DQRM: Deep Quantized Recommendation Models

ANONYMOUS AUTHOR(S)

Large-scale recommendation models are currently the dominant workload for many large Internet companies. These recommenders are characterized by massive embedding tables that are sparsely accessed by the index for user and item features. The size of these 1TB+ tables imposes a severe memory bottleneck for the training and inference of recommendation models. In this work, we propose a novel recommendation framework that is small, powerful, and efficient to run and train, based on the state-of-the-art Deep Learning Recommendation Model (DLRM). The proposed framework makes inference more efficient on the cloud servers, explores the possibility of deploying powerful recommenders on smaller edge devices, and optimizes the workload of the communication overhead in distributed 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 suffered by DLRMs. Consequently, we achieved INT4 quantization of DLRM models without any accuracy drop. We further propose two techniques that improve and accelerate the conventional QAT workload specifically for the embedding tables in the recommendation models. Furthermore, to achieve efficient training, we quantize the gradients of the embedding tables into INT8 on top of the well-supported specified sparsification. We show that combining gradient sparsification and quantization together significantly reduces the amount of communication. Briefly, DQRM models with INT4 can achieve 79.07% accuracy on Kaggle with 0.27 GB model size, and 81.21% accuracy on the Terabyte dataset with 1.57 GB, which even outperform FP32 DLRMs that have much larger model sizes (2.16 GB on Kaggle and 12.58 on Terabyte). Important supplementary materials[1] and codes are published anonymously at: https://anonymous.4open.science/r/Deep_Quantized_Recommendation_Model_DQRM-6B4D

Additional Key Words and Phrases: Quantization, Recommendation Systems, Efficient Distributed Training, Data Parallelism

ACM Reference Format:
Anonymous Author(s). 2023. DQRM: Deep Quantized Recommendation Models. 1, 1 (May 2023), 10 pages. https://doi.org/10.1145/nnnnnnn.nnnnnnn

1 INTRODUCTION

With the widespread adoption of Internet services, personalization becomes a critical function of the services provided by the current Internet giants. Every user's unique taste and preferences need to be catered to. With billions of internet users today, the need for recommendation models is more crucial than ever. According to [12], over 79% of the entire Meta's cloud ML inference cycles are spent on the inference of various sizes of recommendation models. By Amdahl's law, a slight boost in recommendation model efficiency can yield a massive performance boost. On the other hand, if part of the inference workload can be migrated to edge devices, Internet service providers can save precious cloud resources, while the users can have less of their personal data sent to the cloud environment, which strengthens their personal data privacy. To achieve less costly recommenders, in this work, we propose a deep quantized recommendation model (DQRM) framework, where the models are both more efficient on the cloud environment, and are possible to fit on edge devices. Moreover, to address the necessity for periodic retraining in recommenders, the proposed framework is optimized for model training in the cloud-distributed environment.

Specifically, designing deep learning-based recommendation models is challenging, because of the necessity to both process dense and sparse inputs. Successful previous works [21, 28] utilize massive embedding tables, each corresponding to a sparse feature category. Although embedding tables are proven to learn sparse features well, they

[1]We followed the rule of RecSys 2023 on supplementary materials. Detailed experimental setup and supplementary ablation studies can be found in `Supplementary_Materials.pdf` from the anonymous repo.

