# CowClip: Reducing CTR Prediction Model Training Time from 12 Hours to 10 Minutes on 1 GPU

**Zangwei Zheng[1]*, Pengtai Xu[1]*, Xuan Zou[2], Da Tang[2], Zhen Li[2], Chenguang Xi[2], Peng Wu[2], Leqi Zou[2], Yijie Zhu[2], Ming Chen[2], Xiangzhuo Ding[2], Fuzhao Xue[1], Ziheng Qin[1], Youlong Cheng[2], Yang You[1]†**

[1] Department of Computer Science, National University of Singapore
[2] Bytedance Inc.
{zangwei, pengtai, f-xue, zihengq, youy}@comp.nus.edu.sg
{zouxuan, da.tang, zhen.li1, chenguang.xi, peng.wu, leqi.zou, yijie.zhu, ming.chen, xiangzhuo.ding, youlong.cheng}@bytedance.com

## Abstract

The click-through rate (CTR) prediction task is to predict whether a user will click on the recommended item. As mind-boggling amounts of data are produced online daily, accelerating CTR prediction model training is critical to ensuring an up-to-date model and reducing the training cost. One approach to increase the training speed is to apply large batch training. However, as shown in computer vision and natural language processing tasks, training with a large batch easily suffers from the loss of accuracy. Our experiments show that previous scaling rules fail in the training of CTR prediction neural networks. To tackle this problem, we first theoretically show that different frequencies of ids make it challenging to scale hyperparameters when scaling the batch size. To stabilize the training process in a large batch size setting, we develop the adaptive Column-wise Clipping (CowClip). It enables an easy and effective scaling rule for the embeddings, which keeps the learning rate unchanged and scales the L2 loss. We conduct extensive experiments with four CTR prediction networks on two real-world datasets and successfully scaled 128 times the original batch size without accuracy loss. In particular, for CTR prediction model DeepFM training on the Criteo dataset, our optimization framework enlarges the batch size from 1K to 128K with over 0.1% AUC improvement and reduces training time from 12 hours to 10 minutes on a single V100 GPU. Our code locates at github.com/bytedance/LargeBatchCTR.

## Introduction

With the development of the Internet and the e-economy, numerous clicking happens in online shopping (Ma et al. 2020; Zhou et al. 2019), video apps (Gomez-Uribe and Hunt 2016; Xie et al. 2020) and web advertisements (Covington, Adams, and Sargin 2016; Zhao et al. 2019). Click-through Rate (CTR) prediction is to predict whether a user will click on the recommended item. It is a fundamental task in advertising and recommendation systems. An accurate CTR prediction can directly improve user experience (Kaasinen et al. 2009) and enhance ads profit (Wang 2020).

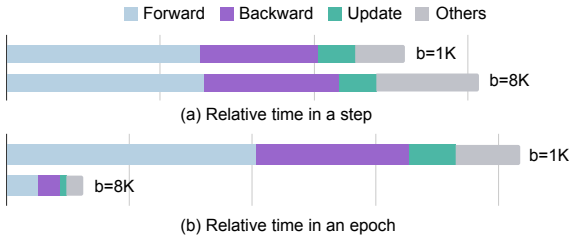


Figure 1: Relative time of training DeepFM model on Criteo dataset with one V100 GPU.

In a typical industrial dataset, the number of click samples has grown up to hundreds of billion (Zhao et al. 2019; Xie et al. 2020) and keeps increasing on a daily basis. The click-through rate (CTR) prediction task is to predict whether a user will click on the recommended item. It is a fundamental task in advertising and recommendation systems. Since CTR prediction is a time-sensitive task (Zhao et al. 2019) (*e.g.*, latest topics, hottest videos, and new users' hobbies), it is necessary to shorten the time needed for re-training on a massive dataset to maintain an up-to-date CTR prediction model. In addition, given a constant computing budget, decreasing the training time also reduces the training cost, giving rise to a high return-to-investment ratio.

Recent years have witnessed rapid growth in GPU processing ability (Baji 2018). With the growth of GPU memory and FLOPS, a larger batch size can take better advantage of the parallel processing capability of GPUs. As shown in Figure 1 (a), the time of one forward and backward pass is almost the same when scaling 8 times batch size, indicating GPU with a small batch size is extremely underused. Since the number of training epochs remains the same, large batch training reduces the number of steps and thus significantly shortens the total training time (Figure 1 (b)). In ad-

*Work done during an internship at Bytedance.
†Yang You is the corresponding author.

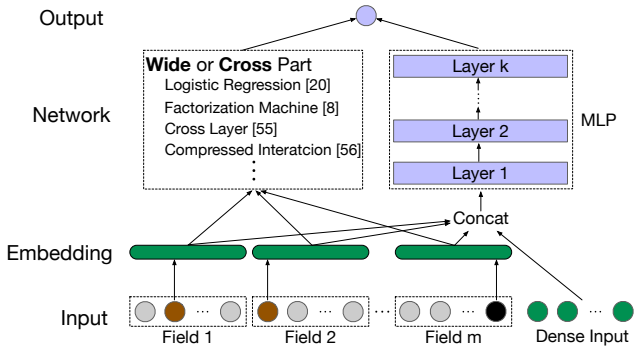


Figure 2: A simple illustration of a Wide/Cross-and-Deep style of CTR prediction model. The green data denotes a dense one, while brown input in a categorical field stands for a selected id.

| | Network | | Embedding (Dataset) | |
|---|---|---|---|---|
| **Name** | DeepFM | DCN | Criteo | Avazu |
| **#Params** | 0.431M | 0.655M | 372M | 104M |

Table 1: Number of parameters for different layers.

dition, a large batch benefits more in a multi-GPUs setting, where gradients of the large embedding layer need to be exchanged between different GPUs and machines, resulting in high communication costs. To avoid distraction from system optimization in reducing communication costs (Mudigere et al. 2021; Zhao et al. 2019; Xie et al. 2020), we focus on designing an accuracy-preserving algorithm for scaling batch size on a single GPU, which can be easily extended for multi-node training. The challenge of applying large batch training is an accuracy loss when naively increasing the batch size (He et al. 2021), especially considering that CTR prediction is a very sensitive task and cannot bear the accuracy loss. Hyperparameter scaling rules (Krizhevsky 2014; Goyal et al. 2017) and carefully designed optimization methods (You, Gitman, and Ginsburg 2017; You et al. 2020) in CV and NLP tasks are not directly suitable for CTR prediction. This is because, in CTR prediction, the inputs are more sparse and frequency-unbalanced, and the embedding layers dominate the parameters of the whole network (*e.g.*, 99.9%, see Table 1). In this paper, we identified the failure reason behind previous scaling rules on CTR prediction and proposed an effective algorithm and scaling rule for large batch training.

