DQRM: Deep Quantized Recommendation Model A Recommendation Framework That is Small, Powerful, and Efficient to Train

Anoynmous Authors

ABSTRACT

Large-scale recommendation models is the dominating workload, nowadays, in the cloud servers of most giant internet companies. Because of the necessity to process various categories of the sparse features, successful recommendation models have to incorporate massive embedding tables, which impose a severe memory bottleneck for recommendation model training and inference. In this work, we propose a novel recommendation framework that is small, powerful and efficient to train, based on the state-of-the-art Deep Learning Recommendation Model (DLRM). The proposed framework makes inference more efficient on the cloud servers, explore the possibility of deploying powerful recommendation on smaller edge devices, and optimize the workload of the distributed communication overhead in training under the Data Parallelism settings. Specifically, we show that Quantization-aware Training (QAT) can impose a strong regularization effect to mitigate the severe overfitting issues suffers from DLRMs. Benefiting from higher efficiency of training data utilization, we achieved uniform INT4 quantization of DLRM models without accuracy drop. We further propose two techniques that improve and speedup the conventional QAT workload specifically for the embedding tables in the recommendation models. Moreover, distributed training time is dominated by communication overhead. To optimize for the gradient communication, we quantize the gradients of the embedding tables into INT8 on top of the well-supported strong specified sparsification. We show that combining gradient sparsification and quantization together drastically reduce the amount of communication, only incuring small reasonable accuracy drop. All the code and experiments are open-sourced in the following link: https://github.com/YangZhou08/Deep_Quantized_Recommendation_Model_DQRM.

CCS CONCEPTS

• **Computing Systems** → *Efficient Large-scale Neural Network Training*; *Distributed Training*; • **Networks**;

KEYWORDS

neural networks, quantization, data parallelism, regularization

ACM Reference Format:

Anoynmous Authors. 2018. DQRM: Deep Quantized Recommendation Model A Recommendation Framework That is Small, Powerful, and Efficient to Train. In *Proceedings of Make sure to enter the correct conference title from your rights confirmation emai (Conference acronym 'XX).* ACM, New York, NY, USA, 11 pages. https://doi.org/XXXXXXX.XXXXXXX

Conference acronym 'XX, June 03–05, 2018, Woodstock, NY

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

1 INTRODUCTION

When the internet service becomes ubiquitous nowadays, personalization becomes the critical area of the services provided by the current internet giants. Every service users' unique taste and preferences need to be catered. With billions of internet users present today, the need for recommendation models is rampant. According to [8], over 79% of the entire Meta's cloud ML inference cycle is spent on the inference of various sizes of recommendation models. By Amdale's law, a slight boost in recommendation model efficiency can incur massive performance boost, highly lucrative to the large internet companies. On the other side, if part of the inference workload can be migrated to edge devices, internet service providers can save precious cloud resources, while the users will have less of their personal data sent to cloud environment, which strengthens their personal data privacy. Both the company and the end users are better off. In this work, we propose a new recommendation model framework that is small, powerful, and efficient to train. The proposed recommendation models are both more efficient in inference on the cloud environment and possible to fit on edge devices for inference. Also, because of the necessity for periodic retraining in the recommendation models, the proposed framework is optimized for model training on the cloud distributed environment.

Designing Deep Learning based Recommendation Models is challenging, because of the necessity to both process dense and sparse inputs. Successful previous works ([16], [22]) utilize massive embedding tables, each corresponds to a sparse feature category. Embedding tables are proven to learn sparse features well. However, embedding tables are always massive, easily taking up GB, even TB, to store. The massive model size makes memory the prime bottleneck in both training and inference. The severe memory bottleneck motivates a huge effort in shrinking recommendation model size ([6], [20], [13], [15], [7], [3]).

In this work, we base our model framework on the Deep Learning Recommendation Model (DLRM) [16], the state-of-the-art recommendation model for Click-Through-Rate (CTR) prediction. The model architecture is summarized in Figure 1 (a). Specifically, we discover that DLRM models suffer from the severe overfitting issue, with testing accuracy dropping steeply after usually the first one or two epoch. We propose to heavily quantize the DLRM model into ultra-low INT4 and show that Quantization-aware Training (QAT) impose a strong regularization effect in DLRM training. When strong quantization is applied, the convergence of the model initially is slower than the full precision model. However, we show QAT is able to make the testing accuracy continuously raise for more than five epochs. Benefiting from the strong regularization

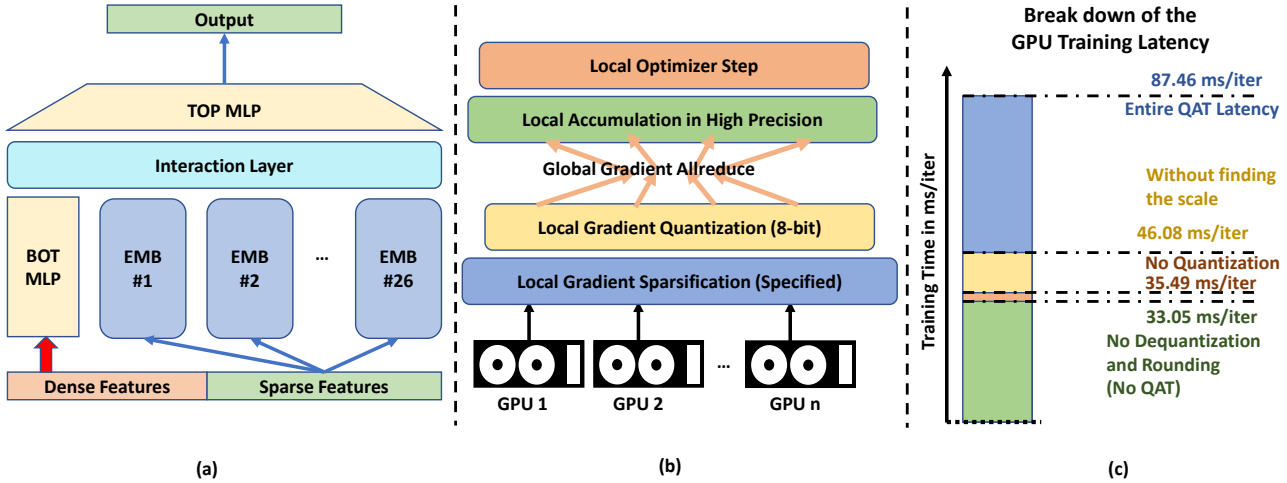

Figure 1: (a) shows the state-of-the-art large-scale recommendation model architecture. The model contains two types of layers: Embedding tables and MLP layers. (b) depicts the entire gradient compression and communication during the training of DQRM quantization framework. (c) shows a breakdown of the training time on four Nvidia M40 GPUs running QAT of DLRM with only embedding table quantization; the majority of the training time is spent on finding the quantization scale, and even more so on the CPU clusters. (Figure best viewed in color)

effect for overfitting and the higher efficiency to utilizing training dataset. We achieve comparable testing accuracy with the unquantized model, or even higher in some dataset, with INT4 quantization of the entire model. Besides, the conventional QAT is highly inefficient in recommendation model quantization. We propose two techniques that reduce giant additional memory copies during QAT and drastically speedup the training time lagged by the traversal of large tensors in memory.