2023. Manuscript submitted to ACM

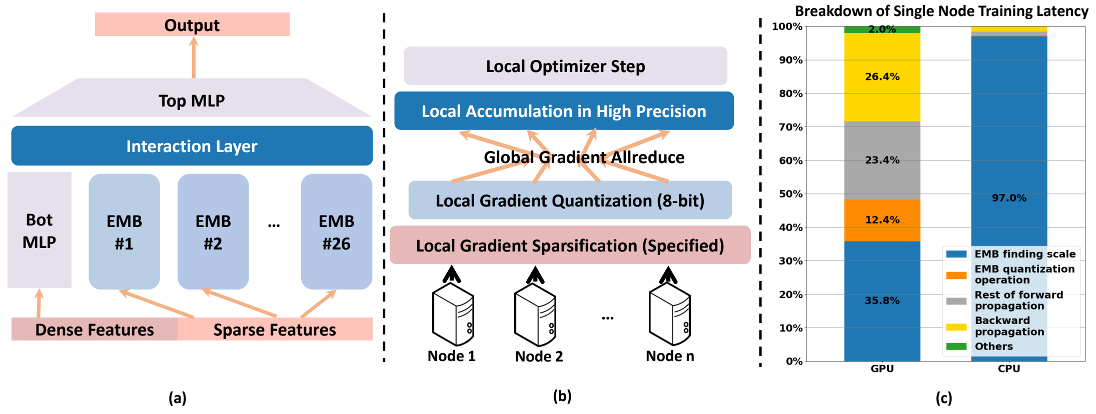

Fig. 1. (a) shows the state-of-the-art large-scale recommendation model architecture. The model contains two types of layers: Embedding tables and MLP layers. (b) Our framework builds on top of specified sparsity and adds quantization to achieve additional gradient compression ratio. (c) shows a breakdown of the single-machine training time running DQRM in INT4; the majority of the training time on the GPU platform (left) is spent on finding the quantization scales, and even more so on the CPU node (right).

are usually massive, and easily take up many GB, and in some cases even TB in DRAM memory. The massive model size makes memory the prime bottleneck in both training and inference. This severe problem motivates a huge effort in shrinking recommendation model size [4, 10, 11, 17, 19, 26]. In our work, we base our framework on the Deep Learning Recommendation Model (DLRM) [21], which is the state-of-the-art for Click-Through-Rate (CTR) ([37]) prediction (as in Figure 1 (a)), and we apply quantization and sparsification to enhance its efficiency.

Based on experiments and analyses, we find that DLRM models suffer from severe overfitting issues, with testing accuracy dropping steeply after the first one or two epochs of training. We propose to heavily quantize the DLRM model into ultra-low INT4 precision, and we show that QAT can greatly reduce the memory and computational costs while imposing a strong regularization effect to benefit DLRM training. As a result, we achieve comparable or even better testing accuracy than the full-precision models. Besides, our experiments indicate that conventional QAT can be highly inefficient in recommendation model quantization. To deal with this issue, we propose two techniques that reduce additional memory copies of large embedding tables during QAT and avoid the costly traversal of large tensors in memory to compute quantization scales. Our proposed methods significantly reduce time costs.

Besides the model size and inference costs, the training of recommendation models is also crucial in real-world applications. Recommenders are usually trained in highly distributed cloud environments, where the training time is dominated by heavy inter and intra-node communications overhead. In this work, we propose to apply communication sparsification and quantization jointly to shrink the workload. We show that using well-supported specified sparsification of gradients can compress the communication by three orders of magnitude. On top of that, we further quantize the communication from the massive embedding tables into INT8, shrinking the remaining size of communication by one-half during training, with a negligible accuracy degradation on the large Criteo Terabyte dataset. We show the backpropagation of the proposed framework in Figure 1 (b). Our contributions are summarized as follows:

- We propose to apply ultra-low precision quantization to alleviate the overfitting of DLRMs meanwhile achieving smaller model sizes,
- We introduce two novel techniques to improve the speed of QAT on recommendation systems,

- We achieve efficient training by jointly applying sparsification and quantization on communications,
- Our DQRM models with INT4 achieve an 8× reduction in model size, while obtaining 79.071% accuracy (0.148% higher than DLRM FP32 baseline) on the Kaggle dataset, and 81.210% accuracy (0.045% higher than DLRM FP32 baseline) on the Terabyte dataset.

2 PREVIOUS WORKS

Compressing Large-scale Recommendation Models - DLRM [21] is a typical and highly popular click-through-rate recommendation model designed and vastly deployed by Meta. Motivated by its importance, many previous works ([10], [26], [32], [5], [29]) focus on compressing DLRM's model size. Since over 99% of the DLRM model size is occupied by the embedding tables, previous efforts focus on shrinking the embedding tables without lowering weight precision. Our work focuses on neural network quantization, which is orthogonal and complementary to these works.

