate groups for samples, S_j^U and S_j^I (lines 9-11). At every π epochs, MASCOT computes the quantization error for each group and determines whether to apply precision switching to the group (lines 14-19).

Let us describe our implementation details to further optimize MASCOT. Given a rating $r_{u,i}$, in order to read the corresponding latent vectors p_u and q_i, it is required to know its group meta information such as the index of the group that user u belongs to and its current precision (i.e., FP16 or FP32) since the precision for each group is different from each other in MASCOT. Every group has the three types of meta information: (i) the index information is used to identify whether a user/item belongs to the group; (ii) the precision information is used to read the latent vector of the user/item

Algorithm 1 Training of MASCOT

Require: $R \in \mathbb{R}^{m \times n}$, $P \in \mathbb{R}^{m \times k}, Q \in \mathbb{R}^{n \times k}$, # of groups g, error estimate period π, sample ratio γ, error threshold θ, learning rate η

```
R, P, Q ← grouping(R, P, Q, g)
Initialize P, Q with half precision
for t = 1, ..., T do
    for each rating r_{u,i} do
        ∇p_u ← e_{u,i} · q_i − λ_P · p_u
        ∇q_i ← e_{u,i} · p_u − λ_Q · q_i
        p_u ← p_u + η · ∇p_u
        q_i ← q_i + η · ∇q_i
        if S ∼ B(γ) then
            S_j^U.push(∇p_u), S_j^I.push(∇q_i)
        end if
        update_latent_matrix(P, Q, p_u, q_i)
    end for
    if t (mod π) == 0 then
        for j = 1, ...g do
            ε^U ← q-error(S_j^U), ε^I ← q-error(S_j^I)
            precision_switching(P, Q, ε^U, ε^I, θ)
        end for
    end if
    ∀j ∈ {1, ..., g}, S_j^U.flush(·), S_j^I.flush(·)
end for
Return P, Q
```

within the group; and (iii) the quantization error information is used to determine whether *g*-switching is applied to the group.

To efficiently handle the group meta information, we reconstruct the rating matrix (*R*) so that users/items belonging to the same group are sequentially positioned. Note that the reconstruction does not affect the training result [30], [34]. Thanks to the group-based matrix reconstruction, the group index information for all users and items can be managed with a small amount of memory space, since each group can identify all users/items belonging the group with only a single value (i.e., the index of the first/last user/item). Thanks to this optimization, we can reduce the memory space for the group meta information from $O(m+n)$ to $O(g)$, where *m* and *n* are the number of users and items, respectively, and *g* is the number of groups. In our settings, *m* and *n* are much larger than *g* (e.g., in the case of Yahoo!Musics dataset, $m = 1,000,990$, $n = 624,961$, and $g = 100$). By taking this advantage, we store the frequently used group meta information in caches and registers of the GPU. Specifically, due to the limited memory space of registers (at most 256KB), we store the group index information of the most frequently used groups in registers (at most 62 groups). Also, for efficiently searching the group to which each user/item belongs, MASCOT searches groups in the descending order of the number of ratings (i.e., the group with a large number of ratings first). MASCOT with all optimizations improves the naive implementation of MASCOT by up to 14% (Section IV-C).

IV. Evaluation

In this section, we evaluate MASCOT by answering the following four research questions:

- RQ1. Does MASCOT improve the training performance of MF models more than existing quantization methods?
- RQ2. Does MASCOT provide the errors of MF models lower than existing quantization methods?
- RQ3. How effective are the strategies and optimizations of MASCOT in improving the MF model training?
- RQ4. How sensitive are the training performance and model error of MASCOT to its hyperparameters?

A. Experiments Settings

Datasets and models. We evaluate MASCOT with four widely used datasets in recommender systems, MovieLens 10M (ML10M), MovieLens 25M (ML25M), Netflix [35], and Yahoo!Music [36]. Table I shows the statistics of the datasets. We apply 5-cross validation for the evaluations on the ML10M and ML25M datasets. For the Netflix and Yahoo!Music, we just use the provided training and test sets [30], [37]. Netflix consists of 99M training samples and 1.4M test samples, and Yahoo!Music consists of 252M training samples and 4M test samples. For the MF model, we set the dimensionality of latent space *k* as 64 and 128 and the regularization parameters λ_P and λ_Q as 0.01 and 0.015 [38].

