Adaptive Low-Precision Training for Embeddings in Click-Through Rate Prediction

Shiwei Li[1*], **Huifeng Guo**[2], **Lu Hou**[2], **Wei Zhang**[2], **Xing Tang**[2], **Ruiming Tang**[2], **Rui Zhang**[3†], **Ruixuan Li**[1†]
[1] Huazhong University of Science and Technology, Wuhan, China
[2] Huawei Noah's Ark Lab, Shenzhen, China
[3] Tsinghua University, Beijing, China
{d202181195, rxli}@hust.edu.cn, {rayteam}@yeah.net,
{huifeng.guo, houlu3, xing.tang, tangruiming}@huawei.com

Abstract

Embedding tables are usually huge in click-through rate (CTR) prediction models. To train and deploy the CTR models efficiently and economically, it is necessary to compress their embedding tables. To this end, we formulate a novel quantization training paradigm to compress the embeddings from the training stage, termed low-precision training (LPT). Also, we provide theoretical analysis on its convergence. The results show that stochastic weight quantization has a faster convergence rate and a smaller convergence error than deterministic weight quantization in LPT. Further, to reduce accuracy degradation, we propose adaptive low-precision training (ALPT) which learns the step size (i.e., the quantization resolution). Experiments on two real-world datasets confirm our analysis and show that ALPT can significantly improve the prediction accuracy, especially at extremely low bit width. For the first time in CTR models, we successfully train 8-bit embeddings without sacrificing prediction accuracy.

1 Introduction

Click-through rate (CTR) prediction is to predict the probability that a user will click on a recommended item under a specific context (Cheng et al. 2016), which is a critical component in recommender systems. It is widely used in various scenarios, such as online shopping (Zhou et al. 2018) and advertising (McMahan et al. 2013). With the development of deep neural networks, CTR models evolve from logistic regression (McMahan et al. 2013), factorization machine models (Juan et al. 2016) to deep learning models. Various deep CTR models have been proposed and deployed in industrial companies, such as Wide & Deep (Cheng et al. 2016) in Google, DIN (Zhou et al. 2018) in Alibaba and and DeepFM (Guo et al. 2017) in Huawei.

Deep CTR models usually follow the *embedding table* and *neural network* paradigm (Guo et al. 2021a). As shown in Figure 1, the *embedding table* transforms the high-dimensional one-hot encoded vectors of categorical features (e.g., `user_id` and `item_id`) into low-dimensional real-valued vectors (i.e., embeddings) (Guo et al. 2017). The *neural network* is used to model feature interactions and make predictions. Usually, each feature has a unique embedding stored in the embedding table $\mathbf{E} \in \mathbb{R}^{n\times d}$, where n is the number of features and d is the embedding dimension. However, since there are usually billions or even trillions of categorical features, embedding tables may take hundreds of GB or even TB to hold (Guo et al. 2021b). For example, the size of the embedding tables in Baidu's advertising systems reaches 10 TB (Xu et al. 2021).

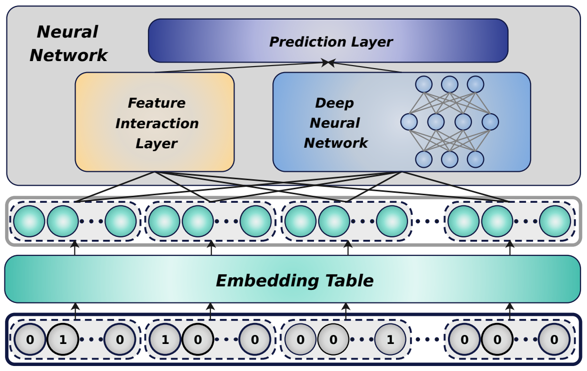


Figure 1: The *embedding table* and *neural network* paradigm of click-through rate prediction models.

To deploy the CTR models with huge embedding tables in real production systems efficiently and economically, it is necessary to compress their embedding tables. Most research on embedding compression in recommender systems focuses on three aspects: (i) NAS-based embedding dimension search (Joglekar et al. 2020; Zhao et al. 2020); (ii) Embedding pruning (Deng et al. 2021; Liu et al. 2021); (iii) Hashing (Shi et al. 2020; Zhang et al. 2020). Unfortunately, these approaches are usually not practical in real recommender systems. Specifically, NAS-based approaches require additional storage and complex calculations to search for the optimal embedding dimension. Embedding pruning approaches usually produce unstructured embedding tables which will cost extra effort to access. Note that the mod-

*This work was done when Shiwei Li worked as an intern at Noah's Ark Lab, Huawei.

†Corresponding authors.

els trained with these two approaches should be retrained to maintain the accuracy. Besides, they can not compress the embeddings at the training stage, which although hashing approaches can do, they usually cause severe degradation in prediction accuracy (Xu et al. 2021; Shi et al. 2020). In industrial recommender systems, the CTR models with extremely large embedding tables usually require distributed training on multiple devices (Xu et al. 2021), where the communication between multiple devices seriously affects the training efficiency. By compressing the embeddings at training stages, CTR models can be trained on less devices or even one single GPU, which can accelerate training by reducing the communication overhead.

In this paper, we aim to compress the large embedding tables in CTR models from the training stage without sacrificing accuracy. As far as we know, the quantization schemes in existing work (Xu et al. 2021; Yang et al. 2020) are also proposed for the same purpose. We term this quantization scheme as low-precision training (LPT). Specifically, LPT keeps the weights in low-precision format during training. Different from the commonly used quantization-aware training (QAT) (Esser et al. 2020) which keeps a copy of full-precision weights for parameter update, LPT directly updates low-precision weights and then quantizes the updated full-precision weights back into low-precision format.

However, without the copy of full-precision weights, LPT usually suffers from inferior performance than QAT. (Xu et al. 2021) claims that their maximum ability is using 16-bit LPT for embeddings, since lower bit-width will cause unacceptable accuracy degradation. Although (Yang et al. 2020) achieves lossless 8-bit LPT for embeddings, they have to keep a full-precision cache which brings extra memory cost. The efforts of (Xu et al. 2021) and (Yang et al. 2020) illustrate that training low-precision embeddings is very challenging, especially in the case of low bit-width. However, they are only empirical and lack theoretical analysis for LPT. Besides, they did not explore the impact of the step size (i.e., the quantization resolution) on the accuracy. Therefore, to explore the limitation of LPT, we first provide theoretical analysis on its convergence. Further, to reduce accuracy loss, we propose adaptive low-precision training to learn the step size. The contributions are summarized as follows:

(1) We formulate a low-precision training paradigm to compress embedding tables from the training stage and provide theoretical analysis on its convergence. The results show that stochastic weight quantization has a faster convergence rate and a smaller convergence error than deterministic weight quantization in LPT.