Previously, quantization [9] has been extensively studied on CNNs [2, 3, 7, 14–16, 22, 27] and Transformers [1, 8, 18, 25, 33] and successfully applied to the Matrix Factorization and Neural Collaborative Filtering models [17, 19]. Recently, some works extend quantization to DLRM models but mainly focus on Post Training Quantization (PTQ). [11] uses codebook quantization to quantize the embedding tables of DLRM models into INT4, while [4] further quantizes the whole model into 4-bit. Both works revealed that PTQ introduces accuracy degradation, which motivates other quantization methods like Quantization-aware Training (QAT) to improve. Besides, low-precision training (LPT) of DLRM [20, 30, 34] received attention. [34] and [30] train CTR models using quantized FP16 weights, while [20] trains DCN [28] using INT8 weights during low-precision training and compress models into 2-bit and 4-bit. Different from the LPT works that achieve savings in training memory usage by allowing accuracy drop, our work explores QAT on DLRM models and aims for ultra-low precision during inference without accuracy loss, while making QAT much more efficient for large-scale recommendation models specifically.

Efficient Training of Recommendation Models - Training of large-scale CTR models is distributed in the real world, and communication bottlenecks the training time. Many previous works compress DLRM communication loads to speedup training time losslessly ([23]) or lossily ([13, 31]) for a higher compression ratio. [13] improves conventional Top-k sparsification on communication during DLRM's hybrid parallelism. [31] quantizes the communication to ultra-low precision with tolerable accuracy drop through a novel error compensation scheme. Previously, [24] combines sparsification and quantization on distributed communication for CNN and ASR models. However, the merit of combining the two is not generalizable to different model architectures. In this work, we look into these techniques' effects on different parts of DQRM under the DP environment and combine specified sparsification and quantization together to further benefit DQRM training.

3 METHODOLOGY

3.1 Reducing Memory of Unused Weights

DLRM-style recommenders have over 99% of their model size occupied by embedding tables, unlike other popular architectures like CNNs or transformers. Because of these giant embedding tables, previous works [10, 19] have shown that training large-scale recommendation models is memory-bound instead of compute-bound. The memory bottleneck problem for DLRMs is further exacerbated if Quantization-aware training (QAT) is naively applied.

As shown in Figure 2 (a), in naive QAT, the entire weight tensor of convolutional or MLP layers is quantized since it is used in the subsequent computation. This requires two copies of the tensors to be stored: a low-precision and a

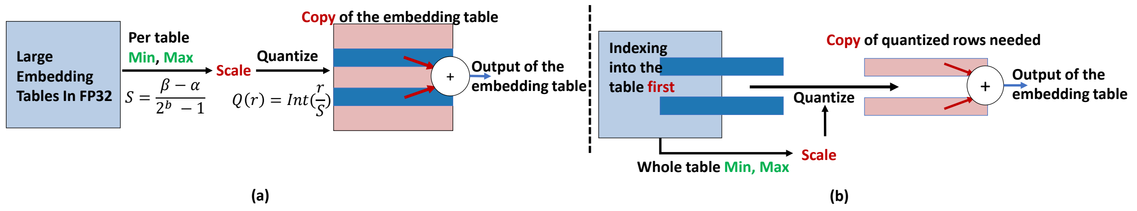
Large Embedding Tables In FP32
Per table Min, Max
$S = \frac{\beta - \alpha}{2^b - 1}$
Scale
Quantize
$Q(r) = Int(\frac{r}{S})$
Copy of the embedding table
+
Output of the embedding table
(a)
Indexing into the table first
Whole table Min, Max
Scale
Quantize
Copy of quantized rows needed
+
Output of the embedding table
(b)

Fig. 2. (a) Conventional QAT method, where the entire set of weights of the embeddings are copied and quantized. As embedding accesses are very sparse, this method is wasteful as it processes unused entries and worsens the memory bottleneck in QAT. (b) Our method to avoid the massive copy is by first performing the functional part of each table and then performing copying and quantization. In this way, only part of the weights that are actually used is copied and quantized, utilizing the memory more efficiently (Figure best viewed in color).

full-precision shadow copy. However, as shown in Figure 2 (b), since not all embedding weights contribute to subsequent computation, copying the entire embedding table in memory is unnecessary during quantization.

Therefore, our solution is to run the embedding layer first and retrieve only the necessary sparse embedding vectors to quantize, so only used embedding vectors have low-precision copies stored. This approach ensures that only active embedding table entries are copied throughout the iteration, which is usually more than three magnitudes smaller than the total number of embedding vectors in the tables on average depending on the training batch size.