TABLE I: Detailed statistics of real-world datasets

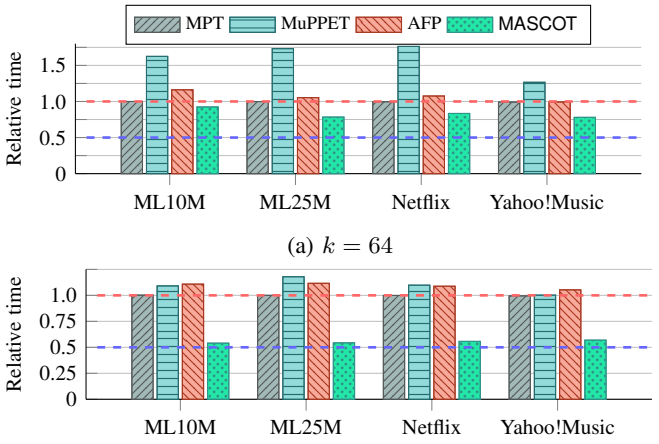
Datasets	# of users	# of items	# of ratings	Sparsity
ML10M	69,878	10,677	10,000,035	98.66%
ML25M	162,541	59,047	24,997,208	99.74%
Netflix	480,189	17,770	100,480,507	98.82%
Yahoo!Musics	1,000,990	624,961	256,804,235	99.96%

System configuration. We use C++ and CUDA to implement all methods including MASCOT on Ubuntu 18.04 OS. We run our experiments on a machine with a NVIDIA RTX 2070 GPU and an Intel i9-9900k CPU with 64 GB memory.

Competing Methods. We compare MASCOT[1] with three existing state-of-the-art quantization methods and two baselines.

- Mixed precision training (MPT) [18]: using multiple (mixed) precisions, low precision (FP16) for computing operations and single precision for parameter update (FP32).
- MuPPET [1]: using fixed-point and single precisions (e.g., INT8, INT12, and FP32) for the computing operations, and applying precision switching based on the quantization error for the entire model.
- Adaptive fixed point (AFP) [2]: using fixed-point precisions (INT8 and INT16) for computing operations, and applying precision switching based on the quantization error for the entire model.
- Two baselines (FP32 and FP16): using single (half) precision for both computing and memory accessing operations.

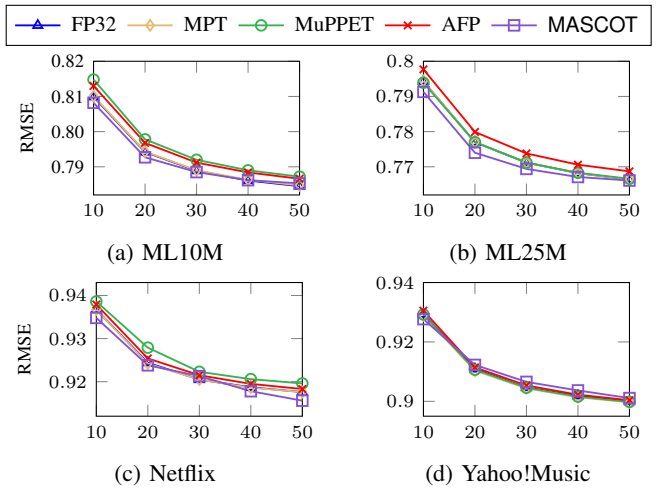
[1]The complete code of MASCOT is available at: https://github.com/Yujaeseo/ICDM-2021_MASCOT



(a) $k = 64$

(b) $k = 128$

Fig. 7: The performance comparison: The relative time of 1 and 0.5 in the Y-axis represent the baseline performance (FP32) and the ideal performance (FP16), respectively.



(a) ML10M

(b) ML25M

(c) Netflix

(d) Yahoo!Music

Fig. 8: The error (RMSE) of the MF model ($k = 128$) trained by each method with respect to training epochs.