(2) Different from previous studies, we offer a solution to learn the step size by gradient descent in LPT, termed adaptive low-precision training (ALPT).

(3) Experiments are conducted on two public real-world datasets for CTR predictions. The results confirm our theoretical analysis and show that ALPT can significantly improve the prediction accuracy. The code of ALPT is publicly available[1].

[1]https://gitee.com/mindspore/models/tree/master/research/recommend/ALPT

2 Preliminaries

In this section, we elaborate on the training process of LPT. In Section 2.1, we first introduce how quantization works. In Section 2.2, we introduce the commonly used QAT, as it is important to understand how LPT differs from it. In Section 2.3, we introduce LPT and explain why it can compress the embedding tables of CTR models at the training stage.

2.1 Quantization

Quantization compresses a network by replacing the 32-bit full-precision weights with their lower-bit counterparts without changing the network architecture. Specifically, for m-bit quantization, the set of quantized values can be denoted as $\mathbb{S} = \{b_0, b_1, ..., b_k\}$, where $k = 2^m - 1$. One commonly used quantization scheme is the uniform quantization, where the quantized values are uniformly distributed, and is usually more hardware-friendly than the non-uniform quantization. In the uniform symmetric quantization, the step size $\Delta = b_i - b_{i-1}$ remains the same for any $i \in [1, k]$, and $b_0 = -2^{m-1}\Delta, b_k = (2^{m-1} - 1)\Delta$.

In this paper, we adopt uniform symmetric quantization on the embeddings. Specifically, given the step size Δ and the bit width m, a full-precision weight w is quantized into $\hat{w} \in \mathbb{S}$, which is represented by the multiplication of the step size Δ and an integer $\tilde{w}$:

$$\tilde{w} = R(clip(w/\Delta, -2^{m-1}, 2^{m-1} - 1)), \qquad (1)$$

$$\hat{w} = Q(w) = \Delta \times \tilde{w}, \qquad (2)$$

where $clip(v, n, p)$ returns v with values below n set to n and values above p set to p, $R(v)$ rounds v to an adjacent integer. There are generally two kinds of rounding functions:

Deterministic Rounding (DR) rounds a floating-point value to its nearest integer:

$$R_D(x) = \begin{cases} \lfloor x \rfloor & \text{if } x - \lfloor x \rfloor < 0.5, \\ \lfloor x \rfloor + 1 & \text{otherwise.} \end{cases} \qquad (3)$$

Stochastic Rounding (SR) rounds a floating-point value to its two adjacent integers with a probability distribution:

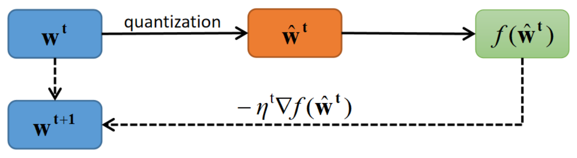
$$R_S(x) = \begin{cases} \lfloor x \rfloor & \text{w.p.} \quad \lfloor x \rfloor + 1 - x, \\ \lfloor x \rfloor + 1 & \text{w.p.} \quad x - \lfloor x \rfloor. \end{cases} \qquad (4)$$

To distinguish the difference in the rounding function, we add subscript for the quantization function in Eq. (2), that is $Q_D()$ and $Q_S()$ for DR and SR, respectively. Note that Δ is also an input of the quantization functions $Q(w, \Delta)$, here we omit Δ for simplicity.

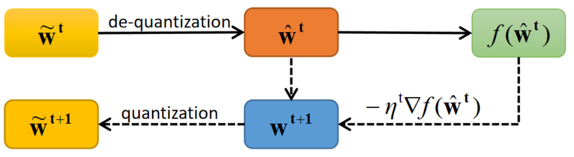
2.2 Quantization-Aware Training

As Figure 2(a) shows, quantization-aware training (QAT) quantizes the full-precision weights in the forward pass and updates the full-precision weights with the gradients estimated by straight through estimator (STE) (Courbariaux, Bengio, and David 2015). Specifically, let $f(\cdot)$ be the loss function and $\nabla f(\hat{\mathbf{w}}^t)$ be the gradients w.r.t. the quantized weights $\hat{\mathbf{w}}^t$. For stochastic gradient descent with a learning rate of η^t, the full-precision weights will be updated as:

$$\mathbf{w}^{t+1} = \mathbf{w}^t - \eta^t \nabla f(\hat{\mathbf{w}}^t). \qquad (5)$$



(a) Quantization-aware training



(b) Low-precision training

Figure 2: Training processes of quantization-aware training (QAT) and low-precision training (LPT).

To achieve better accuracy, recent methods have offered solutions to learn the step size. For example, LSQ (Esser et al. 2020) uses the following quantization function:

$$Q_D(w_i^t) = \Delta \times R_D(\text{clip}(\frac{w_i^t}{\Delta}, -2^{m-1}, 2^{m-1}-1)), \quad (6)$$

and optimizes Δ through gradient descent where the gradient of the step size Δ is estimated by:

$$\frac{\partial Q_D(w_i^t)}{\partial \Delta} = \begin{cases} -2^{m-1} & \text{if } w_i^t/\Delta \le -2^{m-1}, \\ 2^{m-1}-1 & \text{if } w_i^t/\Delta \ge 2^{m-1}-1, \\ R_D(\frac{w_i^t}{\Delta}) - \frac{w_i^t}{\Delta} & \text{otherwise.} \end{cases} \quad (7)$$

After training, each weight matrix can be stored in the format of integers plus one full-precision step size. However, in the training process of QAT, the full-precision weights are involved in the update process, which means QAT can only compress the model size for inference.