3.2 Periodic Update to Find Quantization Scale

Naive QAT poses more challenges to DLRM quantization. During the quantization process, the first step is usually finding the quantization scale. Given the large embedding table size, this step is extremely costly as the program needs to traverse through an enormous chunk of contiguous memory. We run a breakdown of QAT training time shown in Figure 1 (c) under the single-node training setting for both GPU (training on Kaggle dataset) and CPU (training on Terabyte dataset). [1] For reference, without quantization, backward propagation workload dominates total single-node GPU training time: 5.8 ms out of 8.5 ms total. When QAT is applied, backprop (the yellow portion) no longer dominates, as forward propagation becomes significantly longer (indicated by the rest of the pillars). The primary reason for this increase in time is the operation of finding the scale (shown by the blue portion). Finding the scale of large embedding tables occupies more than one-third of the entire training time (left pillar) for GPU on Kaggle and dominates (over 97%) the entire training time for training on single-node CPU on Terabyte (right pillar). The problem of large tensor traversal magnifies specifically on CPUs. [2]

We found that periodic updates of the quantization scale effectively eliminate the latency of founding scales. These updates amortize the step's overhead over hundreds or thousands of iterations without hurting model convergence. We provide empirical results in Section 4 and in-depth ablation studies in Section 1.1 in the supplemental materials to show that periodic updates can even boost model accuracy.

3.3 Distributed Training with Gradient Quantization with only MLP Error Compensation

We package the entire modified quantized model for DLRM into the Deep Quantized Recommendation Model (DQRM). To make DQRM more efficient and competitive in training time compared with normal training, we further optimize the communication workload that occurs in every iteration.

[1]More experiment details can be found in Section 2.2 in Supplemental Materials.
[2]we also run single-node GPU training on Terabyte and find scale operation barely surpasses 50%.

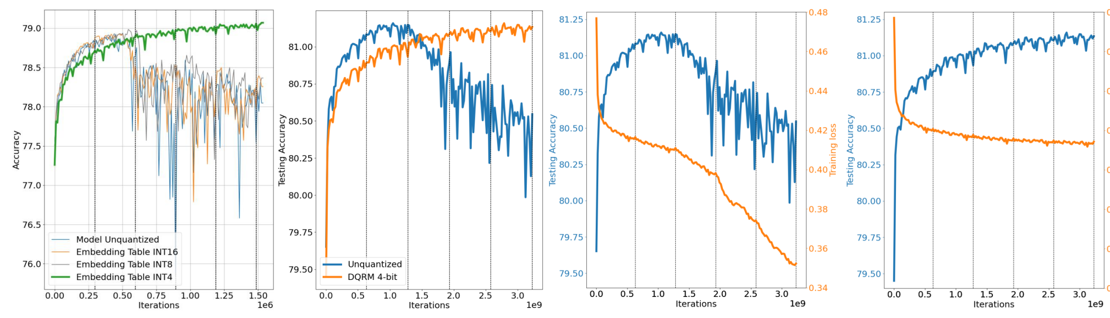

Fig. 3. (a) shows the effect of using different QAT bit widths on quantizing embedding tables in DLRM for five epochs of training (epochs are separated by the black dashed lines in all figures). QAT in uniform 4-bit overcomes the severe overfitting suffered by the original DLRM training and leads to significantly higher testing accuracy over five epochs of training. (b) shows the comparison between DQRM 4-bit compared to normal training on the Terabyte dataset; DQRM, with a significantly smaller model size, achieves on-par test accuracy as DLRM FP32 model by better 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 4-bit DQRM decreases stably throughout five epochs of training.

We break down multi-node distributed data parallelism training of DQRM on both GPUs and CPUs in Section 2.2 in supplemental materials. We found that although PyTorch's built-in specified sparsity greatly reduces all-reduce communication message load, all-reduce gradient communication after every iteration still dominates the training time. The observation motivates us to compress communication further. We explore quantizing all gradients into fixed-point INT8 format on top of the applied significantly specified sparsity. However, naively doing so is challenging. Ablation studies presented in Section 1.3 of supplemental materials show that, naively quantizing the gradient into INT8 hurts model convergence significantly. Although [31]'s error compensation scheme can prevent accuracy degradation, naively performing error compensation to all parts of the weights can also be highly inefficient due to the enormous error buffer needed for the large embedding tables, as adding large error buffers will greatly increase the model size stored in memory during training.