Besides, training DLRM-like model usually occurs in the highly distributed cloud environment. The training time is usually dominated by the heavy communication overhead. We propose to apply communication sparsification and quantization together to shrink the communication workload occuring in every iteration. Specifically, We target the Data Parallelism environment. We show that using well-supported specified sparsification (embedding table gradients nonzero for only vectors that are used in the current iteration) of gradients can compress the communication by three orders of magnitude. On top of that, we further quantize the communication messages from the massive embedding tables into INT8, shinking remaining size of communication into one half during training on the large Terabyte dataset. Also, when both sparsification and quantization are added, we show that the accuracy degredation is reasonably small. We summarize and show the entire backpropagation of the proposed framework in Figure 1 (b). From now, we package the quantization framework as Deep Quantized Recommendation Model (DQRM).

We summarize the contribution of the work as three-fold:

- Propose a novel recommender system network that is small, powerful, and fast to train
- Improve the speed of QAT on recommender system models
- Utilize both sparsification and quantization of gradients on recommender system models

We organize the paper as follows: Section 2 presents the summary of techniques and features of DQRM. Section 3 presents the detailed experiment settings and results evaluating the DQRM framework on the two public CTR datasets: Criteo Kaggle and Criteo Terabyte datasets, while section 4 further provide the ablation studies which offers in-depth experiments that support features' effectiveness in shrinking the large-scale recommendation model DLRM. Section 5 summarize the previous related works on model quantization, recommendation model size compression, and efficient recommendation model training, and we conclude the work in Section 6.

2 METHODOLOGY

2.1 Quantization-aware Training

In this work, we worked on uniform symmetric quantization. When quantizing a floating point number x, we are essentially finding a mapping between x and an integer number I between $[-2^{b-1}, -2^{b-1}-1]$. Formally, quantization is defined as

$$I = Q(x,k) = \frac{clip(x,-\alpha,\alpha)}{S}$$

, where Q is the quantization function and S is the quantization scale. In this work, we use uniform quantization where S is determined using the following formula:

$$S = \frac{\alpha}{2^{b-1}-1}$$

. In both of the above equations, term α is the clipping range of quantization, usually determined by the minimum and the maximum of the distribution of the model weights, X.

In this work particularly, we directly use $max(abs(max(x)), abs(min(x)))$ to compute the clipping range α. During experiment, we found that the minimum and the maximum of the embedding weights are constantly enlarging, rendering the range of clipping

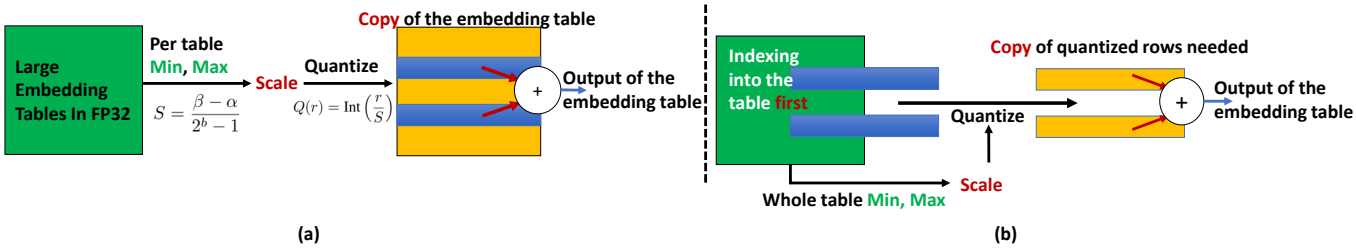

Figure 2: (a) Conventional way in QAT, where the entire weight of the layer is copied and quantized. But, in this way, the majority of the embedding table copied and quantized were not used, and it worsens the already memory bottleneck in QAT. (b) Our way to avoid massive copy by first perform the functional part of each layer and then perform copying and quantization. In this way, only part of the weights that are actually used are copied and quantized, utilizing the memory more efficiently. (Figure best viewed in color)

widen continuously. More detailed study can be found in Appendix C.2. Percentile clipping adds additional overhead in embedding table traversal, which, as we show later, is a big challenge for quantizing embedding tables. On the other hand, trained clipping range may be sub-optimal for the moving target.

We are aware of previous works [7] that uses non-uniform codebook quantization for quantizing embedding tables. Although codebook quantization theoretically can find better fit for an arbitrary weight distribution, but without complex implementation support, it slows down the process of quantization, which is less significant in Post-Training Quantization (PTQ) which their work based on and more significant in our work which uses Quantization-aware Training (QAT). Quantization used in this work is symmetric, meaning that the minimum of the clipping range is the reverse of the maximum. During testing or the inference of the quantized model, the scale is unchanged static.

We deploy Quantization-aware Training (QAT) extensively in our framework. A striking difference between QAT and Post-Training Quantization (PTQ) is the retraining process (finetuning) that comes after the initial usual pretraining using the full precision and later quantization into low-precision. During every iteration of the finetuning process of the quantized model, a copy of full precision weights is always stored, which is updated by the gradients in the back-propagation. After updating the full precision weights from the previous iteration, the model is quantized again based on the full precision model from the previous iteration. However, when training large-scale DLRM models in particular, having both the quantized and the full precision model copies introduce additional inefficiency in memory utilization, which section 2.2 sets out to solve.

2.2 Reducing additional memory of unused weights

Different from other popular neural networks - CNNs or transformers, embedding tables occupy over 99% of the entire model size. From these giant embedding tables, previous works [15**?**] have shown that training large-scale recommendation models is memory-bound, instead of compute-bound as in CNNs and transformers. Additionally, we find that when running the conventional QAT workflow on DLRM naively, the already severe memory bottleneck is worsened.

We present a schematic in Figure 2 (a). Conventionally, all the weight tensors inside convolutional layers or linear layers have to contribute to the computation further on in the model. Therefore, the entire weight tensor in the quantized layer needs to be quantized. As mentioned before, preserving a copy of full precision weight is required in QAT. Therefore, we now have a full precision and a low-precision quantized copy of the same tensor. If such method is blindly applied to embedding table quantization. The entire giant embedding table needs to be copied in memory, which is completely unnecessary because fundamentally different from convolutional and linear layers, not all weight contribute to the later stage of computation.

Instead, we propose to perform the functional part of the layer first, indexing into the embedding tables and retrieving the embedding vectors that are actually needed in the following layers in the current iteration. Then, only quantize the embedding vectors that are needed. In the backward pass, the gradient can still update the full precision embedding tables. The key is that throughout the entire iteration, only used embedding tables are copied, and depending on the batch size and the power-law within training dataset, it is usually more than three magnitudes less than the number of embedding vectors in the embedding tables on average.