Metrics. The goal of this work is to improve the training performance of MF models, not to propose a new recommendation model. Thus, we evaluate MASCOT and competing methods in terms of "errors" in training the MF model with each method, rather than the recommendation accuracy (e.g., precision and recall). We use the root mean square error (RMSE) as the model error metric, which measures the differences between the predicted values (i.e., predicted rating) by a model and the original (observed) rating. We also use the total training time (in seconds) for evaluating the training performance. For a fair evaluation, we run all experiments five times and report the average results.

Hyperparameter Settings. We set batch size b as 1 in all experiments [15], [31], and use Hogwild [39] to fully utilize the computing power of the GPU. We use SGD optimizer and set the learning rate η as 0.01 for all training datasets. We decay the learning rate exponentially with the decay factor 0.1 for 50 epochs [31], [37]. For MuPPET, MPT, and AFP, we set their hyperparameters as recommended in their work [1], [2], [18]. For MASCOT, we empirically found the best values for the error estimate period π, sampling ratio γ, and the number of groups g, and set π as 2, γ as 5%, and g as 100. The experimental results about the effects of the hyperparameters of MASCOT are described in Section IV-D.

B. RQ1 & RQ2: Comparison with Existing Quantizations

In this experiment, we compare MASCOT and three state-of-the-art quantization methods in terms of the training time and the model error. We train the MF model on the four datasets (50 epochs) using all methods, and measure the total training time and the error (RMSE) of the final model. Figure 7 shows that MASCOT improves the training performance of MF models most, compared to the training without quantization. In addition, the performance improvement of MASCOT becomes larger as the dimensionality of the latent space of the MF model increases. In particular, MASCOT provides about 45% performance improvement on average when the dimensionality of latent space k is 128. This result is surprising, considering that the theoretically maximum performance improvement is 50% (FP16). Therefore, these results verify that m-quantization strategy successfully improves the performance of memory access operations in the MF model training, and g-switching fully exploits the performance improvement of the quantization by applying precision switching only to the groups likely to incur larger errors.

On the other hand, all existing quantization methods, originally proposed to improve the DNN model training, cannot improve the training performance of the MF model at all (MPT) or rather degrade the performance (MuPPET and AFP). This is because they use low precision only in the computing operations (i.e., forward and backward) and all operations for memory access are performed with FP32. Especially, MuPPET [1] shows the significant performance degradation (59.64% worse on average) compared to the training with FP32 when the dimensionality of latent space k is 64 (thus, lower computational cost), which means that the additional cost for determining fixed-point precision (such as scaling factor), dominates the performance improvement by the fixed-point precision in the memory-intensive training.

Figure 8 shows the model error (RMSE) of each method with respect to training epochs. MASCOT achieves low model errors, comparable to that of FP32, which indicates that the g-switching of MASCOT applies the precision switching selectively to the groups that are highly likely to incur significant model errors during the training. As as result, the above results demonstrate that MASCOT successfully improves the training performance of the MF training (almost to the limit), while maintaining low errors simultaneously.

C. RQ3. Ablation Study

Strategies of MASCOT: In this experiment, we verify the effectiveness of the m-quantization and g-switching strategies in terms of the training performance and model error. We compare three versions of MASCOT: (i) MASCOT-N1 is with only m-quantization, (ii) MASCOT-N2 is with m-quantization

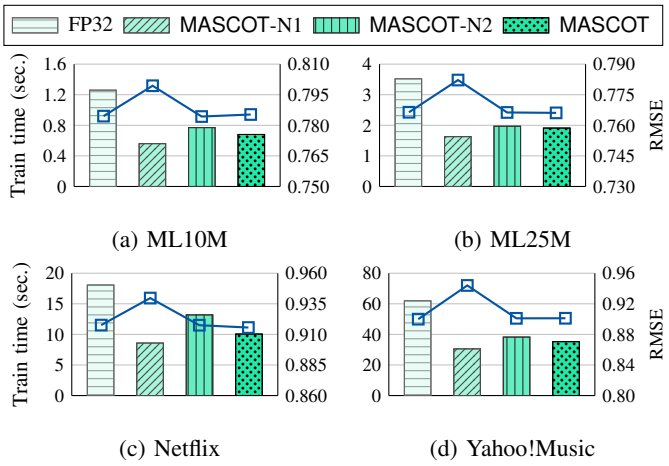


Fig. 9: Effect of the proposed strategies of MASCOT: *m*-quantization and *g*-switching.