We identify surprisingly that gradients in MLP layers are more sensitive to quantization than those in embedding tables while embedding tables are much more robust when specified gradients are heavily quantized, shown empirically in Section 1.2 of supplemental materials. Therefore, we choose to only compensate the MLP layers for gradient quantization. We achieve reasonable accuracy degradation while compressing the communication message size by roughly 4× on top of the already heavy and well-supported gradient sparsification added to embedding tables. Detailed experimental settings, results, and evaluations are presented in Section 4.3.

4 EXPERIMENTAL RESULTS

In this section, we present results evaluating DQRM on two popular datasets for CTR: the Criteo Kaggle Display Advertising Challenge Dataset (shortened below as the Kaggle dataset) and the Criteo Terabyte Dataset (shortened as the Terabyte dataset). We followed [21]'s mlperf DLRM optimal configurations for each dataset as summarized in Table 2.

Table 1. DLRM model architecture configurations

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

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

Quantization	Testing	
Bit Width	Accuracy	ROC AUC
Full-precision	78.923%	0.8047
INT16	78.928% (+0.005%)	0.8046 (-0.0001)
INT8	78.985% (+0.062%)	0.8054 (+0.0007)
INT4	**79.092% (+0.169%)**	**0.8073 (+0.0026)**

4.1 Quantization of Embedding Tables

Embedding tables occupy 99% of DLRM, which motivates heavy model compression of embedding tables. We used one Nvidia A5000 GPU to study the effect of quantization on the embedding tables. Different bit widths for embedding table quantization are used: INT16, INT8, INT4. Unlike previous works [4, 11] that utilize a row-wise scheme for embedding table quantization, we quantize the embedding table with a per-table scheme, as it reduces the number of FP32 scales stored and retrieved every iteration.

Figure 3(a) illustrates testing accuracy curves for various quantization bit widths. DLRM single precision (blue) overfits after the second epoch, causing a steep accuracy drop, which is a common issue in large-scale CTR models as recently explored in [35]. INT16 (orange) and INT8 (grey) follow similar overfitting patterns. However, INT4 (green) converges slower but overcomes overfitting, increasing accuracy over five epochs. In supplementary materials, we empirically show that INT4 convergence lasts up to 16 epochs. We also found that INT2 prevents overfitting like INT4, although it incurs a greater accuracy degradation.

We compare the performance after embedding table quantization in Table 2. Uniform INT4 quantization outperforms the original model in testing accuracy by roughly 0.15% and ROC AUC score by 0.0047 and achieving 8× reduction in embedding table size. The accuracy improvement is significant in the Kaggle dataset. We studied the weight distribution of the training of FP32 weights, INT8, and INT4. We found that the range of distribution for the single-precision weights constantly shifted outwards, while the INT8 weights closely followed this trend. In contrast, INT4 only loosely captures the trend, changing the entire weight distribution much slower compared to higher precision weights. We hypothesize that such observation is linked to DQRM INT4 strong regularization ability. Details are in Section 1.6 of supplementary materials.

4.2 Quantization of the Whole Model

DLRM models have two components: Embedding tables and MLP layers. Compared with the embedding table, we observe that MLP layers are more sensitive to quantization, aligning with [4]. Channel-wise quantization of MLP layers performs better, as it has hardly any accuracy drop from the single-precision MLP, while INT4 matrix-wise MLP quantization badly converges. Additional ablation studies are presented in Section 1.3 of supplementary materials.

We evaluate DQRM INT4 on both the Kaggle and Terabyte datasets. We used the same experiment platforms for the Kaggle model as in Section 4.1. However, to meet the demanding resource required by the Terabyte models, we use the Intel Academic Compute Environment (ACE) CPU clusters and specifically Intel(R) Xeon(R) Platinum 8280 CPUs for model training and inference. The quantized models are all trained in DQRM for five epochs till convergence.

Figure 3 (b), (c), and (d) display training loss and testing accuracy curves for the Terabyte dataset. In (b), original model testing accuracy (blue) overfits at the 2nd epoch, while DQRM INT4 (orange) steadily rises and avoids overfitting.