2.3 Periodic Update in Finding Quantization Scale

The giant embedding tables in DLRM-like large-scale recommendation models also pose another big challenge to conventional quantization workflow. In quantization, the first step is usually finding the quantization scale. Since we use uniform quantization and determine the clipping range for quantization based on embedding table weights on that iteration, we unavoidably have to find the maximum and the minimum value of the entire embedding table during the forward pass of QAT. Given the giant embedding table size, this step is troublesome.

Shown in Figure 1 (c), we run a breakdown in training time on the distributed Data Parallelism (DP) training of DLRM QAT on four Nvidia M40 GPUs. To better showcase the problem, here we only quantize the embedding tables. Under the DP settings, without

adding QAT, distributed DLRM training already poses 33.05 ms/iter, as shown in the green area botton in the figure (note-worthily, without communication, the latency falls down to 10* ms/iter). Here, we enable the efficient implementation of QAT for embedding tables presented in Section 2.2 and specified sparsification of gradient. When adding the QAT entirely, the training time blows up to 87.46 ms/iter. We found out that the heavy training time consumption, nearly half is spent on finding the maximum and minimum of the embedding tables. If the operation of computing quantization scale is left out, the training time drops down to 46.08 ms/iter. In addition, quantization, dequantization, and rounding offers relatively less additional overhead. The breakdown shows that finding embedding table quantization scale is an urgent problem before designing an efficient quantization framework for DLRM models.

We also run experiments on Intel(R) Xeon(R) Platinum 8280 CPU clusters based on PyTorch framework. We found that such problem even significantly magnifies on the CPUs. Finding the quantization scale for large DLRM models under the Terabyte dataset settings can take more than 1000 ms per iteration of QAT. The key problem lies in the traversal of extraordinarily large tensors. We show that periodic update of the quantization scale works effectively in solving the huge lag of computing quantization scale. Periodic update both amortize the huge quantization overhead to hundreds of iteration without introducing tangible hurt in model convergence. We showed no accuracy degredation while amortizing the training latency almost completely on both Kaggle and Terabyte dataset. Detailed experiment settings and results are presented in Section 4.1.

2.4 Quantizing Embedding table gradient only

With our improved techniques on QAT, we can train DLRM-like models in low-precision efficiently. Through our experiments with QAT on DLRM, we find out that, however, when training using low-precision data types, DLRM converge much slower than using single-precision floating point numbers. QAT usually requires a few more epochs to during training. Also, even though QAT time shrinks significantly than initial unopimized state, it still have additional compute overhead in quantization, taking slightly longer than normal training. To make QAT more efficient and competitive in training time compared with normal training, we further optimize the communication workload that occurs in every iteration.

Specifically, we look at two directions. Sparsification and quantization of communication message. We based all the experiments on DP, so the communication is for the gradients, and the communication operator heavily used is allreduce. On the sparsification side, we find out that specified sparsification, or only communicate gradients for embedding vectors that are only used in the current iteration, is already highly effective. Shrinking the gradient communication by three orders of magnitudes already on Kaggle dataset workload. Also, benefiting from the PyTorch `sparse_coo` tensor support, it is already built-in and well supported by PyTorch libraries. We further on quantize the gradient communication further. We find out that MLP layers, which takes up less in the model size and gradient communication, have gradient that are more sensitive to gradient quantization. Therefore, currently, we are able to quantize the embedding table gradient to INT8, while introducing relatively insignificant accuracy drop. Experiment settings and results are presented in Section 3.3.

3 EXPERIMENT RESULTS

In this section, we present our results evaluating our DQRM framework on two commonly used click-through rate recommendation datasets and two DLRM model configurations accordingly. The two datasets are Criteo Kaggle Display Advertising Challenge Dataset (shortened below as the Kaggle dataset) and Criteo Terabyte Dataset (shortened as the Terabyte dataset). The original DLRM work [16] proposes two different configurations that are optimal specifically for each dataset, and we followed these two configurations in our implementation of DQRM. We organize the results of the experiments under three sections, x.1, x.2, and x.3. Section x.1 details experiments on embedding table quantization alone evaluated on the Kaggle dataset. Section x.2 evaluates DQRM on the Kaggle dataset and on the Terabyte dataset under the distributed settings. Section x.3 evaluates DQRM's effect on communication compression over multiple GPUs and multiple node CPU server environments.

3.1 Quantization of Embedding Tables

It is a consensus in previous works ([6, 8]) that the embedding tables occupy over 99% of the entire DLRM model size. Therefore, compressing embedding tables is important to the overall DLRM inference model size compression. This section focuses on embedding table quantization, and the experiment results presented in the section are evaluated on the Kaggle Dataset. The specific DLRM model architecture we used is presented in Table 1 under the column labeled Kaggle.

Table 1: DLRM model architecture configurations for Kaggle and Terabyte Datasets

Model Specifications	Kaggle	Terabyte
# Embedding Tables	26	26
Maximum # Row among Tables	10131227	9994101
Embedding Feature Size	16	64
Bottom MLP Arch	13-512-256-64-16	13-512-256-64
Top MLP Arch	512-256-1	512-512-256-1

We used four Nvidia M40 GPUs to run distributed DQRM Quantization-aware Training (QAT) under the Data Parallelism (DP) settings. We deploy uniform fixed-point quantization on the embedding tables. We investigate different bit widths for embedding table quantizations: INT16, INT8, INT4, and INT2. Unlike previous works [3, 7] that utilize per-row basis for quantizing embedding tables, for QAT, we quantize the embedding table in the per-table basis. DLRMs with quantized embedding tables under different settings are all trained for 3 epochs using QAT. Figure 3(a) shows the curves of the testing accuracy during 3 epochs of QAT using different quantization bit widths. The vertical dashed lines signify the boundaries of each epoch. Through the experiments, we found that DLRM in single precision (blue) suffers from severe overfitting after the first epoch,

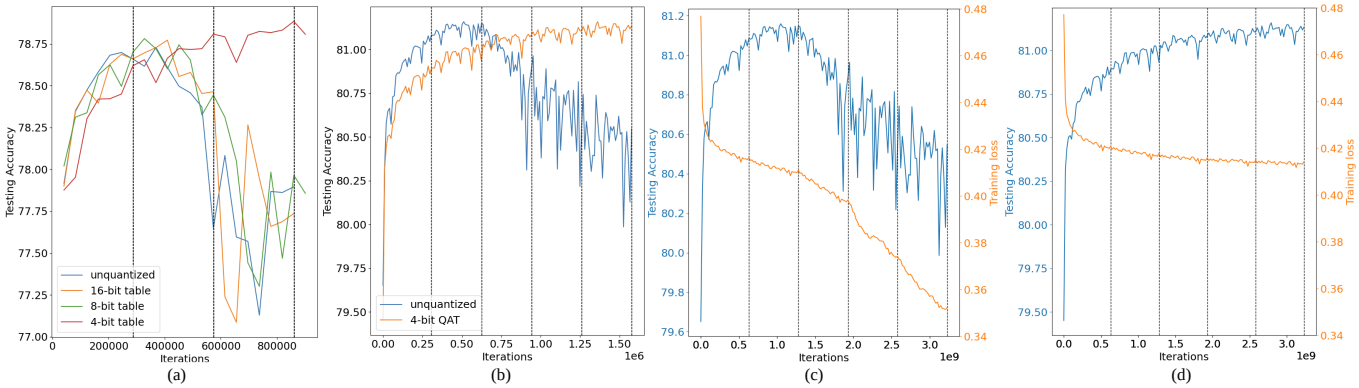

Figure 3: (a) shows the effect of different QAT bit widths has on quantizing embedding tables in DLRM for three epochs of training (epochs are separated by the black dashed lines in all figures). QAT in uniform 4-bit overcomes the severe overfitting suffered by the original DLRM training and lead to significantly higher testing accuracy over three epochs of training. (b) shows the comparison between QAT in 4-bit for DLRM compared to normal training on the Terabyte dataset, and QAT in 4-bit leads to negligible accuracy drop while successfully overcoming the overfitting problem. (c) shows that the training loss (orange curve) for normal training starts decreasing drastically in the third epoch, right where the overfitting occurs. In (d), the training loss curve for QAT in 4-bit decreases stably throughout five epochs of training.