2.3 Low-Precision Training

As Figure 2(b) shows, unlike QAT which still keeps a copy of full-precision weights for parameter update, low-precision training (LPT) directly updates low-precision weights and then quantize the updated full-precision weights back into low-precision format as:

$$\hat{\mathbf{w}}^{t+1} = Q(\hat{\mathbf{w}}^t - \eta^t \nabla f(\hat{\mathbf{w}}^t)). \quad (8)$$

For the embedding tables in CTR models, LPT is quite effective in compression of training memory. Note that embedding tables are usually highly sparse and each batch of the training data only covers very few features. For example, in our processed dataset Avazu which has 24 feature fields and more than 4 million features, a batch of ten thousand samples only contains 1400 features on average. With LPT, the whole embedding table can be stored in the format of integers and only the embeddings of very few features that appear in each batch will be de-quantized into floating-point values for calculation and update. The storage of the step size and the de-quantized floating point weights are negligible compared to the embedding tables. In this way, LPT can effectively compress the model at the training stage, however, the issue of inferior accuracy remains to be resolved. In this paper, we aim to improve the accuracy of LPT from the perspective of the rounding function and the step size.

3 Methodology

As discussed in Section 2.3, we need to quantize the updated full-precision weights back into low-precision format in LPT. When choosing a quantization function, we have to consider two key issues: (i) which rounding function suits LPT better? (ii) how to select reasonable step size flexibly and effectively? To address the first issue, we first analyze the convergence of LPT in Section 3.1. To address the second issue, in Section 3.2, we first point out the difficulties of learning the step size in LPT, then we introduce our adaptive low-precision training algorithm.

3.1 Rounding: Stochastic or Deterministic?

Generally, DR is the common choice of quantization as it produces lower mean-square-error (MSE) (Esser et al. 2020; Choi et al. 2018). Still, recent works (Xu et al. 2021; Li et al. 2017) argue that SR can achieve better performance in LPT. However, they did not provide theoretical analysis about the difference between SR and DR in LPT. Therefore, in this section, we first provide theoretical analysis on the convergence of SR and DR in LPT, and then we use a synthetic experiment to visualize the difference between SR and DR.

Convergence Analysis To analyze the convergence for LPT, we consider empirical risk minimization problems as in Eq.(9) following (Li et al. 2017). $\mathbf{w}$ is used to denote all the parameters in a model and the loss function is decomposed into a sum of multiple loss functions $\mathbb{F} = \{f_1, f_2, ..., f_m\}$:

$$\min_{\mathbf{w}\in W} F(\mathbf{w}) := \frac{1}{m}\sum_{i=1}^{m} f_i(\mathbf{w}). \quad (9)$$

At the t-th iteration of the gradient descent, we select a function $f^t \in \mathbb{F}$ and update the model parameters as:

$$\mathbf{w}^{t+1} = \mathbf{w}^t - \eta^t \nabla f^t(\mathbf{w}^t). \quad (10)$$

(Li et al. 2017) already provides convergence analysis for SR in LPT (Theorem 1). However, it lacks the analysis for DR. Since DR is biased (i.e., the expectation of the quantization error is not zero), it is quite challenging to extend the conclusion of Theorem 1 to DR. In Theorem 2, we provide convergence analysis for DR in LPT, using the same assumptions as (Li et al. 2017): (i) each $f_i \in \mathbb{F}$ is differentiable and convex; (ii) $f^t(\mathbf{w}^t)$ has bounded gradient, i.e., $\mathbf{E}||\nabla f^t(\mathbf{w}^t)||^2 \le G^2$; and (iii) the domain of $\mathbf{w}$ is a convex set and has finite diameter, i.e., $||\mathbf{w}^m - \mathbf{w}^n)||^2 \le D^2$.

Theorem 1 *[Theorem 2 in (Li et al. 2017)] Assume the learning rate decays like $\eta^t = \frac{\eta}{\sqrt{t}}$. At the T-th iteration, with a fixed step size Δ, for SR in LPT, we have:*

$$\mathbf{E}\left[F(\bar{\mathbf{w}}^T) - F(\mathbf{w}^*)\right] \le \frac{D^2}{2\eta\sqrt{T}} + \frac{\eta G^2}{\sqrt{T}} + \frac{\sqrt{d}\Delta G}{2}, \quad (11)$$

where $\bar{\mathbf{w}}^T = \frac{1}{T}\sum_{t=1}^{T}\mathbf{w}^t$, $\mathbf{w}^* = \arg\min_{\mathbf{w}} F(\mathbf{w})$ and d is the dimension of $\mathbf{w}$.

Theorem 2 *Assume the learning rate decays like $\eta^t = \frac{\eta}{\sqrt{t}}$ and let $T_0 = \lfloor \frac{2\eta G}{\sqrt{d}\Delta} \rfloor$. At the T-th iteration, with a fixed step size Δ, for DR in LPT, we have:*

$$\mathbf{E}\left[F(\bar{\mathbf{w}}^T) - F(\mathbf{w}^*)\right] \leq \frac{D^2}{2\eta\sqrt{T}} + \frac{3\eta G^2}{\sqrt{T}} + \frac{\sqrt{d}\Delta G}{2} + \frac{\sqrt{d}D\Delta\sum_{t=1}^{T_0}\sqrt{t}}{2\eta T} + \frac{\sum_{t=T_0+1}^{T} DG}{T}. \tag{12}$$

The detailed proofs can be found here[2]. The theorems illustrate that both DR and SR converge to an accuracy floor in LPT. However, DR has a slower convergence rate and a larger convergence error than SR, which proves that SR is more suitable for LPT.

Remark 1 *If gradient updates are extremely small and $|\eta^t \nabla f(w_i^t)| < \frac{\Delta}{2}$, DR will erase the update of w_i^t and w_i may never change from w_i^t. Such error accumulation results in slower convergence rates and larger convergence errors.*

Remark 2 *Here we only provide analysis based on a fixed step size. Empirically, we shall show in Section 4 that using an adaptive step size in Section 3.2 achieves better accuracy.*

Synthetic Experiment To visualize the difference of DR and SR in LPT, we design a simple convex problem:

$$\min_{w} f(w) = (w - 0.5)^2. \tag{13}$$

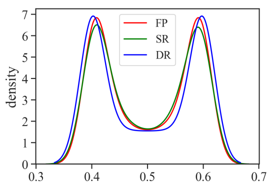
We initialize 1000 parameters between 0 and 1 uniformly. Each parameter is updated by SGD with learning rate $\eta = 1$. We set $\Delta = 0.01$ and $m = 8$ for quantization. Figures 3(a), 3(b), 3(c) show the distribution of the parameters at the iteration $t = 10$, 100 and 1000, respectively. As we can see, SR have a similar or even faster convergence rate compared to the full-precision training, while DR seem to be stagnant as described in Remark 1. As Figure 3(d) shows, in LPT with DR, all the gradients satisfy $|\eta^t \nabla f(w^t)| < \frac{\Delta}{2}$ after 10 iterations, that is the moment when the parameters stop updating.