In conclusion, our contributions are as follows:

- To the best of our knowledge, we are the first to investigate the stability of the training CTR prediction model in very large batch sizes. We attribute the hardship in scaling the batch size to the difference in id frequencies.
- With rigorous mathematical analysis, we prove that the learning rate for infrequent features should not be scaled when scaling up the batch size. With CowClip, we can adopt an easy and effective scaling strategy for scaling up the batch size.

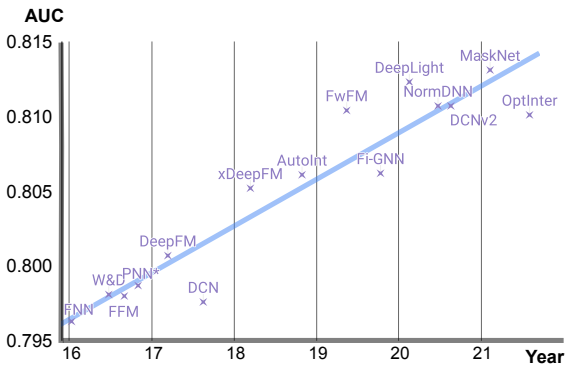


Figure 3: Progress on AUC of CTR prediction models on Criteo dataset in the past six years.

- We propose an effective optimization method of adaptive Column-wise Clipping (CowClip) to stabilize the training process of the CTR prediction task. We successfully scale up 128 times batch size for four models on two public datasets. In particular, we train the DeepFM model with 72 times speedup and 0.1% AUC improvement on the Criteo dataset.

## Related Work

**Embeddings of CTR Prediction Model.** The input of the CTR prediction model is high-dimensional, sparse, and frequent-unbalanced. As we will discuss the frequency in the Section 3, we focus on the fact that the input feature space for CTR prediction is high-dimensional and sparse, which is an essential difference between the CTR prediction model and other deep learning models.

A typical industrial CTR prediction model (Zhao et al. 2019; Xie et al. 2020; Zhou et al. 2019) has a high-dimensional input space with $10^8$ to $10^{12}$ dimensions after one-hot encoding of the categorical features. At the same time, a single clicking log may contain only hundreds of non-zero entries. As a result, when we create the embedding for each feature, the whole embedding layer can be extremely large, and the parameters of the CTR prediction model are dominated (*e.g.*, 99.9%) by the embedding part instead of the deep network part (Miao et al. 2021; Ginart et al. 2021). Table 1 shows the case under our experimental setting.

As the number of parameters in the embedding layer overwhelms the one of the dense networks, the difficulty of large batch optimization lies in the embedding layers. This paper focuses on addressing the training instability caused by the properties of embedding layers in the CTR prediction model. No matter how the dense part, *e.g.*, MLP, LSTM (Chen and Li 2021), Transformer (Chen et al. 2019), changes, the training instability caused by the embedding part still exists.

**CTR Prediction Network.** A thread of work started from (Cheng et al. 2016; Wang et al. 2017) occupies a majority of the above networks. They focused on designing a two-stream network, as shown in Figure 6. Following W&D

| | No Scale | Sqrt Scale | Linear Scale |
|---|---|---|---|
| | **Criteo** | | |
| 1k | 80.76 | 80.76 | 80.76 |
| 2k | –0.15 | –0.01 | –0.01 |
| 4k | –1.35 | –0.06 | –0.11 |
| 8k | –3.21 | –0.21 | –0.20 |
| | **Criteo (Top 3 frequent ids)** | | |
| 1k | 74.97 | 74.97 | 74.97 |
| 2k | –0.10 | –0.01 | +0.04 |
| 4k | –0.20 | –0.02 | –0.02 |
| 8k | –0.28 | –0.01 | –0.01 |

Table 2: AUC (%) changes at different batch sizes on Criteo with DeepFM and a modified version. Previous scaling rules fail on Criteo but work for a revised version.

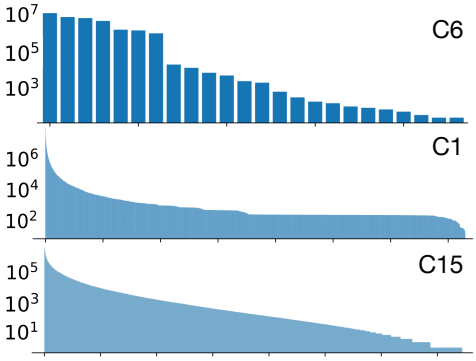


Figure 4: Distribution of different ids in three fields of the Criteo dataset. The y-axis is in logarithm scale. The total number of samples is $4.13 \times 10^7$.

model (Cheng et al. 2016), there are many designs on the wide/cross-stream. The details of DeepFM (Guo et al. 2018), W&D, DCN (Wang et al. 2017), and DCN-v2 (Wang et al. 2021) used in our experiments are presented in the Appendix.

**Large Batch Training Methods.** To preserve the performance of deep models at a large batch size, we need a good scaling rule and a stable optimization strategy. The scaling rule tells us how to scale the hyperparameters when scaling up the batch size. The two most important hyperparameters when scaling the batch size are learning rate and regularization weight. Based on different assumptions, linear scaling (Goyal et al. 2017) and square root scaling (Krizhevsky 2014; Hoffer, Hubara, and Soudry 2017) are the two most common scaling rules in the deep learning community. Besides, optimization strategies such as warmup (Gotmare et al. 2019) and gradient clipping (Zhang et al. 2020) can help stabilize the large batch training process. LARS (You, Gitman, and Ginsburg 2017) and LAMB (You et al. 2020) are two optimizers designed for large batch training, which adopt different adaptive learning rates for each layer. Although they achieve good results in CV and NLP tasks, they are ineffective in the CTR prediction task because it is unnecessary to use a layer-wise optimizer with a shallow network (*e.g.*, three or four layers). This paper re-designs the scaling rule and optimization strategy for the embedding layer, which can successfully scale up the batch size for CTR prediction.

Additional related work can be found in Appendix, including sensitiveness of CTR prediction and works utilizing different frequencies.