and the testing accuracy crashes afterwards. Similarly, when embedding tables quantized into INT16 and INT8, testing accuracies of INT16 (orange) and INT8 (green) quantization resembles the trend. In contrast, INT4 quantization (red), though converges slower at the beginning, overcomes overfitting and continue to rise over the entire five epochs. Due to limited space, we don't show INT2 testing accuracy curve, but its trend is similar to INT4 quantization but always converge slower and lower.

We compare the performance after embedding table quantization in 2. We extend the QAT training to 5 epochs to let INT4 and INT2 quantization converge further. Although the testing accuracy difference is marginal, slight increase in accuracy is still deemed important for recommendation tasks. Bit widths of INT16 and INT8 leads to comparable performance to the original unquantized DLRM model. Uniform INT4 quantization outperforms the original model in testing accuracy by 0.2% and ROC AUC score by 0.0047. Uniform INT2 quantization leads to significant accuracy drop (> 0.1% in testing accuracy). From the experiment results, we deem uniform INT4 quantization is capable of preserving the original model accuracy while drastically compressing the original model size into 8X reduction.

Table 2: DLRM Embedding Tables Quantization, accuracies evaluated on the Kaggle Dataset

Quantization Bit Width	Testing Accuracy	Testing ROC AUC
Unquantized	78.729%	0.7993
INT16	78.772% (+0.043%)	0.8005 (+0.0012)
INT8	78.781% (+0.052%)	0.8007 (+0.0014)
INT4	**78.936% (+0.20%)**	**0.8040 (+0.0047)**
INT2	78.562% (-0.153&)	0.7953 (-0.0040)

3.2 Quantization of the Whole Model

DQRM is based on DLRM which contains two major components in its network architecture, Embedding Table and Multi-Layer Perceptron (MLP). Comparing with quantizing embedding table, we observe that MLP models much more sensitive to quantization, and doing it naively leads to significant accuracy loss, which is similarly observed in the prior work [3]. We found that using the channel-wise quantization of MLP layers is more preferred, as it leads to lower accuracy drop than the matrix-wise quantization. We evaluated MLP quantization in depth in the ablation studies.

The whole DLRM model is quantized into uniform INT4. We evaluated the performance on both the Kaggle and Terabyte datasets. Following the original DLRM work [16] specific model architectures for the two datasets are different and are summarized in Table 1. The model running on Terabyte dataset is much larger comparatively. We used the same experiment platforms for the Kaggle dataset as in section 3.1. However, running Terabyte model is much more resource-demanding. We deploy the models on the Intel VLAB CPU clusters and specifically use Intel(R) Xeon(R) Platinum 8280 CPUs for model training and inference. The quantized models are all trained using QAT for five epochs. CPUs are much slower in finding the quantization scale of the embedding tables, for running the experiment results, we add a periodic update of period of 200 during training. More detailed explanations can be found in the section 4.1.

Similarly to the experiment results on the Kaggle Dataset, when trained for more than one epoch on the Terabyte dataset, the DLRM model again displays strong overfitting, and the training crashes. We record the training loss and the testing accuracy curves in Figure 3 (b), (c), and (d). In (b), we contrasts the testing accuracies of both original model training and uniform 4-bit QAT. Again, the black vertical dashed lines signify the boundaries of each epoch. At the end of the second epoch, the testing accuracy curve (blue) for the unquantized model drops significantly, while the 4-bit QAT

testing accuracy curve (orange) rises steadily and reaches at the comparable testing accuracies at the end of the fifth epoch. The testing accuracies and the training loss curves are further contrasted for both training scenarios in (c) and (d), unquantized training in (c) and 4-bit QAT in (d). In (c), the training loss curve (orange) for the unquantized model drops drastically at the end of the second epoch, the same time when the model testing accuracy curve (blue) crashes. In contrast, the QAT training loss curve (orange) in (d) decreases steadily throughout the training, while the testing accuracy curve (blue) continue to rise and plateau. From the contrast, the 4-bit QAT is effective in preventing the severe overfitting issue that normal DLRM model training suffers and is able to utilize the training dataset more effectively.

We report training and testing performance in Table 3. We are aware of the previous Post-Training Quantization (PTQ) works ([3]) for DLRM models. However, previous works neither evaluate the quantized model on the public available datasets (Kaggle Terabyte datasets), nor do they publish their implementations. To compare, we showed the testing performance of the vanilla PTQ has on quantizing DLRM models in uniform 4-bit. Following the DQRM 4-bit quantization, the vanilla PTQ 4-bit is achieved to have embedding tables quantized channel-wise and MLP quantized row-wise. (a) shows the experiment results on the Kaggle dataset. DQRM is able to get a substantial boost in testing performance while using significantly smaller bit width INT4 than unquantized model. DQRM achieves both the higher testing accuracy (78.897%) and the higher testing ROC AUC score (0.8035), which is a more insightful score for binary classification tasks. The boost is conjectured to come from the better prevention DQRM has on heavy overfitting. The vanilla PTQ incurs significant accuracy drop in both testing accuracy (> 2%) and testing ROC AUC score (> 0.03). (b) presents the experiment results on the Terabyte dataset. DQRM in INT4 incurs insignificant drop in testing accuracy (0.006%) and testing ROC AUC (0.0006). Similarly, vanilla PTQ in INT4 leads to significant testing accuracy drop (> 2%), which is suboptimal for quantizing the model.

3.3 Gradient Compression

Although DQRM is able to compress the model size significantly (8X reduction) while incurring very minimum testing accuracy drop, it converges slower than unquantized model and takes 5 epochs to converge, 4 more than DLRM training. DLRM that are deployed in the commercial settings are usually massive and usually requires multi- GPUs and nodes for training. In distributed settings for training, communication of activation (used in Model Parallelism) and communication of gradients (used in Data Parallelism) occupies larger portion of the per iteration training latency. In order to speed up the DQRM training, communication message reduction is important. Previous works have introduces Sparsification [9] and Quantization [23] in the distributed training settings for DLRMs. This work sets out to combine both techniques together to speedup DQRM training.