3.2 Adaptive Low-Precision Training

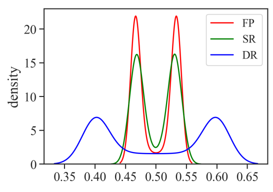
In the previous section, we have demonstrated that SR is more suitable for LPT, yet another key factor that affects the performance is the step size. Intuitively, since the representation range is proportional to the step size given the bit-width, a small step size will lead to a limited representation range. On the contrary, a large step size can not provide fine resolution to the majority of weights within the clipping range. Therefore, it is important to get a reasonable step size. To figure out how to find the desired step size, we first consider two preliminary problems:

- **Manually or adaptively?** If we set the step size as a hyper-parameter, it will take a lot of human efforts to tune the step size for different applications. Nevertheless, the tuned result may still not be optimal. Thus, we should make the step size adaptive, that is to learn the step size together with the embeddings in an end-to-end manner.
- **Global-wise or feature-wise?** A global step size will be shared throughout the whole embedding table, which will fail to trade off the representation range and precision for all embeddings. Besides, when a global step size updates, all the embeddings are supposed to be re-quantized with the updated step size, which significantly reduce the training efficiency. On the contrary, a feature-wise step size is bound with its owner feature and is free to update.

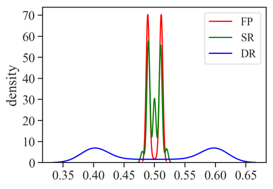
[2]https://arxiv.org/abs/2212.05735



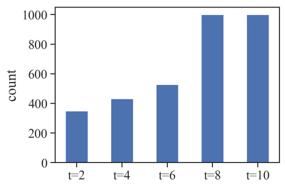
(a) $t = 10$



(b) $t = 100$



(c) $t = 1000$



(d) Gradients in DR

Figure 3: (a), (b) and (c) plot the distributions of the parameters. FP stands for training with full-precision training; DR and SR stands for low-precision training with DR and SR, respectively. (d) plots the number of parameters that satisfy $|\eta^t \nabla f(w^t)| < \frac{\Delta}{2}$ at different iterations of DR.

In light of the above discussions, we should learn a step size for each embedding. However, there is no gradients for the step size in LPT as the quantization is taken place after the backward propagation. To learn the step size, we shall introduce the step size to the forward propagation. However, even if we quantize the low-precision embeddings in the forward propagation like QAT does, the gradients of the step size will always be zero as the embeddings are already in low-precision format. To overcome the above challenges, we propose adaptive low-precision training (ALPT), that is to learn the low-precision embeddings and the step size alternately as described in Algorithm 1.

Specifically, we learn the model parameters and the step size in two separate steps. Here we use b as a subscript to denote the embeddings or the step size of the features in a batch of the input data and use $\tilde{Q}(\cdot)$ to denote the quantization function that returns the quantized integers. In the first step, the integer embeddings $\tilde{\mathbf{w}}_b^t$ will be de-quantized into floating-point values $\hat{\mathbf{w}}_b^t$ for the forward propagation, and then $\hat{\mathbf{w}}_b^t$ will be updated into full-precision weights $\mathbf{w}_b^{t+1}$. In the second step, the model will take $\mathbf{w}_b^{t+1}$ as input and quantize them with $\mathbf{\Delta}_b^t$ in the forward propagation similar

```
Algorithm 1: Adaptive low-precision training
Input: integer embeddings $\tilde{\mathbf{w}}_b^t$, step size $\mathbf{\Delta}_b^t$, other parameters $\mathbf{w}_o^t$ and the loss function $f$.
    # Step 1: Update the weights
1: $\hat{\mathbf{w}}_b^t = \mathbf{\Delta}_b^t \tilde{\mathbf{w}}_b^t$;
2: $\mathbf{w}_b^{t+1} = \hat{\mathbf{w}}_b^t - \eta^t \frac{\partial f(\hat{\mathbf{w}}_b^t, \mathbf{w}_o^t)}{\partial \hat{\mathbf{w}}_b^t}$;
3: $\mathbf{w}_o^{t+1} = \mathbf{w}_o^t - \eta^t \frac{\partial f(\hat{\mathbf{w}}_b^t, \mathbf{w}_o^t)}{\partial \mathbf{w}_o^t}$.
    # Step 2: Update the step size
4: $\mathbf{\Delta}_b^{t+1} = \mathbf{\Delta}_b^t - \eta^t \frac{\partial f(Q_D(\mathbf{w}_b^{t+1}, \mathbf{\Delta}_b^t), \mathbf{w}_o^{t+1})}{\partial \mathbf{\Delta}_b^t}$;
5: $\tilde{\mathbf{w}}_b^{t+1} = \tilde{Q}_S(\mathbf{w}_b^{t+1}, \mathbf{\Delta}_b^{t+1})$;
Output: $\tilde{\mathbf{w}}_b^{t+1}, \mathbf{\Delta}_b^{t+1}, \mathbf{w}_o^{t+1}$.
```

as LSQ. In this way, we can obtain gradient for the step size. After two steps of optimization, both the model parameters and the step size are optimized. Then, we quantize $\mathbf{w}_b^{t+1}$ back into integers with $\mathbf{\Delta}_b^{t+1}$. Inspired by LSQ, to ensure convergence, we scale the gradient of the step size and adjust its learning rate to achieve optimal performance. Specifically, we set the scaling factor $g = 1/\sqrt{bdq}$, where b is the batch size, d is the embedding dimension and $q = 2^{m-1} - 1$.

4 Experiments

4.1 Evaluation Protocol

Dataset In this section, we conduct experiments on two real-world datasets: Criteo [3] and Avazu [4].

- The Criteo dataset consists of 26 categorical feature fields and 13 numerical feature fields. We discretize each numeric value x to $\lfloor \log^2(x) \rfloor$, if $x>2$; $x = 1$ otherwise. For categorical features, we replace the features that appear less than 10 times with a default "OOV" token.
- The Avazu dataset consists of 23 categorical feature fields. We transform the timestamp field into three new fields: hour, weekday, and is_weekend. Further, we replace the categorical features that appear less than twice with a default "OOV" token.