## Method

In CTR prediction, we have the training dataset $\mathcal{D} = \{\boldsymbol{x}_i, y_i\}_{i=1}^N$, where $y \in \{0, 1\}$ denotes whether the user clicked or not. The $\boldsymbol{x}$ contains information about the user, the product, and the interaction, which can be categorical or continuous. The categorical field is one-hot encoded to be a vector $\boldsymbol{x}_i^{\mathrm{f}_j}$ of $d_{\mathrm{f}_j}$ length, where $d_{\mathrm{f}_j}$ is the number of possible values (ids) in this field. To represent the frequency of each id, we denote the $k$-th id in field $j$ as $\mathrm{id}_k^{\mathrm{f}_j}$. The frequency and occurrence probability of the id is:

$$\texttt{count}(\mathrm{id}_k^{\mathrm{f}_j}) = \sum_{i=1}^{N} \delta(\boldsymbol{x}_i^{\mathrm{f}_j}[k] = 1),$$

$$\mathrm{P}(\mathrm{id}_k^{\mathrm{f}_j} \in \boldsymbol{x}) = \frac{\texttt{count}(\mathrm{id}_k^{\mathrm{f}_j})}{N},$$

where $\delta(\cdot)$ equals 1 if the boolean condition holds and 0 otherwise.

Given the predicting network $f$, the prediction is made from $f(\boldsymbol{x})$. The network and embeddings weights are denoted as $w$, and the training loss is $L$. This paper focuses on the Wide/Cross-and-Deep kind of CTR prediction model, as briefly described in Figure 2, one of the state-of-the-art networks in CTR prediction (Wang et al. 2021; Zhang, Huang, and Zhang 2019).

In training the network, we use a batch size of $b = |B|$, where $B$ is a specific batch. The learning rate and L2-regularization weight are denoted as $\eta$ and $\lambda$. The total number of steps in an epoch is $\frac{N}{b}$.

### Failure Cause of Traditional Scaling Rules

When training a neural network, at step $t$, an optimizer $\texttt{Opt}(\cdot)$ takes in the weights and gradients, and output the updated weights. With the L2-regularization, the update process can be formulated as:

$$\boldsymbol{g}_t = \sum_{x \in B_t} \nabla L(w, x) + \frac{\lambda}{2} \cdot \|w\|_2^2$$

$$w_{t+1} = \eta \cdot \texttt{Opt}(w_t, \boldsymbol{g}_t).$$

When changing the batch size, the hyper-parameter learning rate $\eta$ and L2-regularization weight $\lambda$ should be adjusted for maintaining the same performance as the original batch size.

Square root scaling (Krizhevsky 2014; Hoffer, Hubara, and Soudry 2017) and linear scaling (Goyal et al. 2017) are two widely used scaling rules in deep learning. The motivation for sqrt scaling is to keep the covariance matrix of

the parameters update the same, while for linear scaling, the motivation is to keep the update in a large batch equal to updates from $s$ small batches when scaling $s$ times the batch size (details in Appendix). The two scaling rules have been shown effective in CV and NLP tasks, and they are shown as follows:

**Scaling Rule 1 (Sqrt Scaling)** *When scaling batch size from $b$ to $s \cdot b$, do as follows:*

$$\eta \to \sqrt{s} \cdot \eta, \quad \lambda \to \sqrt{s} \cdot \lambda$$

**Scaling Rule 2 (Linear Scaling)** *When scaling batch size from $b$ to $s \cdot b$, do as follows:*

$$\eta \to s \cdot \eta, \quad \lambda \to \lambda$$

Our first attempt at large batch training of the CTR prediction model is to apply the above classic scaling rules: no scaling, linear scaling (Goyal et al. 2017), and square root scaling (Krizhevsky 2014). However, as seen in experiments on the Criteo dataset with DeepFM model in Table 2 left, the above rules fail in a CTR prediction model. We claim that the reason for the failure lies in the different frequencies of ids.

The product id of a popular item or ids in fields with a few options (*e.g.*, male and female in gender field) are frequent, while the id of an inactive user seldom appears. In Figure 4, we visualize the distribution of different ids' frequencies in three fields. The exponential distribution reveals different frequencies among different ids. For the dense weights (*e.g.*, kernel weights), their gradients appear for each sample while embedding does not have gradients if the corresponding ids do not show up. In CTR prediction, the embedding layers dominate the parameters of the whole network, and different occurrences of gradients make a great difference from other deep neural networks.

First, we empirically verify our claim by the following experiment. We keep the top three frequent ids in each field and label the rest as a fourth id. In this way, all four ids are very frequent and variations in frequencies are ablated in this modified version of Criteo. As shown in Table 2 right, both scaling rules successfully apply to the modified dataset, which means the traditional scaling rule does not work in CTR prediction due to the presence of infrequent ids.

Next, we provide the theoretical analysis for the failure of sqrt and linear scaling. Different frequencies lead to varying occurrences of ids in batches. Only when the id appears in the batch can the corresponding embedding be updated. They only occur in a small fraction of batches for ids with a low frequency. Suppose we draw the training samples with replacement from the dataset, the probability of an id $\mathrm{id}_k^{\mathrm{f}_j}$ in the batch $B$ is:

$$\mathrm{P}(\mathrm{id}_k^{\mathrm{f}_j} \in B) = 1 - (1 - \mathrm{P}(\mathrm{id}_k^{\mathrm{f}_j} \in \boldsymbol{x}))^b.$$

For frequent ids, and also dense weights whose frequency rate is 1, we have $(1 - \mathrm{P}(\mathrm{id}_k^{\mathrm{f}_j} \in \boldsymbol{x}))^b \approx 0$; while for the infrequent ids, when $p \ll \frac{1}{B}$, we can use binomial approximation and obtain:

$$\mathrm{P}(\mathrm{id}_k^{\mathrm{f}_j} \in B) \approx \begin{cases} 1 & \mathrm{id}_k^{\mathrm{f}_j} \text{ is frequent} \\ b \cdot \mathrm{P}(\mathrm{id}_k^{\mathrm{f}_j} \in \boldsymbol{x}) & \mathrm{id}_k^{\mathrm{f}_j} \text{ is infrequent} \end{cases}. \quad (1)$$

Now, reconsider the linear scaling motivation for an id's embedding $w$. Denote the weight update as $\Delta w = w_t - w_{t+1}$. Consider the expected update in a large batch $B' = \bigcup_{i=1}^{s} B_i$ with $b' = |B'| = s \cdot b$, we have

$$\begin{aligned} \mathbb{E}[\Delta w] &= \mathbb{E}[\eta' \cdot \delta(\mathrm{id}_k^{\mathrm{f}_j} \in B') \cdot \frac{1}{b'} \sum_{x \in B'} \nabla L(w, x)] \\ &= \eta' \cdot \mathrm{P}(\mathrm{id}_k^{\mathrm{f}_j} \in B') \cdot \mathbb{E}[\nabla L(w, x)]. \end{aligned}$$