Table 3: DQRM 4-bit quantization results evaluated on Kaggle and Criteo datasets

(a) 4-bit quantization results for DLRM on Kaggle

Quantization Settings	Model Bit Width	Training loss	Testing Accuracy	Testing ROC AUC
baseline	FP32	0.303685	78.718%	0.8001
vanilla PTQ	INT4	-	76.571%	0.7675
DQRM	INT4	0.436685	78.897%	0.8035

(b) 4-bit quantization results for DLRM on Criteo

Quantization Settings	Model Bit Width	Training loss	Testing Accuracy	Testing ROC AUC
baseline	FP32	0.347071	81.165%	0.8004
vanilla PTQ	INT4	-	78.681%	0.7283
DQRM	INT4	0.412979	81.159%	0.7998

We evaluated the DQRM framework under the Data Parallelism (DP) settings, where gradient communication occurs during every iteration of training. We found that the PyTorch DistributedDataParallel library doesn't support gradient quantization, and we customized our own Data Parallel library on PyTorch that implements gradient quantization locally before the allreduce operation. More details of our implementation can be found in the Appendix A.1, and we also open-sourced our implementation in the project's official github repository. Currently, the implementation only supports DP on multiple GPUs.

For experiments on the Kaggle dataset, we utilized four Nvidia A5000 GPUs and present the gradient compression results in Table 7. Naively, because of the giant model size, when the gradient is completely uncompressed, the overhead of gradient communication goes up to 2.1 GB per iteration. Among the overhead, 99% is from the embedding tables. However, since most rows in the embedding tables are unused in every iteration, the gradients for the unused rows are guaranteed to be zero. These zero-valued gradients is omitted during gradient communication, creating the specified sparsity in the gradient tensors. PyTorch has built-in support to exploit specified sparsity in the EmbeddingBag modules, and during gradient communication, we extensively utilize PyTorch built-in `sparse_coo` tensor types to package sparsified gradients for allreduce operations. By Utilizing the specified sparsity, the communication overhead is drastically reduced from 2.161 GB to 1.339 MB, and the training time per iteration is drastically reduced

[1] specified sparsification is a lossless compression for embedding tables so the testing accuracy is exactly the same as uncompressed case

[2] PyTorch sparse tensor allreduce library doesn't support low-precision arithmetics, without further system level effort in low-precision optimization, the latency per iteration increases purely from the quantization overhead per iteration

[3] Our customized Data Parallel library currently doesn't support distributed training on multiple CPU nodes. All experiment results are run in simulation on a single CPU node. Training time is therefore not available

Table 4: Communication compression for Distributed Data Parallelism training among Four nodes or GPUs

Model Settings	Communication Compression settings	Communication Overhead per iter	Latency per iter	Training Loss	Testing Accuracy	Testing ROC AUC
Kaggle	gradient uncompressed	2.161 GB	>1000 ms	0.436685	78.897%	0.8035
	EMB gradient sparfication (specified)[1]	1.339 MB	61 ms	0.436685	78.897%	0.8035
	EMB gradient sparfication + 8-bit quantization	1.218 MB	140 ms[2]	0.442300	78.875%	0.8028
Terabyte[3]	gradient uncompressed	12.575 GB	-	0.412001	81.149%	0.7997
	EMB gradient sparfication (specified)	11.416 MB	-	0.412001	81.149%	0.7997
	EMB gradient sparfication + 8-bit quantization	4.595 MB	-	0.414731	81.035%	0.7961

Table 5: Multi-node Experiment Results with 8-bit gradients, loss evaluated on the Terabyte Dataset

#Node	Training Loss Drop	Testing Acc Drop	Testing AUC Drop
2	-0.002447	0.092	0.0025
4	-0.00273	0.114	0.0036
8	-	-	-

(>15X). Also, using specified sparsity is lossless and has no impact to model training performance.

Besides, embedding table gradients are further quantized into fixed-point INT8 to achieve additional compression. We provide more detailed evaluation for gradient quantization in DQRM in the ablation studies. Unfortunately, PyTorch built-in allreduce operators currently do not support low-precision tensor arithmetic well: INT8 tensors are still broadcasted into FP32 during transmission and calculation. We cannot obtain additional speedup benefiting from the lower theoretical communication overhead. Instead, quantization performed per iteration adds additional computation overhead to the training time per iteration. We argues that with in-depth system-level optimization for allreduce and gradient quantization workload, the training time cost can be further reduced, but such endeavor is outside the scope of the current work. Nonetheless, we showed that adding gradient quantization only introduces insignificant decrease in the testing accuracy and the ROC AUC score.

For experiments on the Terabyte dataset, we currently based our experiments on a simulated platform, using one single Intel(R) Xeon(R) Platinum 8280 CPU node and train the model sequentially using partitioned training dataset to simulate multi-node DP training. More details about the experiment setup can be found in the Appendix A.2. Different from the Kaggle settings, the embedding tables trained in the Terabyte dataset and the training mini batch size are much larger. Similarly, the communication overhead per iteration is drastically reduced by utilizing the specified sparsity. However, of the large mini batch size, more proportion of gradients from the embedding tables left over from specified sparsity compared with the gradients from the MLP layers. Gradient quantization gains more effect in compression ratio. Similarly, gradient compression of the embedding tables introduce insignificant accuracy loss with 0.1% for testing accuracy and less than 0.004 for testing ROC AUC.

Besides, we also evaluate the effect of different node count on gradient quantization. The result is listed in Table 5. Currently, different node counts are simulated on the single CPU node. Across three different node counts, 2, 4, and 8, the drop in training loss, testing accuracy, and ROC AUC score are consistent and small.

Table 6: Evaluation of Periodic Update on Kaggle and Terabyte Datasets

Model Settings	Period	Latency per iter	Testing Accuracy	Testing ROC AUC
Kaggle	1	71 ms	78.936%	0.8040
	200	46 ms	78.938%	0.8046
	500	46 ms[4]	78.897%	0.8032
Terabyte	1	>1200 ms	-	-
	200	57 ms	81.159%	0.7998
	500	34 ms	81.193%	0.8009
	1000	-	-	-

4 ABLATION STUDIES

In this section, we present extended studies on features inside the DQRM framework. Features are categorized into four parts. We study different update period's impact on DQRM in section 4.1, different gradient quantization bit width's effect on model convergence in section 4.2, different sensitivity to quantization among layers in DQRM in section **??**, and the effect of QAT without retraining has on DQRM.

4.1 Periodic update

Large embedding tables from DQRM models present a unique challenge to normal QAT workflow. As showed during the methodology section, embedding tables are so large such that finding the minimum and the maximum element among the table becomes the bottleneck of the forward pass. We propose to utilize periodic update the quantization scale of each embedding table during quantization in order to amortize the huge latency of table look up into multiple iterations. Among the process, we found that the optimal period is different for training settings with different model size and batch size. The experiment results are presented in the Table 6.