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

(a) 4-bit quantization for DLRM on Kaggle

Quant Settings	Model Bit Width	Model Size	Training loss	Training time/it	Testing Accuracy	Testing ROC AUC
Baseline	FP32	2.161 GB	0.304	7 ms	78.923%	0.8047
Vanilla PTQ	INT4	0.270 GB	-	-	76.571%	0.7675
PACT* [2]	INT4	0.270 GB	Cannot Converge			
(MLP in FP32)		0.271 GB	0.303	69 ms	78.858%	0.8040
LSQ [7]	INT4	0.270 GB	0.350	25 ms	78.972%	0.8051
(MLP in FP32)		0.271 GB	0.352	21 ms	78.987%	0.8059
HAWQ [6]	INT4	0.270 GB	0.437	31 ms	79.040%	0.8064
(MLP in FP32)		0.271 GB	0.436	27 ms	79.070%	0.8075
DQRM (Ours)	INT4	0.270 GB	0.437	22 ms	79.071%	0.8073
(MLP in FP32)		0.271 GB	0.436	20 ms	79.092%	0.8073

*PACT [2] uses DoReFa [36] for weight quantization

(b) 4-bit quantization for DLRM on Terabyte

Quant Settings	Model Bit Width	Model Size	Training loss	Training time/it	Testing Accuracy	Testing ROC AUC
Baseline	FP32	12.575 GB	0.347071	19 ms	81.165%	0.8004
Vanilla PTQ	INT4	1.572 GB	-	-	78.681%	0.7283
PACT* [7]	INT4	1.572 GB	Cannot Finish >1000 ms/it			
HAWQ [6]	INT4	1.572 GB	Cannot Finish >1000 ms/it			
LSQ [7]	INT4	1.572 GB	0.350	42 ms	81.134%	0.7996
(MLP in FP32)		1.572 GB	0.356	42 ms	81.127%	0.7998
DQRM (Ours)	INT4	1.572 GB	0.409774	29 ms	81.210%	0.8015
(MLP in FP32)		1.572 GB	0.412	29 ms	81.200%	0.8010

Comparing DLRM (c) and DQRM INT4 (d), the latter demonstrates better regularization, with a consistent decrease in loss (orange curve in (d)) and a plateau in testing accuracy (blue curve in (d)). In contrast, DLRM's accuracy (blue curve in (c)) and training loss (orange curve in (c)) crash after the second epoch.

We report single-node training and testing performance in Table 3. We compare DQRM INT4 against PTQ and prior popular QAT works [2, 6, 7] that achieve strong INT4 results on CNNs. Unfortunately, [4] does not open-source their INT4 PTQ implementation, so we apply vanilla PTQ using our quantization method with channel-wise quantization for MLP. Also, we implemented [2, 7] ourselves on DLRM. Table 3 (a) shows the experimental results on the Kaggle dataset using GPU. Using an update period of 200, DQRM achieves both a higher testing accuracy (79.071%) and a higher testing ROC AUC score (0.8073) than DLRM in FP32. On the Terabyte dataset in (b), by using an update period of 1000, DQRM INT4 achieves slightly higher than FP32 baseline test accuracy (0.45%) and testing ROC AUC (0.0011). The vanilla PTQ incurs a significant accuracy drop in both datasets. Interestingly, prior QAT works exhibit their inefficiencies. PACT [2] and LSQ [7] do not avoid overfitting completely like DQRM and, thus, achieves lower testing accuracy on both datasets. HAWQ [6] cannot effectively eliminate unnecessary memory traversal, making it consistently slower than DQRM in training time and even failing to finish on Terabyte. Extended evaluation of prior QAT on DLRM is shown in Section 1.1 of supplementary materials.

4.3 Adding Gradient Quantization on Specified Sparsity

Our experiments are based on synchronous data parallelism (DP). However, we argue that the gradient compression techniques presented are directly applicable to hybrid parallelism settings as in [21]. Moreover, the only difference between the data parallelism and the DLRM's hybrid parallelism is the all2all round of communication, and it can also be directly benefited from DQRM INT with message load shrunk by 8× directly. Due to PyTorch `DistributedDataParallel` library limitations, we customized and open-sourced our own Data Parallel library implementing gradient quantization before `allreduce`. Section 2.1 in supplementary Materials details the implementation. Our approach supports multi-GPU and CPU node platforms, with gradient compression experiment results in Table 4.