For both datasets, we split them in a ratio of 8:1:1 randomly to get corresponding training, validation, and test sets. Note that the pre-processing rules are similar to (Zhu et al. 2021).

Settings As presented in (Zhu et al. 2021), the performance of different deep CTR models are similar to each other, therefore we choose DCN (Wang et al. 2017), which is widely used, as the backbone model. The embedding dimension is primarily set to 16 in our experiments. We measure the performance of the ALPT and the baselines in terms of AUC and Logloss, which are two commonly-used metrics for CTR prediction. Note that an increase of 0.001 in AUC is generally considered as a significant improvement for CTR prediction asks (Cheng et al. 2016).

[3]https://www.kaggle.com/c/criteo-display-adchallenge

[4]https://www.kaggle.com/c/avazu-ctr-prediction

To ensure that the compared baselines are sufficiently tuned, we refer to the open benchmark for CTR prediction (Zhu et al. 2021) to set up our experiments. We use Adam (Kingma and Ba 2015) as the optimizer. The learning rate is set to 0.001 and the maximum number of epochs is 15. We reduce the learning rate tenfold after the 6th and 9th epochs. For regularization, we set the weight decay of embeddings to $5e-8$ and $1e-5$ for Avazu and Criteo, respectively. In addition, we adpot a dropout of 0.2 on MLP for Criteo. For the step size of ALPT, we set its learning rate to $2e-5$ and adopt the same weight decay and learning rate decay as the embeddings. All the experiments are run on a single Tesla V100 GPU with Intel Xeon Gold-6154 CPUs. Each experiment is performed at least five times.

Baselines To show the superiority of ALPT, we set four kinds of baselines:

- **FP:** Full-precision training for embeddings.
- **Hashing** and **Pruning** for embeddings. We implement the hashing method in (Shi et al. 2020) and the embedding pruning method in (Deng et al. 2021) according to the instructions in the corresponding papers. (Shi et al. 2020) uses the quotient ($\mathtt{id}/r$) and remainder ($\mathtt{id}\%r$) to index two embeddings, where $\mathtt{id}$ is the feature id and r is the compression ratio. The two embeddings will be multiplied as the final embedding. (Deng et al. 2021) prunes and retrains the embeddings, where the pruning ratio gradually increases.
- QAT for embeddings. We implement two of the SOTA methods: **PACT** (Choi et al. 2018) and **LSQ** (Esser et al. 2020). LSQ learns the step size by gradient descent. Similarly, PACT is to learn the clipping value (i.e., the range of quantized weights). Note that we use DR in LSQ and PACT since DR is the common choice in QAT.
- **LPT** for embeddings: We implement the vanilla LPT in (Xu et al. 2021). Following (Xu et al. 2021), we tune the clipping value among [1, 0.1, 0.01, 0.001].

4.2 Overall Performance

In this section, we will compare the performance of ALPT and the baselines. Also, we will verify our theoretical analysis about using SR or DR in LPT and ALPT. Note that the bit width of quantization is set to 8 and the compression ratio of hashing and pruning is set to $2\times$.

As Table 1 shows, ALPT achieves lossless compression and obtains the best accuracy. In contrast, the accuracy degradation of LPT, hashing or pruning is severe and unacceptable. Compared to LPT, ALPT has significant improvement on the prediction accuracy. Compared to the hashing and pruning methods, ALPT enables higher compression ratio and better accuracy at the same time. Although LSQ and PACT achieve comparable prediction accuracy, ALPT outperforms them at the training memory usage. Note that we also count the storage of the step size into the embedding size which slightly weakens the compression capability of ALPT. However, as the embedding dimension increases, the effect of the step size on the compression ratio is negligible.

As for the efficiency, we report the training time and inference time of various methods. Specifically, we record the

	Avazu			Criteo			Compression ratio	
	AUC	Logloss	Epochs × Time	AUC	Logloss	Epochs × Time	Training	Inference
FP	0.7949(±2e-4)	0.37069(±1e-4)	1 × 6min	0.8144(±5e-5)	0.43764(±5e-5)	9 × 9min	1x	1x
Hashing	0.7928(±3e-4)	0.37203(±2e-4)	1 × 6min	0.8119(±7e-5)	0.44130(±6e-5)	11 × 9min	2x	2x
Pruning	0.7926(±2e-4)	0.37202(±1e-4)	1 × 27min	0.8123(±5e-5)	0.44049(±5e-5)	8 × 16min	1x	2x
PACT	0.7948(±2e-4)	0.37074(±1e-4)	1 × 6min	0.8144(±8e-5)	0.43765(±6e-5)	9 × 9min	1x	4x
LSQ	0.7949(±1e-4)	0.37073(±1e-4)	1 × 6min	0.8144(±3e-5)	0.43764(±3e-5)	9 × 9min	1x	4x
LPT(DR)	0.7654(±4e-4)	0.38844(±3e-4)	15 × 6min	0.7966(±1e-4)	0.45306(±1e-4)	15 × 9min	4x	4x
LPT(SR)	0.7927(±2e-4)	0.37205(±1e-4)	1 × 6min	0.8123(±8e-5)	0.43945(±3e-5)	9 × 9min	4x	4x
ALPT(DR)	0.7928(±3e-4)	0.37216(±2e-4)	1 × 7min	0.8096(±2e-4)	0.44198(±2e-4)	6 × 11min	3.2x	3.2x
ALPT(SR)	**0.7951(±2e-4)**	**0.37062(±1e-4)**	1 × 7min	**0.8144(±3e-5)**	**0.43763(±3e-5)**	9 × 11min	3.2x	3.2x

Table 1: Performance of ALPT and the baselines on Criteo and Avazu.

	Avazu				Criteo			
	2-bit		4-bit		2-bit		4-bit	
	AUC	Logloss	AUC	Logloss	AUC	Logloss	AUC	Logloss
PACT	0.7678	0.38604	0.7925	0.37211	0.7900	0.45945	**0.8114**	**0.44030**
LSQ	**0.7912**	**0.37317**	**0.7940**	**0.37129**	**0.8089**	**0.44296**	0.8105	0.44278
LPT(SR)	0.7829	0.37806	0.7906	0.37342	0.8009	0.44973	0.8079	0.44379
ALPT(SR)	0.7851	0.37674	0.7919	0.37260	0.8033	0.44760	0.8111	0.44072

Table 2: Accuracy of different quantization methods with smaller bit widths.