With the assumption that $\mathbb{E}[\nabla L(w_i, x)] \approx \mathbb{E}[\nabla L(w, x)]$, the expected update in small batches $B_i$ is:

$$\begin{aligned} \mathbb{E}[\Delta w] &= \mathbb{E}[\eta \cdot \sum_{i=1}^{s} \delta(\mathrm{id}_k^{\mathrm{f}_j} \in B_i) \cdot \frac{1}{b} \sum_{x \in B_i} \nabla L(w_i, x)] \\ &\approx \eta \cdot s \cdot \mathrm{P}(\mathrm{id}_k^{\mathrm{f}_j} \in B) \cdot \mathbb{E}[\nabla L(w, x)]. \end{aligned}$$

For dense weight or embeddings of frequent id, the term $\mathrm{P}(\mathrm{id}_k^{\mathrm{f}_j} \in B)$ equals 1, making no difference to the original linear scaling rule. However, with an infrequent id, it shows that the new scaling strategy should be using the same learning rate when scaling the batch size due to the following fact for infrequent ids:

$$\mathrm{P}(\mathrm{id}_k^{\mathrm{f}_j} \in B') \approx s \cdot \mathrm{P}(\mathrm{id}_k^{\mathrm{f}_j} \in B).$$

A similar discussion based on sqrt scaling motivation (see Appendix) shows that under a very strong assumption can we obtain the same conclusion. However, without the assumption, we cannot even choose hyperparameters maintaining the same covariance matrix after scaling the batch size.

When using a relatively small batch size we find most ids satisfied $p < \frac{1}{B}$. Thus, we propose to use no scaling on the whole embedding layers, which suits infrequent ids. In addition, a smaller learning rate for layers at the bottom leads to a smooth learning process. Experiments show this scaling rule leads to a better result.

After the discussion of learning rate scaling, now let's turn to the L2-regularization weight $\lambda$. In CTR prediction, an unsuitable $\lambda$ can easily lead to overfitting. For the scaling of $\lambda$, we first consider the embedding vector $w$ of $\mathrm{id}_k^{\mathrm{f}_j}$, the expected gradient of which in a batch is:

$$\begin{aligned} \mathbb{E}[\boldsymbol{g}] &= \frac{1}{b} \mathbb{E}[\delta(\mathrm{id}_k^{\mathrm{f}_j} \in B) \sum_{x \in B} \nabla L(w, x)] \\ &= \mathrm{P}(\mathrm{id}_k^{\mathrm{f}_j} \in B) \cdot \mathbb{E}[\nabla L(w, x)]. \end{aligned} \quad (2)$$

The term $\mathrm{P}(\mathrm{id}_k^{\mathrm{f}_j} \in B)$ still has no effect with dense weight and embeddings of frequent ids as the probability equals to 1. However, for the infrequent ids, there is a scaling multiplier before the expectation of the gradient as some ids may not appear in a certain batch. When using an adaptive optimizer such as Adam, this scaling multiplier results in a different behaviour, which is equivalent to adjusting the L2-regularization weight $\lambda$ as follows (see Appendix for the proof):

$$\frac{\lambda}{\mathrm{P}(\mathrm{id}_k^{\mathrm{f}_j} \in B)} = \frac{\lambda}{b \cdot \mathrm{P}(\mathrm{id}_k^{\mathrm{f}_j} \in \boldsymbol{x})}.$$

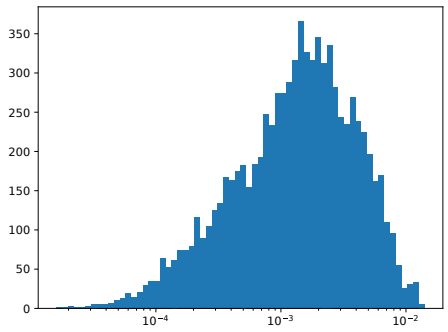


Figure 5: L2-norm distribution of different columns gradients at 1000th step of DeepFM on Criteo dataset. Only columns with existing ids in the batch are shown. The x-axis is the L2 norm value and y-axis is the count of columns.

Thus, to maintain the same L2-regularization strength, we scale up the $\lambda$ by $n$. Combined with the learning rate scaling rule, we have the following one.

**Scaling Rule 3 (CowClip Scaling)** *When scaling batch size from $b$ to $s \cdot b$, use sqrt scaling for the dense weights, and do as follows for embeddings:*

$$\eta_e \to \eta_e, \quad \lambda \to s \cdot \lambda$$

However, we find directly applying the above rule leads to overfitting due to $s$ times less application of L2-regularization when scaling up batch size by $s$ times. If no additional regularization technique is introduced, L2-regularization should be strengthened further with a large batch size. In the case of an SGD optimizer, we have (details in Appendix):

$$\eta' \lambda' \approx s\eta\lambda.$$

Hence, we need to further scale up the $\lambda$ by $s$ times when the learning rate is unchanged. Although the behavior of adaptive optimizers such as Adam is different from SGD, we find a larger $\lambda$ prevents overfitting. Thus, we have the following scaling rule which can scale up the batch size to 4K without additional optimization strategy.

**Scaling Rule 4 ($n^2$–$\lambda$ Scaling)** *When scaling batch size from $b$ to $s \cdot b$, use sqrt scaling for the dense weights, and do as follows for embeddings:*

$$\eta_e \to \eta_e, \quad \lambda_e \to s^2 \cdot \lambda_e$$

## CowClip Algorithm

Although the Scaling Rule 4 helps us scale to 4 times the original batch size, it fails on a larger batch size. The challenge in choosing the proper learning rate $\eta$ and L2-weight $\lambda$ mentioned above impairs the performance for larger batch sizes. To enable large batch training, the gradient norm clipping (Zhang et al. 2020) can smooth the process of training and alleviate the sensitiveness of hyperparameters. Given a clip threshold $\texttt{clip\_t}$, gradient norm clipping does follows:

$$\boldsymbol{g} \to \min\{1, \frac{\texttt{clip\_t}}{\|\boldsymbol{g}\|}\} \cdot \boldsymbol{g}$$

Gradient norm clipping smoothes the training process by reducing the norm of a large gradient greater than a threshold. However, it is hard to choose an appropriate threshold for clipping the norm. Besides, as one column of the embedding matrix represents the embedding vector for an id, Figure 5 shows that the magnitude of gradients for different columns varies. We denote the column for an id as $w[\mathrm{id}_k^{\mathrm{f}_j}]$. Clipping on the whole embeddings whose gradient norm is dominated by gradients of columns with large gradients impairs the ones with normal but smaller gradients. In addition, according to Equation (2), since we want to clip on $1 \cdot \nabla L(w, x)$, the different frequencies of ids lead to the scaler of $\mathrm{P}(\mathrm{id}_k^{\mathrm{f}_j} \in B)$ on the expected gradients.