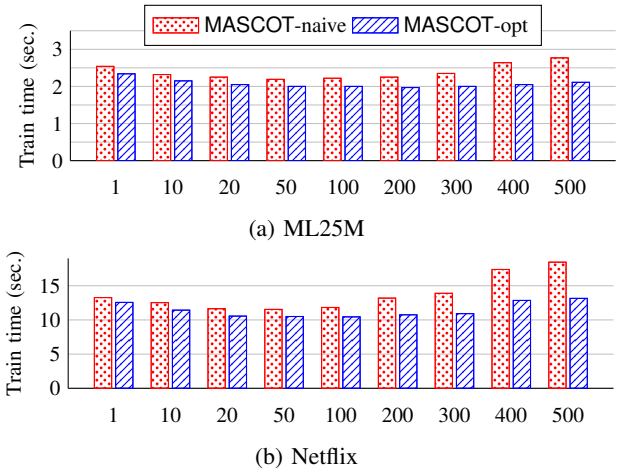


Fig. 10: Effect of the optimization technique of MASCOT with the increasing number of groups.

and the existing precision switching method based on quantization errors for the entire model, and (iii) MASCOT is with both *m*-quantization and *g*-switching strategies. We train the MF model on all datasets (50 epochs) using three versions of MASCOT, and measure the total training time (in seconds) and the error (RMSE) of the final model.

As shown in Figure 9, MASCOT-N1 shows the best improvement in reducing training time (in seconds), compared with FP32. This result demonstrates that *m*-quantization successfully accelerates the training of memory-intensive MF models. However, the training with *m*-quantization alone cannot cope with increasing quantization errors as the training progresses On the other hand, MASCOT-N2 achieves low model errors, comparable to that of FP32, but spends more training time by 34% on average, compared to MASCOT-N1. In MASCOT-N2, the different quantization errors among varying groups cannot be addressed simply by estimating the quantization error based on the entire model. Thus, the precision switching is applied unnecessarily, degrading the performance improvement of quantization as we claimed in Section III-C. Finally, MASCOT provides a performance improvement of about 45% on average, compared to FP32, while achieving comparable model errors. Through this experiment, we verify that both strategies of MASCOT are quite effective in the MF model training.

Optimization technique: Next, we evaluate the optimization technique to efficiently handle the overhead required to read latent vectors for each input rating (line 4 in Algorithm 1). We evaluate this optimization in terms of training time since it does not affect the model error. We compare two versions of MASCOT. (i) MASCOT-naive is a version of MASCOT without our optimization where all group information is managed in the GPU device memory, and (ii) MASCOT-opts is with our optimization technique that manages the frequently used group information (with many ratings) in caches and registers of the GPU, and searches the group index table in descending order of the number of ratings. Specifically, due to the limited memory space of registers (at most 256KB), we store at most 62 groups' index information in the register, and the rest in the caches. We train the MF model using the two versions of MASCOT while varying the number of groups g from 1 to 500, and measure the total training time.

As clearly shown in Figure 10, MASCOT-opt always outperforms MASCOT-naive in the training performance, regardless of the number of groups. The gap tends to grow as the number of groups increases. This result indicates that our optimization effectively addresses the increased searching overhead by the increasing number of groups. Specifically, when the number of groups g is 100, MASCOT-opt outperforms MASCOT-naive by up to 14% on the ML10M (10% on average). We note that we empirically found the best value for g, considering both the training performance and model error. Although the number of groups that we set in this work is 100, a larger number may be best for the training of more complicated models or datasets. Therefore, this result is still promising since the performance improvement of our optimization becomes larger as the number of groups increases. As a result, our optimization technique effectively handles the overhead required to access group meta information for each user/item in MASCOT.