[4] rechecking

Among all the experiments presented in the section, gradient communication is not quantized. For experiments running on the Kaggle dataset, we run on distributed data parallelism settings using four Nvidia M40 GPUs. We again trained the model for 5 iterations. Currently, during QAT, only embedding tables are quantized into INT4, while the MLP layers are unquantized and are in FP32. The testing accuracy is expected to be slightly higher than quantizing the whole model as presented previously section 3.2. On GPU workload, the training time per iteration decreases significantly, from over 70 ms to 46 ms, when the quantization scale is updated once every 200 iterations compared with every iteration. Also, notice model trained does not lose testing performance from 200 iterations of delayed updates for the quantization scale. We also test updating the scale once every 500 iterations, and we observe diminishing gain in training time reduction and further slight decrease of testing performance.

Similar experiments are conducted on the CPU clusters. We currently use one CPU node to run DQRM under the Terabyte training settings. We found that with the current ML training framework and no additional optimization, CPUs are much slower in traversing large embedding tables. A single iteration of training of the model under the Terabyte setting takes more than 1.2 seconds to finish, rendering the entire training time to be intimidatingly large. We cannot complete the entire training process. Thus, testing performance results are not available. When the quantization scale is periodically updated once every 200 iterations. The training time per iteration drops drastically to 57 ms per iteration. Due to difference in model size and training mini batch size, we found that using longer period 500 iterations brings further speedup in training time per iteration, dropping to 34 ms, while even slightly benefiting the training accuracy (0.03% compared with 200 iterations as the period). We argue that the reason for the slight accuracy boost is because that QAT inherently has quantization scale growing unstably, periodic update of the quantization scale helps stablize training, which further helps the model convergence.

4.2 Different Gradient Quantization Bit Width

Table 7: Evaluation of Gradient Quantization Bit Width

Embedding Gradient Bit Width	Communication Overhead per iter	Testing Accuracy	ROC AUC
Unquantized	1.34 MB	78.887%	0.8032
INT16	1.26 MB	78.878%	0.8032
INT8	1.22 MB	78.858%	0.8023
INT4	1.19 MB	78.400%	0.7921

In this section, we present our findings related to the quantization of the gradients during distributed DP training for DQRM. All experiments presented in the section assume specified sparsity enabled. Firstly, we study the effect of naive quantization of gradients on the model convergence. The experiments are run on a single Nvidia M40 GPU under a simulated DP environment, using the Kaggle training settings. More details of the experiment platform setup can be found in Appendix A.2. Quantizing all the gradients naively introduces huge accuracy loss and largely hurts the model convergence. We showed the model testing accuracy versus training iterations curves in Figure 4 (c). In the figure, the blue curve signifies the baseline: training curve with single precision gradients. The orange curve is for the INT16 quantization of all gradients, while the green curve is for the INT8 quantization of all gradients. Even INT16 quantization of all gradient communication (orange) every iteration hurts the convergence significantly. It is clear that both the INT16 and the INT8 quantization force the training to plateau earlier and converge to a much lower testing accuracy.

Following our first experiment, we investigate whether we can treat the embedding tables and MLP layers differently. In Figure 4 (d) we investigated quantizing just embedding table gradients or just MLP layers gradient but not both. We show that only quantizing embedding table gradients allows the model to continuously converge throughout the five epochs while quantizing the MLP layers significantly slows down the model convergence.

We then follow the observation and continue quantizing only the embedding tables. We utilize four Nvidia M40 GPUs to train DQRM under the Kaggle settings in the real distributed DP environment. More details about the experiment platform implementation are presented in Appendix A.1. The results are presented in Table 7. We investigated different bit widths for the embedding table bit-width quantization. We found that INT8 gives a reasonable tradeoff between the communication compression ratio and the drop in testing accuracy introduced. INT4 comparatively loses testing accuracy during training more significantly.

4.3 Quantization of different parts of the model

Table 8: Quantization Evaluation of Each Part of the Model

Settings	Testing Accuracy	Testing ROC AUC
Baseline	78.718%	0.8001
+ Embedding Tables in 4-bit	78.936%	0.8040
+ MLP in 4-bit matrix-wise	78.830%	0.8022
+ MLP in 4-bit channelwise	78.897%	0.8035
+ MLP in 8-bit channelwise	78.950%	0.8045

In DQRM training, we observe that different layers display different sensitivity to quantization. To study this, we utilize four Nvidia M40 GPUs and conduct experiments on the Kaggle dataset under the distributed DP settings. The experiment results are presented in Table 4.3. Comparing with quantizing embedding tables, MLP layers quantization cannot benefit from the diminishing overfitting effect. Instead, when quantizing MLP layers in the matrix-wise fashion into INT4 and quantize activation during QAT, the DQRM framework fails to converge. When just quantizing the MLP layers in the matrix-wise fashion and without quantizing activations, DQRM gives more than 0.1% testing accuracy drop, which is a lot for the Kaggle dataset. In contrast, we analyze MLP quantization in the channel-wise fashion, specifically finding a scale for every row inside the linear layer weights. MLP channel-wise quantization out-performs matrix-wise, bringing the overall accuracy loss for entire INT4 model to be under 0.04% and 0.0005 in testing ROC

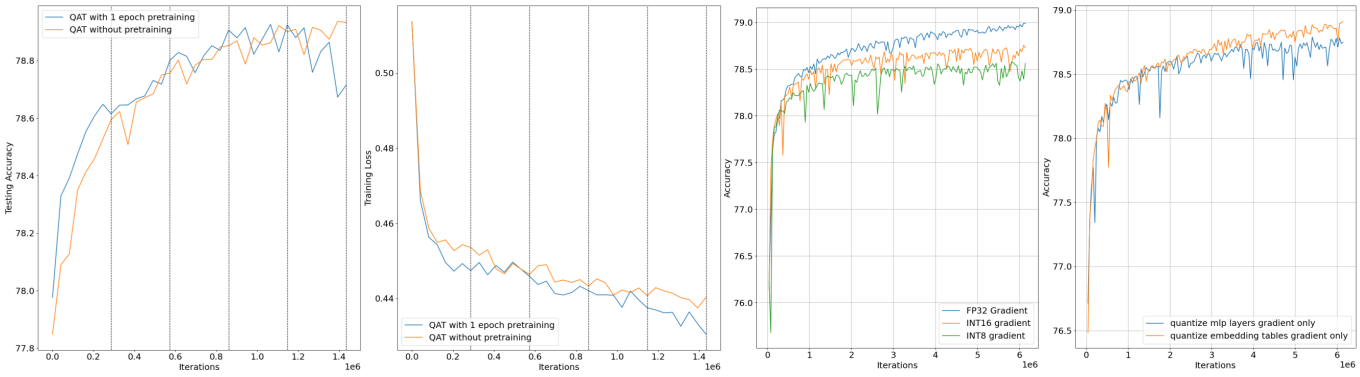

Figure 4: (a) Testing accuracy over the 5 epochs for 1 epoch of pretraining before 4 epochs of QAT and 5 epoch of QAT without pretraining. Pretraining leads to faster overfitting, and QAT without pretraining avoid overfitting and achieves better testing accuracy from 5 epochs of training. (b) Training loss over the 5 epochs. Pretraining before QAT leads to a overall faster decrease of training loss in DLRM compared with QAT without pretraining. (c) Testing Accuracy of naively quantizing gradients for communication into different bit widths. Naive gradient quantization leads to significant accuracy drop. (d) MLP gradients are more sensitive to quantization. If only quantize embedding table gradients, it will leads to less drop in accuracy compared to only quantizing MLP gradients.