To tackle the above problems, inspired by LAMB optimizer (You et al. 2020), which normalizes the norm of gradients of each kernel to be proportional to the norm of the kernel weight, we relate the clip threshold with the norm of id embedding vectors. The difference between our clipping method and the gradient norm clipping is three-fold: First, every id embedding vector has a unique clipping threshold for more flexible clipping. Second, the clipping threshold is multiplied by the occurrence number of the id to make sure the bound is based on $1 \cdot \nabla L(w, x)$. Last but not the least, the clipping threshold is calculated by the norm of the id vector in consideration of different magnitudes:

$$\texttt{clip}(\mathrm{id}_k^{\mathrm{f}_j}) = \mathrm{cnt}(\mathrm{id}_k^{\mathrm{f}_j}) \cdot \max\{r \cdot \|w_t^e[\mathrm{id}_k^{\mathrm{f}_j}]\|, \ \zeta\}$$

where $\texttt{cnt}(\mathrm{id}_k^{\mathrm{f}_j})$ is the number of occurence of the id in a batch.

As the weights grow larger in the training process, the benefit of a threshold proportional to the norm of the weight is that the clipping value adaptively grows with the network. As some infrequent id embedding vectors become too small due to the continual application of L2-regularization with no id occurrence in steps, we restrict the clipping norm by a lower-bound $\zeta$ to avoid a too strong clipping.

The network training with CowClip is summarized in the Algorithm 1. In practice, tensor multiplication instead of for-loop is adopted for less computational overhead. Since CowClip stabilizes the training process, it is possible to use the CowClip scaling 3 rule to 128× batch size, leaving $\eta_e$ unchanged and linear scaling the $\lambda$. We give a proof sketch on the convergence of CowClip method in Appendix. Our large batch training framework contains the CowClip gradient clipping and scaling strategy.

# Experiment

## Experimental Setting

**Datasets.** We evaluate our algorithms on the following public datasets which are widely adopted by the community (Cheng et al. 2016; Li et al. 2019; Deng et al. 2021; Wang et al. 2021; Miao et al. 2021). **Criteo** (Labs 2014) is a real-world CTR prediction dataset. It collects 45M records on ad display information, and the corresponding user clicks feedback. There are 13 continuous fields and 26 categorical fields, which are all anonymized to protect users' privacy. Following (Guo et al. 2018; Zhang, Huang,

| | 1K | | 8K | | 128K | |
|---|---|---|---|---|---|---|
| | Prev. best | CowClip | Prev. best | CowClip | Prev. best | Cowclip |
| Criteo | 80.76 | 80.86 | 80.55 | 80.97 | – | 80.90 |
| Criteo-seq | 80.48 | 80.50 | 80.03 | 80.50 | – | 80.49 |
| Avazu | 78.84 | 78.83 | 76.69 | 79.06 | – | 78.80 |

Table 3: Performance comparison between Cowclip and previous scaling methods with different batch size.

| | 1K (1024) | | 2K (2048) | | 4K (4096) | | 8K (8192) | |
|---|---|---|---|---|---|---|---|---|
| | AUC (%) | LogLoss | AUC (%) | LogLoss | AUC (%) | LogLoss | AUC (%) | LogLoss |
| No Scaling | 80.76 | 0.4438 | 80.66 | 0.4456 | 80.48 | 0.4518 | 80.31 | 0.4530 |
| Sqrt Scaling | 80.76 | 0.4438 | 80.71 | 0.4430 | 80.59 | 0.4450 | 80.28 | 0.4582 |
| Sqrt Scaling* | 80.76 | 0.4438 | 80.75 | 0.4444 | 80.69 | 0.4449 | 80.55 | 0.4547 |
| LR Scaling | 80.76 | 0.4438 | 80.77 | 0.4434 | 80.65 | 0.4434 | 80.46 | 0.4542 |
| $n^2$–$\lambda$ Scaling (Ours) | 80.76 | 0.4438 | 80.86 | 0.4432 | 80.90 | 0.4426 | 80.73 | 0.4441 |
| CowClip (Ours) | 80.86 | 0.4430 | 80.93 | 0.4427 | 80.97 | 0.4422 | 80.97 | 0.4425 |

Table 4: Performance of different scaling methods on Criteo dataset from 1K to 8K on DeepFM.

Algorithm 1: Adaptive Column-wise Clipping(CowClip)

**Input:** CowClip coefficient $r$ and lower-bound $\zeta$, number of steps $T$, batch size $b$, learning rate for dense and embedding $\eta, \eta_e$, optimizer $\texttt{Opt}(\cdot)$

**for** $t \leftarrow 1$ to $T$ **do**
  Draw $b$ samples $B$ from $\mathcal{D}$
  $\boldsymbol{g}_t, \boldsymbol{g}_t^e \leftarrow \frac{1}{b}\sum_{x\in B}\nabla L(x, w_t, w_t^e)$
  $w_{t+1} \leftarrow \eta \cdot \texttt{Opt}(w_t, \boldsymbol{g}_t)$ // Update dense weights
  **for** each field and each column in the field **do**
    $n_{\boldsymbol{g}} \leftarrow \|\boldsymbol{g}_t^e[\text{id}_k^{\text{f}_j}]\|$
    $\texttt{cnt} \leftarrow |\{x \in B | \text{id}_k^{\text{f}_j} \in x\}|$
    // Calculate the number of occurrence cnt
    $\texttt{clip\_t} \leftarrow \texttt{cnt} \cdot \max\{r \cdot \|w_t^e[\text{id}_k^{\text{f}_j}]\|,\ \zeta\}$
    // Clip norm threshold
    $\boldsymbol{g}_c \leftarrow \min\{1, \frac{\texttt{clip\_t}}{n_{\boldsymbol{g}}}\} \cdot \boldsymbol{g}_t^e[\text{id}_k^{\text{f}_j}]$
    // Gradient clipping
    $w_t^e[\text{id}_k^{\text{f}_j}] \leftarrow \eta_e \cdot \texttt{Opt}(w_t^e[\text{id}_k^{\text{f}_j}], \boldsymbol{g}_c)$
    // Update the id embedding
  **end for**
**end for**

and Zhang 2019), the data is split into training and test sets by 90%:10%. **Criteo-seq** is a sequential learning setting of Criteo dataset. The first six days' data are used for training and the last day's data for testing. This setting evaluates the performance of the algorithm in a sequential learning setting. **Avazu** (Avazu 2015) is another ad click-through dataset containing 32M training samples. It has 24 anonymous categorical fields. According to (Zhang, Huang, and Zhang 2019), we split the dataset into training and test sets by 80%:20%.