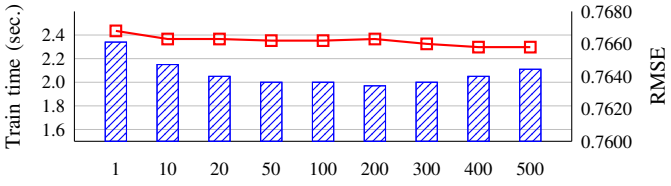
D. RQ4. Hyperparameter Sensitivity

Finally, we evaluate the hyperparameter sensitivity of MASCOT, and provide the best values for each hyperparameter, maximizing the performance improvement while maintaining low model errors.

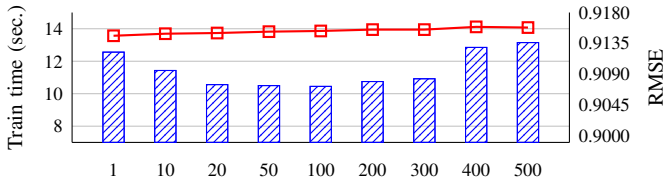
Efficient quantization error estimation In this experiment, we evaluate the effects of hyperparameters π and γ, which determines the overhead of the quantization error estimation. We conduct extensive experiments on MASCOT with varying π and γ (24 combinations in total), where we set the number of groups as 100. Table II shows the results. As the error estimate period π gets larger and the sampling ratio γ gets smaller, the overhead required for estimating the quantization error for each group decreases, which leads to shorter training time. While, it is more difficult to accurately estimate the

TABLE II: The training time (sec.) and model error (RMSE) of MASCOT with respect to the hyperparameters π and γ

		Sampling ratio (γ%)											
		1%		5%		10%		20%		50%		100%	
Dataset	(π)	RMSE	Time(s)	RMSE	Time(s)	RMSE	Time(s)	RMSE	Time(s)	RMSE	Time(s)	RMSE	Time(s)
ML10M	1	0.7861	0.71	0.7847	0.69	0.7846	0.70	0.7845	0.71	0.7846	0.78	0.7844	0.93
	2	0.7858	0.68	**0.7853**	**0.68**	0.7850	0.67	0.7846	0.68	0.7846	0.72	0.7845	0.79
	4	0.7865	0.66	0.7852	0.67	0.7847	0.67	0.7846	0.67	0.7848	0.68	0.7846	0.73
	8	0.7873	0.66	0.7852	0.67	0.7851	0.66	0.7851	0.66	0.7848	0.67	0.7850	0.69
ML25M	1	0.7667	1.91	0.7663	1.19	0.7662	1.98	0.7662	2.15	0.7661	2.62	0.7660	3.47
	2	0.7663	1.89	**0.7661**	**1.91**	0.7662	1.94	0.7661	2.01	0.7663	2.27	0.7663	2.70
	4	0.7669	1.86	0.7667	1.87	0.7666	1.90	0.7662	1.94	0.7665	2.08	0.7663	2.34
	8	0.7672	1.84	0.7665	1.86	0.7666	1.87	0.7667	1.89	0.7664	1.96	0.7663	2.13
Netflix	1	0.9169	10.04	0.9153	10.23	0.9153	10.43	0.9151	11.11	0.9150	12.87	0.9150	15.97
	2	0.9175	9.94	**0.9156**	**10.06**	0.9152	10.23	0.9152	10.54	0.9151	11.56	0.9151	13.18
	4	0.9177	9.83	0.9156	10.00	0.9153	10.06	0.9152	10.32	0.9151	10.84	0.9151	11.69
	8	0.9188	9.86	0.9161	10.04	0.9155	10.03	0.9154	10.13	0.9153	10.42	0.9153	10.95
Yahoo!Music	1	0.9013	34.08	0.9011	35.18	0.9009	37.04	0.9010	40.05	0.9009	48.89	0.9009	65.92
	2	0.9015	33.63	**0.9011**	**35.27**	0.9009	36.00	0.9008	37.67	0.9008	42.31	0.9010	51.01
	4	0.9018	33.24	0.9010	35.04	0.9007	35.15	0.9007	36.22	0.9007	38.80	0.9005	43.89
	8	0.9018	33.25	0.9010	34.13	0.9011	34.45	0.9011	35.04	0.9007	36.62	0.9008	39.68