Table 4. Communications compression for Distributed Data Parallelism training among four nodes or GPUs

Model	Training	Communication	Communication	Latency	Training	Testing	
Settings	Platforms	Compression settings	Overhead per iter	per iter	Loss	Accuracy	ROC AUC
Kaggle	4X	gradient uncompressed (DQRM 4-bit)	2.161 GB	>1000 ms	0.436685	78.897%[1]	0.8035
	Nvidia A5000	+ EMB gradient sparsification (specified)[2]	2.010 MB	61 ms	0.436685	78.897%	0.8035
	GPUs	+ INT8 gradient Quantization	0.509 MB	110 ms[3]	0.442300	78.840%	0.8023
Terabyte	2X (2 processes)	gradient uncompressed (DQRM 4-bit)	12.575 GB	>1000 ms	0.412688	81.156%	0.7997
	Intel(R) Xeon(R)	+ EMB gradient sparsification (specified)	6.756 MB	210 ms	0.412688	81.156%	0.7997
	Platinum 8280 CPU	+ INT8 gradient Quantization	1.732 MB	225 ms	0.414731	81.035%	0.7960

[a]data parallelism consistently lowers the test accuracy in all settings compared with single-node training
[b]Specified sparsification is a lossless compression for embedding tables so the testing accuracy is exactly the same as uncompressed case
[c]PyTorch sparse tensor allreduce library doesn't support low-precision arithmetic, without further system level effort in low-precision optimization, the latency per iteration increases purely from the quantization overhead per iteration

For experiments on the Kaggle dataset, we utilized four NVIDIA A5000 GPUs. For Terabyte dataset, we ran on two Intel(R) Xeon(R) Platinum 8280 CPU nodes. Naively, because of the giant model size, when the gradient is completely uncompressed, the overhead of gradient communication is up to 2.1 GB per iteration. PyTorch has built-in support to exploit specified sparsity in the EmbeddingBag modules which is very powerful in the allreduce in DP and can compress 2.1 GB to 2 MB. Moreover, using specified sparsity is lossless and has no impact on model training performance.

Surprisingly, after the specified sparsity is applied, the MLP gradient becomes significant among the total gradient left. After careful profiling, from our profiling, occupying around 95% and 45% in size among the remaining gradient for training in DP on Kaggle and Terabyte respectively. We further add error compensation to the MLP layers to reduce accuracy degradation. PyTorch doesn't support low-precision sparse allreduce well. With limited backend support and significant communication overhead caused by sparse tensor coalescing, achieving speedup from communication reduction remains difficult. With in-depth system-level optimization for low-precision sparse all reduce and gradient quantization workload, the training time cost shrink of distributed DQRM can be realistically fulfilled, but such an endeavor is outside the scope of our current work. Nevertheless, we showed empirically that adding gradient quantization only introduces a trivial decrease of 0.057% in the testing accuracy and 0.0012 in the ROC AUC score despite a significant reduction in communication size for the Kaggle dataset.

Similarly, gradient compression of the embedding tables introduces insignificant accuracy loss with roughly 0.1% for testing accuracy and less than 0.004 for testing ROC AUC. We detail more evaluations of MLP gradient sensitivity for quantization and the effect of error compensation in section 1.2 of supplementary Materials.

5 CONCLUSION

In this work, we propose a systematic quantization framework DQRM for large-scale recommendation models. Specifically, we discover that the DLRM model suffers severely from the overfitting problem. We show that ultra-low precision quantization can help overcome the strong overfitting and better utilize the training dataset, which eventually leads to higher test accuracy. We observe that conventional QAT is troublesome in training large-scale recommendation models and we propose two techniques that significantly alleviate the issue. Besides, to further optimize DQRM under the distributed environment, we combine specified sparsification and quantization together to compress communications. Our framework is intensively evaluated on the published dataset Kaggle and Terabyte, where we outperform the full-precision DLRM baselines while achieving an $8\times$ reduction of model size.