**Implementation Details.** We use two popular metrics (Mattson et al. 2020) in CTR prediction: AUC (Area Under ROC) and Logloss (Logistic loss). Our implementation is based on Tensorflow (Abadi et al. 2015) and DeepCTR (Shen 2017) framework. The experiments are conducted on one Tesla V100 GPU. We use Adam (Kingma and Ba 2015) optimizer and an L2-regularization on embedding layers. The base learning rate and L2-regularization weight on batch size 1024 are $10^{-4}$ and $10^{-5}$. Scaling rules are performed based on the 1024 batch size. For CowClip, we use $r = 1$ and tune $\zeta \in \{10^{-5}, 10^{-4}\}$ due to a different initialization weight norm. We also use learning rate warmup (Gotmare et al. 2019) and larger initialization weights. More discussion on hyperparameter choice and techniques can be found in the Appendix. We run our experiments with three random seeds, and the standard deviation among all experiments for AUC is less than 0.012%.

**Baselines.** Four CTR prediction models are considered in our experiments: Wide-and-Deep Network (**W&D**) (Cheng et al. 2016), **DeepFM** (Guo et al. 2018), Deep-and-Cross Network (**DCN**) (Wang et al. 2017), **DCN v2** (Wang et al. 2021). The architectures of these networks are detailed in Appendix). For the scaling strategy, **No Scaling** means we use the same hyper-parameters as the ones in batch size 1K. **Sqrt Scaling** and **LR Scaling** are described in Section 3. **Sqrt Scaling*** is a variant version of Sqrt Scaling used in (Guo et al. 2018), which does not scale up the L2-regularization. For batch size from 1K to 8K, we also do a grid search on learning rate and the weight decay, but it turns out no simple combination yields better results than the above scaling methods. **DLRM** (Naumov et al. 2019) uses model parallelism on the embedding table to accelerate the training. **XDL** (Adnan et al. 2021) is a highly optimized implementation of the above model. **FAE** (Adnan et al. 2021) takes the frequency of embeddings into consideration as well and uses a hot-embedding aware data layout in the memory. **Hotline** (Adnan 2021) better organizes the frequent and infrequent embeddings in the GPU and main memory. **CowClip** denotes training with the CowClip method and the CowClip scaling rule 3.

| | | Baseline | 1K | 2K | 4K | 8K | 16K | 32K | 64K | 128K |
|---|---|---|---|---|---|---|---|---|---|---|
| DeepFM (Guo et al. 2018) | AUC (%) | 80.76 | 80.86 | 80.93 | 80.97 | 80.97 | 80.94 | 80.95 | 80.96 | 80.90 |
| | Logloss | 0.4438 | 0.4430 | 0.4427 | 0.4422 | 0.4425 | 0.4424 | 0.4423 | 0.4429 | 0.4430 |
| W&D (Cheng et al. 2016) | AUC (%) | 80.75 | 80.86 | 80.94 | 80.96 | 80.96 | 80.95 | 80.94 | 80.96 | 80.89 |
| | Logloss | 0.4439 | 0.4430 | 0.4424 | 0.4422 | 0.4425 | 0.4422 | 0.4428 | 0.4429 | 0.4434 |
| DCN (Wang et al. 2017) | AUC (%) | 80.76 | 80.86 | 80.93 | 80.96 | 80.97 | 80.98 | 80.95 | 80.99 | 80.91 |
| | Logloss | 0.4438 | 0.4429 | 0.4424 | 0.4422 | 0.4428 | 0.4419 | 0.4426 | 0.4426 | 0.4428 |

Table 5: Performance of CowClip methods on Criteo dataset from 1K to 128K on four models.

## Large Batch Training Results

First, as shown in Table 3, previous scaling strategy fails to maintain the performance at batch size 8K, and fails to converge at batch size 128K. In contrast, CowClip methods can achieve a better accuracy at 8K batch size and almost no performance loss at batch size 128K, which shows the CowClip method successfully stablizes the training processes.

Then, we compare different scaling strategies on the DeepFM model. The results on Criteo dataset are presented in Table 4, and the results on Criteo-seq and Avazu are attached in Appendix. As we can see, traditional scaling rules fail to meet the AUC requirement with a large gap when the batch size grows up to 4K. This is consistent with results in (Guo et al. 2018), where results with 4K batch size are worse than those with 1K. With $n^2$–$\lambda$ Scaling rule 4, it can scale batch size to 4K but fails with 8K. When we successfully scale the batch size, there is a performance gain in the AUC, which is also observed in (Zhu et al. 2021). For our CowClip algorithm, it outperforms the original optimization method by $0.1\%$ AUC on the Criteo dataset at a small batch size. When scaling to a large batch size in Table 5 and Table 12, instead of AUC loss, our algorithm achieves a further performance gain of about $0.1\%$ in the Criteo. For Criteo-seq and Avazu, CowClip can scale up to $128\times$ and $64\times$ batch size without performance loss respectively. Equipped with CowClip, it can scale all four models to a large batch size with performance improvement, as shown in Table 5 and Table 12. This shows that CowClip is a model-agnostic optmization technique.

Training with large batches can significantly reduce the training time. As shown in Appendix, for both the Criteo dataset and the Avazu dataset, the speedup achieved by scaling up the batch size is almost linear when the batch size is under 16K. We can still accomplish a sublinear speedup when continuing to scale up the batch size and achieve a $76.8\times$ speed up with 128K batch size on the Criteo dataset. As a result, we can finish the training on Criteo dataset with DeepFM model within **10 minutes**. Our method takes the advantage of large batch training, which can achieve a much shorter training time within only one GPU resource and obtain a higher AUC score.

## Ablation Study

Next, we show the superiority of CowClip over other clipping method designs with DeepFM on the Criteo dataset at batch size 128K. Table 6 gives the ablation of different gradient clipping designs. GC means the traditional gradient norm clipping and it fails with b=128K. For the embedding table, we have two granularity: field and column (e.g., "Device" is a field, and "Mobile", "Computer" are columns). Field-wise GC and Column-wise GC show that gradient clipping on fine-grained granularity yields better results. The next two lines add the adaptive design to the clipping on the above two granularities, which adaptively decide the clipping values for each column (line 8 in Alg. 1). The reason that Field-wise adaptive GC fails to achieve a good result is that magnitudes of column gradients are different even in a field. Thus, Gradient clipping applied to a smaller unit yields better performance. CowClip (Adaptive Column-wise GC) outperforms all other methods in both settings. Hyperparameters for these clipping variants and more ablation studies into the effectiveness of each component of CowClip can be found in the Appendix, which shows each component contributes to the final results.

| | **b = 128K** | |
|---|---|---|
| | AUC (%) | LogLoss |
| Gradient Clipping (GC) | 77.24 | 0.4953 |
| Field-wise GC | 80.62 | 0.4454 |
| Column-wise GC | 80.75 | 0.4432 |
| Adaptive Field-wise GC | 77.90 | 0.4824 |
| Adaptive Column-wise GC | 80.90 | 0.4430 |

Table 6: Ablation study of CowClip on Criteo with DeepFM.