(a) ML25M



(b) Netflix

Fig. 11: The training time (sec.) and model error (RMSE) of MASCOT with the increasing number of groups.

quantization error, which may degrade the model accuracy. Via these extensive experiments, we found a sweet spot, highlighted in Table II, that significantly improves the training performance and maintains the model error, comparable to that of FP32. Based on these results, we set the sampling ratio γ as 5% and error estimate period π as 2. Also, for the best values for γ and π, we conducted the *t*-tests with a 95% confidence level, and verified that MASCOT with the best values ($\gamma = 5$ and $\pi = 2$) outperforms the naive version of MASCOT ($\gamma = 100$ and $\pi = 1$) with statistical significance (i.e., the *p*-values are below 0.05 in all cases).

The number of groups for users and items As explained in Section III-C, as the number of groups increases, MASCOT is able to estimate the quantization error for each group more accurately by dividing users/items with different ratings more precisely. On the other hand, the additional overhead of managing group information increases as well, which leads to training performance degradation. Thus, in this experiment, we evaluate the effect of the number of groups on the training performance and model error of MASCOT. We train the MF model with varying the number of groups g from 1 to 500, and measure the total training time and the error (RMSE) of the trained model. As shown in Figure 11, the training time tends to decrease as the number of groups increases, but it starts to increase when the number of groups is around 300. As the number of groups increases, the quantization error for each group can be managed more precisely. Thus, the g-switching can be applied only to the groups that may cause the significant model error, avoiding unnecessary precision switching. However, when the number of groups is too large, the overhead required to search the group meta information becomes significant, and degrades the training performance. Meanwhile, the model error of MASCOT is consistent regardless of the number of groups, which means that MASCOT is insensitive to the number of groups. Based on these results, we set the number of groups g as 100.

V. Conclusions

In this paper, we observed that the state-of-the-art quantization methods are not effective in the training of MF models, which is quite different from their success in the training of DNN models. We then identified two distinctive features that cause the difference: (i) the training of MF models is much more memory-intensive than that of DNN models, and (ii) the quantization error for each user/item is quite different from each other, depending on the number of ratings that the user/item has. From these two observations, we proposed a quantization framework for efficient training of MF models, named MASCOT, in recommender systems. The framework employs two novel strategies, *m*-quantization and *g*-switching, to successfully address the distinctive features in the training of MF models. To the best of our knowledge, this is the first quantization framework to accelerate the training of MF models. Through extensive evaluation with four real-world datasets, we

validated that MASCOT significantly improves the training performance of MF models by about 45% on average (almost ideal), compared to the training without quantization, while maintaining low model errors.

ACKNOWLEDGMENT

The work of Sang-Wook Kim was supported by Samsung Research Funding & Incubation Center of Samsung Electronics under Project Number SRFC-IT1901-03. The work of Dongwon Lee was supported by the NSF award #212114824.

REFERENCES

[1] A. Rajagopal, D. Vink, S. Venieris, and C. S. Bouganis. Multi-precision policy enforced training (MuPPET): A precision-switching strategy for quantised fixed-point training of CNNs. In *International Conference on Machine Learning (ICML)*, pages 7943–7952, 2020.

[2] X. Zhang et al. Fixed-point back-propagation training. In *IEEE/CVF Conference on Computer Vision and Pattern Recognition (CVPR)*, pages 2330–2338, 2020.

[3] P. Resnick, N. Iacovou, M. Suchak, and J. R. P. Bergstorm. GroupLens: An open architecture for collaborative filtering of netnews. In *ACM Conference on Computer Supported Cooperative Work (CSCW)*, page 175–186, 1994.

[4] Y. Hu, Y. Koren, and C. Volinsky. Collaborative filtering for implicit feedback datasets. In *IEEE International Conference on Data Mining (ICDM)*, pages 263–272, 2008.