	Avazu				Criteo			
	d=32		threshold=1		d=32		threshold=2	
	AUC	Logloss	AUC	Logloss	AUC	Logloss	AUC	Logloss
FP	0.7951	0.37055	0.7945	0.37099	0.8123	0.44039	0.8125	0.43975
LPT(SR)	0.7934	0.37162	0.7923	0.37234	0.8119	0.44013	0.8115	0.44036
ALPT(SR)	**0.7955**	**0.37040**	**0.7946**	**0.37098**	**0.8126**	**0.44000**	**0.8126**	**0.43948**

Table 3: Accuracy with larger embedding dimension and more categorical features.

average training time of each epoch on the training set and the average inference time of each step on the validation set. The inference time of different methods is similar (i.e., about 150 ms for each step). The training time of most methods is similar and ALPT only takes an extra minute on Avazu and two extra minutes on Criteo to learn the step size.

Additionally, as shown in Table 1, SR always achieves better performance than DR. Specifically, in LPT, SR has a better accuracy and can converge in fewer epochs, which is consistent with our analysis in Section 3.1. While ALPT can improve the convergence rate of DR by learning the step size, however, the convergence accuracy is still far from SR.

4.3 Scalability

In this section, to further validate the effectiveness of ALPT, we study the scalability of ALPT from three aspects, that is the bit width of quantization, the embedding dimension and the number of categorical features, respectively.

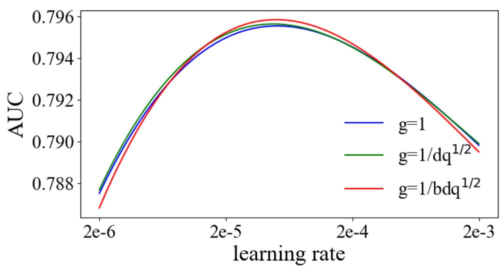
Smaller Bit Widths For the quantization methods in Table 1, we set the bit width to 4 and 2, respectively. Note that we have tuned the clipping value among [1, 0.1, 0.01, 0.001] for LPT and set the clipping value to 0.1 for 2-bit and 4-bit quantization. In ALPT, we adopt smaller weight decay for the step size (i.e., 0 for Avazu and 1e-6 for Criteo). Since lower bit width has a smaller representation range, the corresponding step size should be larger. As shown in Table 2, with different bit widths, the performance of ALPT is also consistently higher than that of LPT, especially in 2-bit. However, the accuracy gap between ALPT and LSQ becomes larger as the bit width decreases.

Larger Embedding Dimension In the above experiments, the embedding dimension is set to 16. Here, we increase the embedding dimension to 32 and set the bit width to 8. As shown in Table 3, with d=32, ALPT significantly improves the accuracy of LPT and even slightly surpasses the full-precision embeddings.

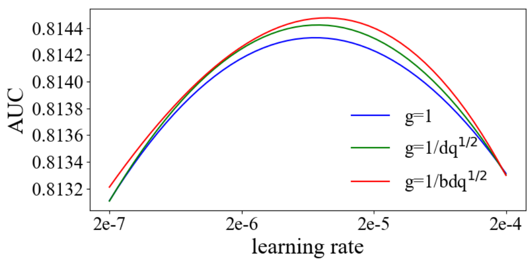
More Categorical Features In Section 4.1, we replace the features that appear less than twice in Avazu or 10 times in Criteo with a default ”OOV” token, which determines that Avazu has 4428293 features and Criteo has 1086895 features. In this section, to obtain datasets with more categorical features, we decrease the threshold, that is from 2 to 1 for Avazu and from 10 to 2 for Criteo. In this way, Criteo has 6780382 features and Avazu has 9449238 features. We conduct experiments with the regenerated datasets to validate the scalability of ALPT on larger datasets. As shown in Table 3, ALPT has always achieved lossless compression, which is a comprehensive proof of its good scalability.

4.4 Hyper-Parameters

Inspired by LSQ, we scale the gradient of the step size. The scaling factor g is tuned among $[1, 1/\sqrt{dq}, 1/\sqrt{bdq}]$. However, we find that the gradient scaling has little influence on accuracy, instead the learning rate has a significant effect. As Figure 4 shows, different scaling factors has similar accuracy given the learning rate. In our analysis, each step size



(a) Avazu



(b) Criteo

Figure 4: AUC under different learning rates and gradient scaling factors of the step size.

is only responsible for representing the weights in the corresponding embedding, which is easy to fit, so gradient scaling has little effect. In contrast, the weight decay on the step size is more sensitive to the learning rate, which makes the learning rate have a significant impact on the final performance.

5 Related Work

In this paper, a novel adaptive low-precision training paradigm is proposed to compress the embedding tables from the training stage. The most related domains are embedding compression and quantization. In this section, we discuss related work in these two domains.

5.1 Embedding Compression

As we mentioned in Section 1, most research on embedding compression in recommender systems focuses on three aspects: NAS-based approaches, embedding pruning and hashing. Specifically, NAS-based approaches search a proper dimension for each feature to save memory and improve the prediction accuracy. For example, (Joglekar et al. 2020) uses a reinforcement learning algorithm to search the optimal embedding dimension for users and items, while (Zhao et al. 2020) adopts differential architecture search (DARTS) algorithm (Liu, Simonyan, and Yang 2019) to learn the embedding dimension for each feature field. Considering that the search processes of these methods are quite time-consuming, recent works (Chen et al. 2021; Lyu et al. 2022) search for the optimal embedding dimension with well-designed search strategies. Similarly, parameter pruning reduce the number of weights in the embedding tables with an unstructured manner. (Deng et al. 2021) prunes and retrains the embedding table alternatively so that the mistakenly pruned weights can grow back. (Liu et al. 2021) maintains a learnable threshold to prune the embedding weights. The learnable threshold will be updated together with other parameters. Different from them, (Shi et al. 2020) converts an embedding table into two smaller matrices by two hash functions. Further, (Zhang et al. 2020) combine frequency hashing with double hashing for better accuracy. Also, (Yin et al. 2021; Wang et al. 2020; Xia et al. 2022) adopt tensor train decomposition to compress the embedding tables. (Su et al. 2021) reduces the number of embeddings by detecting and using only beneficial interactions.