AUC compared single precision unquantized MLP model. We also provide the performance of quantizing the MLP layers in INT8 channel-wise, it is on-par with the single precision MLP in test accuracy and test ROC AUC score.

4.4 QAT without Pre-training

Table 9: Evaluation of QAT from scratch on Recommendation Models

Settings	Testing Accuracy	Testing ROC AUC
One epoch pretraining + Four epochs of QAT	78.926%	0.8039
Five epochs of QAT without pretraining	78.936%	0.8040

Previously, QAT has always been used as a finetuning technique on training CNN and transformer models. Usually, with pretraining on the single-precision bit-width, the trained weights can reduce accuracy loss when then finetune on low-precision settings. We examine such paradigm on the DLRM models. We train DLRM in QAT but only quantize the embedding tables into INT4 under the Kaggle dataset using four Nvidia M40 GPUs under the distributed DP settings. We compare between DLRM with one epoch of pre-training in the single precision followed by four epochs of INT4 QAT with DLRM with INT4 QAT from scratch.

We plot the testing accuracy versus iteration curves in Figure 4 (a). The experiment results are presented in Table 4.4. In the diagram, the vertical dashed lines signify the boundary of each epoch. The blue curve is for QAT with pretraining, while the orange curve is without. After the first epoch, we can see that as the model transition from single-precision to INT4 quantized data type, the blue curve drops slightly, which is expected. Further, in the mid third epoch, QAT with pretraining (blue) reaches its peak and then dropping quickly afterwards. QAT from scratch eventually has a slight testing accuracy edge over QAT with pretraining. Also, in Figure 4 (b), we plot the training loss across five epochs for two different settings. QAT with pretraining (blue) has training loss constantly below QAT from scratch. From here, we argue that QAT with pretraining on DLRM models do speedup the model convergence, taking less number of epochs to reach to its peak. However, it suffers from earlier overfitting compared with QAT from scratch. Under our settings, we observe that QAT from scratch slightly gains in testing accuracy of 0.01% with 0.0001 of testing ROC AUC score.

5 PREVIOUS WORKS

Neural Network Quantization - Neural networks quantization has been widely explored in almost all types of neural networks. We point to one survey paper [5] for a comprehensive summary of quantization development and techniques. Besides, we give some highlight to the most relevant previous works to our project. Quantization-aware Training (QAT) has been widely deployed in CNNs ([2], [10], [11], [12], [17], [21]) and transformers ([1], [19], [4], [24], [14]). Compared with Post-Training Quantization (PTQ), QAT, benefitting from the additional round of finetuning using low-precision during forward pass, always achieve lower accuracy loss for various different model achitectures. However, because of the extra quantization step, QAT usually takes much longer time to prepare compared to PTQ. The problem exacerbates in large-scale recommendation models, conventional QAT worsen the already severe memory overhead during training, because of the presence of large embedding tables. This work proposes method to alleviate the issue by optimizing the forward pass of QAT taylored for DLRM-like models.

Compressing Large-scale Recommendation Model - DLRM ([16]) is the state-of-the-art click-through-rate recommendation model designed and deployed by Meta. However, under the MLPerf settings, the model trained on the Terabyte dataset already contains

Embedding tables that requires storage more than 89 GB, which exceeds the memory capacity of every current GPU. The giant model size has brought about huge difficulties in both model training and inference, which motivates works that compress the model. Over 99% of DLRM model size is occupied by the embedding tables, and various previous works attends to shrink the embedding tables. [6] proposes to shrink the embedding dimension of each embedding tables based on their hotness evaluated through a metric based on spectral decay and the popularity of each table. Similarly, [20] proposed to break large embedding tables into smaller sub-tables and combine sub-tables vectors using compositional operations. We argue that both works essentially prune the embedding table sizes and leave the precision of the model unchanged. Our work is complementary to theirs. Also, there are previous works attends to quantize the embedding tables and vectors in recommendation models. [13] uses differentiable product quantization for matrix factorization and neural collaborative filtering models, and [15] shows that half-precision quantized model weights benefit the training speed and does not hurt convergence for matrix factorization models. Besides, [7] uses codebook quantization to quantize the embedding tables of DLRM models into INT4 under the PTQ settings, while [3] further quantize the whole model into 4-bit. Both works revealed that PTQ introduces accuracy degredation which is not as competitive with QAT.

Efficient Training of Recommendation Model - On the Matrix Factorization models side, [15] targets model training and propose to quantize the gradients with dynamic shifting of gradient precision to preserve accuracy. On the other side, for large-scale recommendation model such as DLRM, the models are usually trained using multiple GPUs or CPU nodes. The training time is heavily affected by the communication strategies between nodes. [9] improved upon conventional Top-k sparsification and apply heavy sparsification upon the communication of all-to-all and allreduce under both Model Parallelism (MP) and Data Parallelism (DP) settings. [23] impose quantization on communication message under both MP and DP settings and show that with heavy error compensation, the communication workload can be significantly reduced without significant error degredation. On CNN and ASR models, previous work ([18]) looked at combining sparsification and quantization on communication message together. However, it shows that the merit of combining the two techniques varies hugely between different layer architecture, making their technique not generalizable to all different model architectures. In this work, we looked into these techniques' effects for different part of DQRM under the DP environment and combine specified sparsification and quantization together to further benefit DQRM training.

6 CONCLUSION

In this work, we propose a systematic quantization framework DQRM for large-scale recommendation models to make them small, powerful, and efficient to train. Specifically, We discover that the DLRM model suffer severely from the overfitting problem. We show that heavy quantization can help overcome the strong overfitting and utilize the training dataset more efficiently, achieving higher test accuracy eventually. We observe that conventional QAT is troublesome in training large-scale recommendation models and propose two techniques that hugely alleviate the issue. Besides, to further optimize DQRM under the distributed environment, we combine specified sparsification and quantize together to apply to communication gradient compression. Our framework is evaluated intensively on the published dataset, and we provide extensive exploration on detailed feature decision within the DQRM framework.

REFERENCES

[1] Aishwarya Bhandare, Vamsi Sripathi, Deepthi Karkada, Vivek Menon, Sun Choi, Kushal Datta, and Vikram Saletore. 2019. Efficient 8-Bit Quantization of Transformer Neural Machine Language Translation Model. https://doi.org/10.48550/ARXIV.1906.00532

[2] Matthieu Courbariaux, Yoshua Bengio, and Jean-Pierre David. 2015. Binaryconnect: Training deep neural networks with binary weights during propagations. *Advances in neural information processing systems* 28 (2015).