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] Jungwook Choi, Zhuo Wang, Swagath Venkataramani, Pierce I-Jen Chuang, Vijayalakshmi Srinivasan, and Kailash Gopalakrishnan. 2018. Pact: Parameterized clipping activation for quantized neural networks. *arXiv preprint arXiv:1805.06085* (2018).

[3] 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).

[4] 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.

[5] Aditya Desai, Li Chou, and Anshumali Shrivastava. 2021. Random Offset Block Embedding Array (ROBE) for CriteoTB Benchmark MLPerf DLRM Model: 1000 times Compression and 3.1 times Faster Inference. *arXiv preprint arXiv:2108.02191* (2021).

[6] Zhen Dong, Zhewei Yao, Amir Gholami, Michael W Mahoney, and Kurt Keutzer. 2019. Hawq: Hessian aware quantization of neural networks with mixed-precision. In *Proceedings of the IEEE/CVF International Conference on Computer Vision*. 293–302.

[7] Steven K Esser, Jeffrey L McKinstry, Deepika Bablani, Rathinakumar Appuswamy, and Dharmendra S Modha. 2019. Learned step size quantization. *arXiv preprint arXiv:1902.08153* (2019).

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

[9] 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

[10] Antonio A Ginart, Maxim Naumov, Dheevatsa Mudigere, Jiyan Yang, and James Zou. 2021. Mixed dimension embeddings with application to memory-efficient recommendation systems. In *2021 IEEE International Symposium on Information Theory (ISIT)*. IEEE, 2786–2791.

[11] 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).

[12] 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

[13] 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.

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

[15] 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.

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

[17] 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.

[18] 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.

[19] 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.

[20] Shiwei Li, Huifeng Guo, Lu Hou, Wei Zhang, Xing Tang, Ruiming Tang, Rui Zhang, and Ruixuan Li. 2022. Adaptive Low-Precision Training for Embeddings in Click-Through Rate Prediction. *arXiv preprint arXiv:2212.05735* (2022).

[21] 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

[22] 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

[23] Sarunya Pumma and Abhinav Vishnu. 2021. Semantic-Aware Lossless Data Compression for Deep Learning Recommendation Model (DLRM). In *2021 IEEE/ACM Workshop on Machine Learning in High Performance Computing Environments (MLHPC)*. IEEE, 1–8.

[24] 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

[25] 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

[26] 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.

[27] 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

[28] 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.

[29] Xiaorui Wu, Hong Xu, Honglin Zhang, Huaming Chen, and Jian Wang. 2020. Saec: similarity-aware embedding compression in recommendation systems. In *Proceedings of the 11th ACM SIGOPS Asia-Pacific Workshop on Systems*. 82–89.

[30] Zhiqiang Xu, Dong Li, Weijie Zhao, Xing Shen, Tianbo Huang, Xiaoyun Li, and Ping Li. 2021. Agile and accurate CTR prediction model training for massive-scale online advertising systems. In *Proceedings of the 2021 international conference on management of data*. 2404–2409.

[31] 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)*.

[32] Chunxing Yin, Bilge Acun, Carole-Jean Wu, and Xing Liu. 2021. Tt-rec: Tensor train compression for deep learning recommendation models. *Proceedings of Machine Learning and Systems* 3 (2021), 448–462.

[33] 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

[34] Jian Zhang, Jiyan Yang, and Hector Yuen. 2018. Training with low-precision embedding tables. In *Systems for Machine Learning Workshop at NeurIPS*, Vol. 2018.

[35] Zhao-Yu Zhang, Xiang-Rong Sheng, Yujing Zhang, Biye Jiang, Shuguang Han, Hongbo Deng, and Bo Zheng. 2022. Towards understanding the overfitting phenomenon of deep click-through rate prediction models. *arXiv preprint arXiv:2209.06053* (2022).

[36] Shuchang Zhou, Yuxin Wu, Zekun Ni, Xinyu Zhou, He Wen, and Yuheng Zou. 2016. Dorefa-net: Training low bitwidth convolutional neural networks with low bitwidth gradients. *arXiv preprint arXiv:1606.06160* (2016).

[37] Jieming Zhu, Jinyang Liu, Shuai Yang, Qi Zhang, and Xiuqiang He. 2020. Fuxictr: An open benchmark for click-through rate prediction. *arXiv preprint arXiv:2009.05794* (2020).