5.2 Quantization

The deep learning community has extensive research and applications on quantization, which can be further divided into two sub-categories, that is quantizing a pre-trained model (Banner, Nahshan, and Soudry 2019; Guan et al. 2019; Nagel et al. 2020) or training a quantized model from scratch (Esser et al. 2020; Hou, Yao, and Kwok 2017). In the second category, most work follows the quantization-aware training paradigm (Courbariaux, Bengio, and David 2015; Rastegari et al. 2016), which quantizes weights in the forward pass and updates the full-precision weights with the gradients of the quantized weights. Recent works have also studied the loss-aware quantization (Hou and Kwok 2018; Hou, Zhang, and J. 2019) which explicitly considers the effect of quantization on the loss function.

In addition to different quantization paradigms, much work has explored the key factors of quantization, such as the step size and the rounding function. For example, (Esser et al. 2020; Choi et al. 2018) learn the step size and the clipping value by gradient descent, (Hou and Kwok 2018) approximates the optimal clipping values by second-order optimization. (Nagel et al. 2020) studies the rounding function and propose adaptive rounding. Similarly, other work also explored the strength of stochastic rounding. (Courbariaux, Bengio, and David 2015) studies stochastic weight binarization, (Lin et al. 2016) considers studies stochastic weight ternarization. (Hou, Zhang, and J. 2019; Chmiel et al. 2021) consider stochastic gradient quantization. When we ignore the error caused by the clipping function, stochastic weight quantization is equivalent to adding a zero-mean error term (i.e. Gaussian noise) to the weights, which can also be seen as a form of regularization.

6 Conclusion

In this paper, we formulate the low-precision training paradigm to compress embedding tables from the training stage. We provide theoretical analysis on its convergence with stochastic and deterministic weight quantization, which shows that stochastic weight quantization is more suitable for LPT. To reduce accuracy degradation, we further propose adaptive low-precision training (ALPT) to learn the step size and embeddings alternately. We conduct experiments on two real-world datasets which confirm our analysis while validating the superiority and scalability of ALPT.

Acknowledgments

This work is supported in part by National Natural Science Foundation of China under grants U1836204, U1936108, 62206102, and Science and Technology Support Program of Hubei Province under grant 2022BAA046.

References

Banner, R.; Nahshan, Y.; and Soudry, D. 2019. Post training 4-bit quantization of convolutional networks for rapid-deployment. In *Annual Conference on Neural Information Processing Systems 2019*, 7948–7956.

Chen, T.; Yin, H.; Zheng, Y.; et al. 2021. Learning Elastic Embeddings for Customizing On-Device Recommenders. In *The 27th ACM SIGKDD Conference on Knowledge Discovery and Data Mining*, 138–147. ACM.

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

Chmiel, B.; Banner, R.; Hoffer, E.; Ben-Yaacov, H.; and Soudry, D. 2021. Logarithmic Unbiased Quantization: Practical 4-bit Training in Deep Learning. *CoRR*, abs/2112.10769.

Choi, J.; Wang, Z.; Venkataramani, S.; Chuang, P. I.; Srinivasan, V.; and Gopalakrishnan, K. 2018. PACT: Parameterized Clipping Activation for Quantized Neural Networks. *CoRR*, abs/1805.06085.

Courbariaux, M.; Bengio, Y.; and David, J. 2015. BinaryConnect: Training Deep Neural Networks with binary weights during propagations. In *Advances in Neural Information Processing Systems*, 3123–3131.

Deng, W.; Pan, J.; Zhou, T.; Kong, D.; Flores, A.; and Lin, G. 2021. DeepLight: Deep Lightweight Feature Interactions for Accelerating CTR Predictions in Ad Serving. In *WSDM '21, The Fourteenth ACM International Conference on Web Search and Data Mining*, 922–930. ACM.

Esser, S. K.; McKinstry, J. L.; Bablani, D.; Appuswamy, R.; and Modha, D. S. 2020. Learned Step Size quantization. In *8th International Conference on Learning Representations, ICLR 2020,*. OpenReview.net.

Guan, H.; Malevich, A.; Yang, J.; Park, J.; and Yuen, H. 2019. Post-Training 4-bit Quantization on Embedding Tables. *CoRR*, abs/1911.02079.

Guo, H.; Chen, B.; Tang, R.; Zhang, W.; Li, Z.; and He, X. 2021a. An Embedding Learning Framework for Numerical Features in CTR Prediction. In *KDD '21: The 27th ACM SIGKDD Conference on Knowledge Discovery and Data Mining*, 2910–2918. ACM.

Guo, H.; Guo, W.; Gao, Y.; Tang, R.; He, X.; and Liu, W. 2021b. ScaleFreeCTR: MixCache-based Distributed Training System for CTR Models with Huge Embedding Table. In *SIGIR '21: The 44th International ACM SIGIR Conference on Research and Development in Information Retrieval*, 1269–1278. ACM.

Guo, H.; Tang, R.; Ye, Y.; Li, Z.; and He, X. 2017. DeepFM: A Factorization-Machine Based Neural Network for CTR Prediction. In *Proceedings of the Twenty-Sixth International Joint Conference on Artificial Intelligence, IJCAI 2017*, 1725–1731. AAAI Press.

Hou, L.; and Kwok, J. T. 2018. Loss-aware Weight Quantization of Deep Networks. In *International Conference on Learning Representations*.

Hou, L.; Yao, Q.; and Kwok, J. T. 2017. Loss-aware Binarization of Deep Networks. In *International Conference on Learning Representations*.

Hou, L.; Zhang, R.; and J., K. T. 2019. Analysis of Quantized Models. In *International Conference on Learning Representations*.

Joglekar, M. R.; Li, C.; Chen, M.; Xu, T.; Wang, X.; Adams, J. K.; Khaitan, P.; Liu, J.; and Le, Q. V. 2020. Neural Input Search for Large Scale Recommendation Models. In *KDD '20: The 26th ACM SIGKDD Conference on Knowledge Discovery and Data Mining*, 2387–2397. ACM.

Juan, Y.; Zhuang, Y.; Chin, W.; and Lin, C. 2016. Field-aware Factorization Machines for CTR Prediction. In *Proceedings of the 10th ACM Conference on Recommender Systems, Boston, MA, USA, September 15-19, 2016*, 43–50. ACM.