[5] D.-K. Chae, J. Kim, D. Chau, and S.-W. Kim. AR-CF: Augmenting virtual users and items in collaborative filtering for addressing cold-start problems. In *ACM SIGIR International Conference on Research and Development in Information Retrieval (SIGIR)*, pages 1251–1260, 2020.

[6] Y.-C. Lee, S.-W. Kim, and D. Lee. gOCCF: Graph-theoretic one-class collaborative filtering based on uninteresting items. In *AAAI International Conference on Artificial Intelligence (AAAI)*, pages 3448–3456, 2018.

[7] Y. Lee, S.-W. Kim, S. Park, and X. Xie. How to impute missing ratings?: Claims, solution, and its application to collaborative filtering. In *International Conference on World Wide Web (WWW)*, pages 783–792, 2018.

[8] J. Lee, W.-S. Hwang, J. Parc, Y. Lee, S.-W. Kim, and D. Lee. l-injection: Toward effective collaborative filtering using uninteresting items. *IEEE Transactions on Knowledge and Data Engineering*, 31(1):3–16, 2019.

[9] S. Zhang, W. Wang, J. Ford, F. Makedon, and J. Pearlman. Using singular value decomposition approximation for collaborative filtering. In *IEEE Congress on Evolutionary Computation (CEC)*, pages 257–264, 2005.

[10] X. He, H. Zhang, M. Y. Kan, and T. S. Chua. Fast matrix factorization for online recommendation with implicit feedback. In *ACM SIGIR International Conference on Research and Development in Information Retrieval (SIGIR)*, pages 549–558, 2016.

[11] S. Kang, J. Hwang, W. Kweon, and H. Yu. DE-RRD: A knowledge distillation framework for recommender system. In *ACM International Conference on Information and Knowledge Management (CIKM)*, pages 605–614, 2020.

[12] D.-K. Chae, J.-S. Kang, S.-W. Kim, and J.-T. Lee. CFGAN: A generic collaborative filtering framework based on generative adversarial networks. In *ACM International Conference on Information and Knowledge Management (CIKM)*, pages 137–146, 2018.

[13] X. Wang, Y. Chen, J. Yang, L. Wu, Z. Wu, and X. Xie. A reinforcement learning framework for explainable recommendation. In *IEEE International Conference on Data Mining (ICDM)*, pages 587–596, 2018.

[14] K.-J. Cho, Y.-C. Lee, K. Han, J. Choi, and S.-W. Kim. No, that's not my feedback: TV show recommendation using watchable interval. In *IEEE International Conference on Data Engineering (ICDE)*, pages 316–327, 2019.

[15] S. Rendle, W. Krichene, L. Zhang, and J. Anderson. Neural collaborative filtering vs. Matrix factorization revisited. In *ACM Conference on Recommender Systems (RecSys)*, pages 240–248, 2020.

[16] M. F. Dacrema, S. Boglio, P. Cremonesi, and D. Jannach. A troubling analysis of reproducibility and progress in recommender systems research. *ACM Transactions on Information Systems*, 39(2):1–49, 2021.

[17] S. Rendle, L. Zhang, and Y. Koren. On the difficulty of evaluating baselines: A study on recommender systems. In *CoRR abs/1905.01395 (2019). arXiv:1905.01395 http://arxiv.org/abs/1905.01395*, 2019.

[18] P. Micikevicius et al. Mixed precision training. In *International Conference on Learning Representations (ICLR)*, 2018.

[19] S. Gupta, A. Agrawal, K. Gopalakrishnan, and P. Narayanan. Deep learning with limited numerical precision. In *International Conference on Machine Learning (ICML)*, pages 1737–1746, 2015.

[20] M. Courbariaux, Y. Bengio, and J. P. David. Training deep neural networks with low precision multiplications. In *International Conference on Learning Representations (ICLR)*, 2015.

[21] S. Wu, G. Li, F. Chen, and L. Shi. Training and inference with integers in deep neural networks. In *International Conference on Learning Representations (ICLR)*, 2018.