# Conclusion

To accelerate the training of CTR prediction models on one GPU, we have explored large batch training and found that different frequencies hinder the scaling of learning rate and L2-regularization weight when scaling the batch size. Since previous scaling rules used in CV and NLP fail, we propose a novel optimization strategy CowClip with a simple scaling rule to stabilize the training process for large batch training in CTR prediction system. Experiments show that our method successfully scales the batch size to the state-of-the-art number and achieves significant training speedup. Our CowClip algorithm is also applicable to other tasks with a large embedding table such as NLP tasks.

## Acknowledgements

We thank Google TFRC for supporting us to get access to the Cloud TPUs. This work is supported NUS startup grant, the Singapore MOE Tier-1 grant, and the ByteDance grant.

## References

Abadi, M.; Agarwal, A.; Barham, P.; Brevdo, E.; Chen, Z.; Citro, C.; Corrado, G. S.; Davis, A.; Dean, J.; Devin, M.; Ghemawat, S.; Goodfellow, I.; Harp, A.; Irving, G.; Isard, M.; Jia, Y.; Jozefowicz, R.; Kaiser, L.; Kudlur, M.; Levenberg, J.; Mané, D.; Monga, R.; Moore, S.; Murray, D.; Olah, C.; Schuster, M.; Shlens, J.; Steiner, B.; Sutskever, I.; Talwar, K.; Tucker, P.; Vanhoucke, V.; Vasudevan, V.; Viégas, F.; Vinyals, O.; Warden, P.; Wattenberg, M.; Wicke, M.; Yu, Y.; and Zheng, X. 2015. TensorFlow: Large-Scale Machine Learning on Heterogeneous Systems. https://www.tensorflow.org/. Accessed: 2022-03-28.

Adnan, M. 2021. *Accelerating input dispatching for deep learning recommendation models training*. Ph.D. thesis, University of British Columbia.

Adnan, M.; Maboud, Y. E.; Mahajan, D.; and Nair, P. J. 2021. Accelerating Recommendation System Training by Leveraging Popular Choices. In *Proceedings of the VLDB Endowment*, volume 15, 127–140.

Avazu. 2015. Avazu Click-Through Rate Prediction. https://www.kaggle.com/c/avazu-ctr-prediction. Accessed: 2022-03-28.

Baji, T. 2018. Evolution of the GPU Device widely used in AI and Massive Parallel Processing. In *2018 IEEE 2nd Electron Devices Technology and Manufacturing Conference (EDTM)*, 7–9. IEEE.

Chen, Q.; and Li, D. 2021. Improved CTR Prediction Algorithm based on LSTM and Attention. In *Proceedings of the 5th International Conference on Control Engineering and Artificial Intelligence*, 122–125.

Chen, Q.; Zhao, H.; Li, W.; Huang, P.; and Ou, W. 2019. Behavior sequence transformer for e-commerce recommendation in alibaba. In *Proceedings of the 1st International Workshop on Deep Learning Practice for High-Dimensional Sparse Data*, 1–4.

Cheng, H.-T.; Koc, L.; Harmsen, J.; Shaked, T.; Chandra, T.; Aradhye, H. B.; Anderson, G.; Corrado, G. S.; Chai, W.; Ispir, M.; Anil, R.; Haque, Z.; Hong, L.; Jain, V.; Liu, X.; and Shah, H. 2016. Wide & Deep Learning for Recommender Systems. In *Proceedings of the 1st Workshop on Deep Learning for Recommender Systems*.

Covington, P.; Adams, J. K.; and Sargin, E. 2016. Deep Neural Networks for YouTube Recommendations. In *Proceedings of the 10th ACM Conference on Recommender Systems*.

Deng, W.; Pan, J.; Zhou, T.; Flores, A.; and Lin, G. 2021. DeepLight: Deep Lightweight Feature Interactions for Accelerating CTR Predictions in Ad Serving. In *Proceedings of the 14th ACM International Conference on Web Search and Data Mining*.

Ginart, A. A.; Naumov, M.; Mudigere, D.; Yang, J.; and Zou, J. Y. 2021. Mixed Dimension Embeddings with Application to Memory-Efficient Recommendation Systems. In *2021 IEEE International Symposium on Information Theory (ISIT)*, 2786–2791.

Gomez-Uribe, C.; and Hunt, N. 2016. The Netflix Recommender System: Algorithms, Business Value, and Innovation. *ACM Transactions on Management Information Systems (TMIS)*, 6: 13:1–13:19.

Gotmare, A. D.; Keskar, N. S.; Xiong, C.; and Socher, R. 2019. A Closer Look at Deep Learning Heuristics: Learning rate restarts, Warmup and Distillation. *arXiv*, abs/1810.13243.

Goyal, P.; Dollár, P.; Girshick, R. B.; Noordhuis, P.; Wesolowski, L.; Kyrola, A.; Tulloch, A.; Jia, Y.; and He, K. 2017. Accurate, Large Minibatch SGD: Training ImageNet in 1 Hour. *arXiv*, abs/1706.02677.

Guo, H.; Tang, R.; Ye, Y.; Li, Z.; He, X.; and Dong, Z. 2018. DeepFM: An End-to-End Wide & Deep Learning Framework for CTR Prediction. *arXiv*, abs/1804.04950.

He, X.; Xue, F.; Ren, X.; and You, Y. 2021. Large-Scale Deep Learning Optimizations: A Comprehensive Survey. *arXiv*, abs/2111.00856.

Hoffer, E.; Hubara, I.; and Soudry, D. 2017. Train longer, generalize better: closing the generalization gap in large batch training of neural networks. *arXiv*, abs/1705.08741.

Kaasinen, E.; Roto, V.; Roloff, K.; Väänänen-Vainio-Mattila, K.; Vainio, T.; Maehr, W.; Joshi, D.; and Shrestha, S. 2009. User experience of mobile internet: analysis and recommendations. *International Journal of Mobile Human Computer Interaction (IJMHCI)*, 1(4): 4–23.

Kingma, D. P.; and Ba, J. 2015. Adam: A Method for Stochastic Optimization. *arXiv*, abs/1412.6980.