Kingma, D.; and Ba, J. 2015. Adam: A method for stochastic optimization. In *International Conference on Learning Representations*.

Li, H.; De, S.; Xu, Z.; Studer, C.; Samet, H.; and T., G. 2017. Training Quantized Nets: A Deeper Understanding. In *Advances in Neural Information Processing Systems*.

Lin, Z.; Courbariaux, M.; Memisevic, R.; and Bengio, Y. 2016. Neural Networks with Few Multiplications. In Bengio, Y.; and LeCun, Y., eds., *4th International Conference on Learning Representations, ICLR*.

Liu, H.; Simonyan, K.; and Yang, Y. 2019. DARTS: Differentiable Architecture Search. In *7th International Conference on Learning Representations, ICLR 2019, New Orleans, LA, USA, May 6-9, 2019*. OpenReview.net.

Liu, S.; Gao, C.; Chen, Y.; Jin, D.; and Li, Y. 2021. Learnable Embedding sizes for Recommender Systems. In *9th International Conference on Learning Representations, ICLR 2021, Virtual Event, Austria, May 3-7, 2021*. OpenReview.net.

Lyu, F.; Tang, X.; Zhu, H.; Guo, H.; Zhang, Y.; Tang, R.; and Liu, X. 2022. OptEmbed: Learning Optimal Embedding Table for Click-through Rate Prediction. In *Proceedings of the 31st ACM International Conference on Information & Knowledge Management*, 1399–1409. ACM.

McMahan, H. B.; Holt, G.; Sculley, D.; Young, M.; Ebner, D.; Grady, J.; Nie, L.; Phillips, T.; Davydov, E.; Golovin, D.; Chikkerur, S.; Liu, D.; Wattenberg, M.; Hrafnkelsson, A. M.; Boulos, T.; and Kubica, J. 2013. Ad click prediction: a view from the trenches. In *The 19th ACM SIGKDD International Conference on Knowledge Discovery and Data Mining, KDD 2013, Chicago, IL, USA, August 11-14, 2013*, 1222–1230. ACM.

Nagel, M.; Amjad, R. A.; van Baalen, M.; Louizos, C.; and Blankevoort, T. 2020. Up or Down? Adaptive Rounding for Post-Training Quantization. In *Proceedings of the 37th International Conference on Machine Learning, ICML 2020*, volume 119, 7197–7206. PMLR.

Rastegari, M.; Ordonez, V.; Redmon, J.; and Farhadi, A. 2016. XNOR-Net: ImageNet Classification Using Binary Convolutional Neural Networks. In *European Conference on Computer Vision*.

Shi, H. M.; Mudigere, D.; Naumov, M.; and Yang, J. 2020. Compositional Embeddings Using Complementary Partitions for Memory-Efficient Recommendation Systems. In *KDD '20: The 26th ACM SIGKDD Conference on Knowledge Discovery and Data Mining, Virtual Event, CA, USA, August 23-27, 2020*, 165–175. ACM.

Su, Y.; Zhang, R.; Erfani, S. M.; and Xu, Z. 2021. Detecting Beneficial Feature Interactions for Recommender Systems. In *35th AAAI Conference on Artificial Intelligence, AAAI 2021*, 4357–4365. AAAI Press.

Wang, Q.; Yin, H.; Chen, T.; Huang, Z.; Wang, H.; Zhao, Y.; and Hung, N. Q. V. 2020. Next Point-of-Interest Recommendation on Resource-Constrained Mobile Devices. In *Proceedings of Machine Learning and Systems 2021, MLSys 2021*, 906–916. ACM / IW3C2.

Wang, R.; Fu, B.; Fu, G.; and Wang, M. 2017. Deep & Cross Network for Ad Click Predictions. In *Proceedings of the ADKDD'17*, 12:1–12:7. ACM.

Xia, X.; Yin, H.; Yu, J.; Wang, Q.; Xu, G.; and Nguyen, Q. V. H. 2022. On-Device Next-Item Recommendation with Self-Supervised Knowledge Distillation. In *The 45th International ACM SIGIR Conference on Research and Development in Information Retrieval*, 546–555. ACM.

Xu, Z.; Li, D.; Zhao, W.; Shen, X.; Huang, T.; Li, X.; and Li, P. 2021. Agile and Accurate CTR Prediction Model Training for Massive-Scale Online Advertising Systems. In *SIGMOD '21: International Conference on Management of Data, Virtual Event, China, June 20-25, 2021*, 2404–2409. ACM.

Yang, J. A.; Huang, J.; Park, J.; Tang, P. T. P.; and Tulloch, A. 2020. Mixed-Precision Embedding Using a Cache. *CoRR*, abs/2010.11305.

Yin, C.; Acun, B.; Wu, C.; and Liu, X. 2021. TT-Rec: Tensor Train Compression for Deep Learning Recommendation Models. In *Proceedings of Machine Learning and Systems 2021, MLSys 2021*. mlsys.org.

Zhang, C.; Liu, Y.; Xie, Y.; Ktena, S. I.; Tejani, A.; Gupta, A.; Myana, P. K.; Dilipkumar, D.; Paul, S.; Ihara, I.; Upadhyaya, P.; Huszar, F.; and Shi, W. 2020. Model Size Reduction Using Frequency Based Double Hashing for Recommender Systems. In *RecSys 2020: Fourteenth ACM Conference on Recommender Systems, Virtual Event, Brazil, September 22-26, 2020*, 521–526. ACM.

Zhao, X.; Liu, H.; Liu, H.; Tang, J.; Guo, W.; Shi, J.; Wang, S.; Gao, H.; and Long, B. 2020. Memory-efficient Embedding for Recommendations. *CoRR*, abs/2006.14827.

Zhou, G.; Zhu, X.; Song, C.; Fan, Y.; Zhu, H.; Ma, X.; Yan, Y.; Jin, J.; Li, H.; and Gai, K. 2018. Deep Interest Network for Click-Through Rate Prediction. In *Proceedings of the 24th ACM SIGKDD International Conference on Knowledge Discovery & Data Mining*, 1059–1068.

Zhu, J.; Liu, J.; Yang, S.; Zhang, Q.; and He, X. 2021. Open Benchmarking for Click-Through Rate Prediction. In *International Conference on Information and Knowledge Management*, 2759–2769. ACM.