[22] D. Das et al. Mixed precision training of convolutional neural networks using integer operations. In *International Conference on Learning Representations (ICLR)*, 2018.

[23] R. Banner, I. Hubara, E. Hoffer, and D. Soudry. Scalable methods for 8-bit training of neural networks. In *International Conference on Neural Information Processing Systems (NIPS)*, pages 5151–5159, 2018.

[24] C. Sakr and N. Shanbhag. Per-tensor fixed-point quantization of the back-propagation algorithm. In *International Conference on Learning Representations (ICLR)*, 2019.

[25] F. Zhu et al. Towards unified INT8 training for convolutional neural network. In *IEEE/CVF Conference on Computer Vision and Pattern Recognition (CVPR)*, pages 1969–1979, 2020.

[26] N. Wang, J. Choi, D. Brand, C. Y. Chen, and K. Gopalakrishnan. Training deep neural networks with 8-bit floating point numbers. In *International Conference on Neural Information Processing Systems (NIPS)*, pages 7686–7698, 2018.

[27] M. Courbariaux, Y. Bengio, and J. P. David. BinaryConnect: Training deep neural networks with binary weights during propagations. In *International Conference on Neural Information Processing Systems (NIPS)*, pages 3123–3131, 2015.

[28] I. Hubara, M. Courbariaux, D. Soudry, R. El-Yaniv, and Y. Bengio. Binarized neural networks. In *International Conference on Neural Information Processing Systems (NIPS)*, pages 4114–4122, 2016.

[29] D. Yin, A. Pananjady, M. Lam, D. Papailiopoulos, K. Ramchandran, and P. Bartlett. Gradient diversity: A key ingredient for scalable distributed learning. In *International Conference on Artificial Intelligence and Statistics (AISTATS)*, pages 1998–2007, 2018.

[30] J. Oh, W.-S. Han, H. Yu, and X. Jiang. Fast and robust parallel SGD matrix factorization. In *ACM SIGKDD International Conference on Knowledge Discovery and Data mining (KDD)*, pages 865–874, 2015.

[31] X. Xie, W. Tan, L. L. Fong, and Y. Liang. CuMF_SGD: Parallelized stochastic gradient descent for matrix factorization on GPUs. In *International Symposium on High-Performance Parallel and Distributed Computing (HPDC)*, pages 79–92, 2017.

[32] H. Steck. Item popularity and recommendation accuracy. In *ACM Conference on Recommender Systems (RecSys)*, pages 125–132, 2011.

[33] H. Abdollahpouri, M. Mansoury, R. Burke, and B. Mobasher. The connection between popularity bias, calibration, and fairness in recommendation. In *ACM Conference on Recommender Systems (RecSys)*, pages 726–731, 2020.

[34] Y. Zhuang, W. S. Chin, Y. C. Juan, and C. J. Lin. A fast parallel SGD for matrix factorization in shared memory systems. In *ACM Conference on Recommender Systems (RecSys)*, pages 249–256, 2013.

[35] R. M. Bell and Y. Koren. Lessons from the Netflix prize challenge. *ACM SIGKDD Explorations Newsletter*, 9(2):75–79, 2007.

[36] G. Dror, N. Koenigstein, Y. Koren, and M. Weimer. The Yahoo! Music dataset and KDD-Cup'11. In *Proceedings of KDD Cup 2011*, pages 3–18, 2012.

[37] H. Yun, H. F. Yu, C. J. Hsieh, S. V. N. Vishwanathan, and I. Dhillon. NOMAD: Non-locking, stochastic multi-machine algorithm for asynchronous and decentralized matrix completion. *Proceedings of the VLDB Endowment*, 7(11):975–986, 2014.

[38] Z. Gantner, S. Rendle, C. Freudenthaler, and L. Schmidt-Thieme. MyMediaLite: A free recommender system library. In *ACM Conference on Recommender Systems (RecSys)*, pages 305–308, 2011.

[39] B. Recht, C. Re, S. Wright, and F. Niu. Hogwild: A lock-free approach to parallelizing stochastic gradient descent. In *International Conference on Neural Information Processing Systems (NIPS)*, pages 693–701, 2011.