Krizhevsky, A. 2014. One weird trick for parallelizing convolutional neural networks. *arXiv*, abs/1404.5997.

Labs, C. 2014. Display Advertising Challenge. https://www.kaggle.com/c/criteo-display-ad-challenge. Accessed: 2022-03-28.

Li, Z.; Cui, Z.; Wu, S.; Zhang, X.; and Wang, L. 2019. Fi-GNN: Modeling Feature Interactions via Graph Neural Networks for CTR Prediction. In *Proceedings of the 28th ACM International Conference on Information and Knowledge Management*.

Ma, Y.; Narayanaswamy, B.; Lin, H.; and Ding, H. 2020. Temporal-Contextual Recommendation in Real-Time. In *Proceedings of the 26th ACM SIGKDD International Conference on Knowledge Discovery & Data Mining*.

Mattson, P.; Cheng, C.; Coleman, C. A.; Diamos, G. F.; Micikevicius, P.; Patterson, D.; Tang, H.; Wei, G.-Y.; Bailis, P.; Bittorf, V.; Brooks, D. M.; Chen, D.; Dutta, D.; Gupta, U.; Hazelwood, K. M.; Hock, A.; Huang, X.; Jia, B.; Kang, D.; Kanter, D.; Kumar, N.; Liao, J.; Ma, G.; Narayanan, D.; Oguntebi, T.; Pekhimenko, G.; Pentecost, L.; Reddi, V. J.; Robie, T.; John, T. S.; Wu, C.-J.; Xu, L.; Young, C.; and Zaharia, M. A. 2020. MLPerf Training Benchmark. *arXiv*, abs/1910.01500.

Miao, X.; Zhang, H.; Shi, Y.; Nie, X.; Yang, Z.; Tao, Y.; and Cui, B. 2021. HET: Scaling out Huge Embedding Model Training via Cache-enabled Distributed Framework. In *Proceedings of the VLDB Endowment*, volume 15, 312–320.

Mudigere, D.; Hao, Y.; Huang, J.; Jia, Z.; Tulloch, A.; Sridharan, S.; Liu, X.; Ozdal, M.; Nie, J.; Park, J.; Luo, L.; Yang, J. A.; Gao, L.; Ivchenko, D.; Basant, A.; Hu, Y.; Yang, J.; Ardestani, E. K.; Wang, X.; Komuravelli, R.; Chu, C.-H.; Yilmaz, S.; Li, H.; Qian, J.; Feng, Z.; Ma, Y.-A.; Yang, J.; Wen, E.; Li, H.; Yang, L.; Sun, C.; Zhao, W.; Melts, D.; Dhulipala, K.; Kishore, K. G.; Graf, T.; Eisenman, A.; Matam, K. K.; Gangidi, A.; Chen, G. J.; Krishnan, M.; Nayak, A.; Nair, K.; Muthiah, B.; khorashadi, M.; Bhattacharya, P.; Lapukhov, P.; Naumov, M.; Mathews, A. S.; Qiao, L.; Smelyanskiy, M.; Jia, B.; and Rao, V. 2021. Software-Hardware Co-design for Fast and Scalable Training of Deep Learning Recommendation Models. *arXiv*, abs/2104.05158.

Naumov, M.; Mudigere, D.; Shi, H.-J. M.; Huang, J.; Sundaraman, N.; Park, J.; Wang, X.; Gupta, U.; Wu, C.-J.; Azzolini, A. G.; Dzhulgakov, D.; Mallevich, A.; Cherniavskii, I.; Lu, Y.; Krishnamoorthi, R.; Yu, A.; Kondratenko, V.; Pereira, S.; Chen, X.; Chen, W.; Rao, V.; Jia, B.; Xiong, L.; and Smelyanskiy, M. 2019. Deep Learning Recommendation Model for Personalization and Recommendation Systems. *arXiv*, abs/1906.00091.

Shen, W. 2017. DeepCTR: Easy-to-use,Modular and Extendible package of deep-learning based CTR models. https://github.com/shenweichen/deepctr. Accessed: 2022-03-28.

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

Wang, R.; Shivanna, R.; Cheng, D. Z.; Jain, S.; Lin, D.; Hong, L.; and Chi, E. H. 2021. DCN V2: Improved Deep & Cross Network and Practical Lessons for Web-scale Learning to Rank Systems. In *Proceedings of the Web Conference 2021*.

Wang, X. 2020. A Survey of Online Advertising Click-Through Rate Prediction Models. In *2020 IEEE International Conference on Information Technology, Big Data and Artificial Intelligence (ICIBA)*, volume 1, 516–521.

Xie, M.; Ren, K.; Lu, Y.; Yang, G.; Xu, Q.; Wu, B.; Lin, J.; Ao, H.; Xu, W.; and Shu, J. 2020. Kraken: memory-efficient continual learning for large-scale real-time recommendations. In *International Conference for High Performance Computing, Networking, Storage, and Analysis (SC)*.

You, Y.; Gitman, I.; and Ginsburg, B. 2017. Large Batch Training of Convolutional Networks. *arXiv*, abs/1708.03888.

You, Y.; Li, J.; Reddi, S. J.; Hseu, J.; Kumar, S.; Bhojanapalli, S.; Song, X.; Demmel, J.; Keutzer, K.; and Hsieh, C.-J. 2020. Large Batch Optimization for Deep Learning: Training BERT in 76 minutes. *arXiv*, abs/1904.00962.

Zhang, J.; He, T.; Sra, S.; and Jadbabaie, A. 2020. Why Gradient Clipping Accelerates Training: A Theoretical Justification for Adaptivity. *arXiv*, abs/1905.11881.

Zhang, J.; Huang, T.; and Zhang, Z. 2019. FAT-DeepFFM: Field Attentive Deep Field-aware Factorization Machine. In *ICDM*.

Zhao, W.; Zhang, J.; Xie, D.; Qian, Y.; Jia, R.; and Li, P. 2019. AIBox: CTR Prediction Model Training on a Single Node. In *Proceedings of the 28th ACM International Conference on Information and Knowledge Management*.

Zhou, G.; Mou, N.; Fan, Y.; Pi, Q.; Bian, W.; Zhou, C.; Zhu, X.; and Gai, K. 2019. Deep Interest Evolution Network for Click-Through Rate Prediction. *arXiv*, abs/1809.03672.

Zhu, J.; Liu, J.; Yang, S.; Zhang, Q.; and He, X. 2021. Open Benchmarking for Click-Through Rate Prediction. In *Proceedings of the 30th ACM International Conference on Information & Knowledge Management*.