[3] Zhaoxia Deng, Jongsoo Park, Ping Tak Peter Tang, Haixin Liu, Jie Yang, Hector Yuen, Jianyu Huang, Daya Khudia, Xiaohan Wei, Ellie Wen, et al. 2021. Low-precision hardware architectures meet recommendation model inference at scale. *IEEE Micro* 41, 5 (2021), 93–100.

[4] Angela Fan, Edouard Grave, and Armand Joulin. 2019. Reducing transformer depth on demand with structured dropout. *arXiv preprint arXiv:1909.11556* (2019).

[5] Amir Gholami, Sehoon Kim, Zhen Dong, Zhewei Yao, Michael W. Mahoney, and Kurt Keutzer. 2021. A Survey of Quantization Methods for Efficient Neural Network Inference. https://doi.org/10.48550/ARXIV.2103.13630

[6] Antonio Ginart, Maxim Naumov, Dheevatsa Mudigere, Jiyan Yang, and James Zou. 2019. Mixed Dimension Embeddings with Application to Memory-Efficient Recommendation Systems. *CoRR* abs/1909.11810 (2019). https://arxiv.org/abs/1909.11810

[7] Hui Guan, Andrey Malevich, Jiyan Yang, Jongsoo Park, and Hector Yuen. 2019. Post-training 4-bit quantization on embedding tables. *arXiv preprint arXiv:1911.02079* (2019).

[8] Udit Gupta, Xiaodong Wang, Maxim Naumov, Carole-Jean Wu, Brandon Reagen, David Brooks, Bradford Cottel, Kim M. Hazelwood, Bill Jia, Hsien-Hsin S. Lee, Andrey Malevich, Dheevatsa Mudigere, Mikhail Smelyanskiy, Liang Xiong, and Xuan Zhang. 2019. The Architectural Implications of Facebook's DNN-based Personalized Recommendation. *CoRR* abs/1906.03109 (2019). https://arxiv.org/abs/1906.03109

[9] Vipul Gupta, Dhruv Choudhary, Peter Tang, Xiaohan Wei, Xing Wang, Yuzhen Huang, Arun Kejariwal, Kannan Ramchandran, and Michael W Mahoney. 2021. Training recommender systems at scale: Communication-efficient model and data parallelism. In *Proceedings of the 27th ACM SIGKDD Conference on Knowledge Discovery & Data Mining*. 2928–2936.

[10] Philipp Gysel, Mohammad Motamedi, and Soheil Ghiasi. 2016. Hardware-oriented approximation of convolutional neural networks. *arXiv preprint arXiv:1604.03168* (2016).

[11] Philipp Gysel, Jon Pimentel, Mohammad Motamedi, and Soheil Ghiasi. 2018. Ristretto: A framework for empirical study of resource-efficient inference in convolutional neural networks. *IEEE transactions on neural networks and learning systems* 29, 11 (2018), 5784–5789.

[12] Itay Hubara, Matthieu Courbariaux, Daniel Soudry, Ran El-Yaniv, and Yoshua Bengio. 2016. Binarized neural networks. *Advances in neural information processing systems* 29 (2016).

[13] Wang-Cheng Kang, Derek Zhiyuan Cheng, Ting Chen, Xinyang Yi, Dong Lin, Lichan Hong, and Ed H Chi. 2020. Learning multi-granular quantized embeddings for large-vocab categorical features in recommender systems. In *Companion Proceedings of the Web Conference 2020*. 562–566.

[14] Sehoon Kim, Amir Gholami, Zhewei Yao, Michael W Mahoney, and Kurt Keutzer. 2021. I-bert: Integer-only bert quantization. In *International conference on machine learning*. PMLR, 5506–5518.

[15] Yunyong Ko, Jae-Seo Yu, Hong-Kyun Bae, Yongjun Park, Dongwon Lee, and Sang-Wook Kim. 2021. MASCOT: A Quantization Framework for Efficient Matrix Factorization in Recommender Systems. In *2021 IEEE International Conference on Data Mining (ICDM)*. IEEE, 290–299.

[16] Maxim Naumov, Dheevatsa Mudigere, Hao-Jun Michael Shi, Jianyu Huang, Narayanan Sundaraman, Jongsoo Park, Xiaodong Wang, Udit Gupta, Carole-Jean Wu, Alisson G. Azzolini, Dmytro Dzhulgakov, Andrey Mallevich, Ilia Cherniavskii, Yinghai Lu, Raghuraman Krishnamoorthi, Ansha Yu, Volodymyr Kondratenko, Stephanie Pereira, Xianjie Chen, Wenlin Chen, Vijay Rao, Bill Jia, Liang Xiong, and Misha Smelyanskiy. 2019. Deep Learning Recommendation Model for Personalization and Recommendation Systems. *CoRR* abs/1906.00091 (2019). https://arxiv.org/abs/1906.00091

[17] Renkun Ni, Hong-min Chu, Oscar Castañeda, Ping-yeh Chiang, Christoph Studer, and Tom Goldstein. 2020. WrapNet: Neural Net Inference with Ultra-Low-Resolution Arithmetic. https://doi.org/10.48550/ARXIV.2007.13242

[18] Cedric Renggli, Saleh Ashkboos, Mehdi Aghagolzadeh, Dan Alistarh, and Torsten Hoefler. 2018. SparCML: High-Performance Sparse Communication for Machine Learning. https://doi.org/10.48550/ARXIV.1802.08021

[19] Sheng Shen, Zhen Dong, Jiayu Ye, Linjian Ma, Zhewei Yao, Amir Gholami, Michael W. Mahoney, and Kurt Keutzer. 2019. Q-BERT: Hessian Based Ultra Low Precision Quantization of BERT. (2019). https://doi.org/10.48550/ARXIV.1909.05840

[20] Hao-Jun Michael Shi, Dheevatsa Mudigere, Maxim Naumov, and Jiyan Yang. 2020. Compositional embeddings using complementary partitions for memory-efficient recommendation systems. In *Proceedings of the 26th ACM SIGKDD International Conference on Knowledge Discovery & Data Mining*. 165–175.

[21] Shyam A. Tailor, Javier Fernandez-Marques, and Nicholas D. Lane. 2020. Degree-Quant: Quantization-Aware Training for Graph Neural Networks. https://doi.org/10.48550/ARXIV.2008.05000

[22] Ruoxi Wang, Bin Fu, Gang Fu, and Mingliang Wang. 2017. Deep & cross network for ad click predictions. In *Proceedings of the ADKDD'17*. 1–7.

[23] Jie Amy Yang, Jongsoo Park, Srinivas Sridharan, and Ping Tak Peter Tang. 2020. Training deep learning recommendation model with quantized collective communications. In *Conference on Knowledge Discovery and Data Mining (KDD)*.

[24] Ali Hadi Zadeh, Isak Edo, Omar Mohamed Awad, and Andreas Moshovos. 2020. GOBO: Quantizing Attention-Based NLP Models for Low Latency and Energy Efficient Inference. In *2020 53rd Annual IEEE/ACM International Symposium on Microarchitecture (MICRO)*. IEEE. https://doi.org/10.1109/micro50266.2020.00071

Received 20 February 2007; revised 12 March 2009; accepted 5 June 2009