CAFE: Towards Compact, Adaptive, and Fast Embedding for Large-scale Recommendation Models

Hailin Zhang*, Zirui Liu*, Boxuan Chen, Yikai Zhao, Tong Zhao, Tong Yang, Bin Cui
School of Computer Science, Peking University
z.hl@pku.edu.cn, zirui.liu@pku.edu.cn, 2100012923@stu.pku.edu.cn, zyk@pku.edu.cn, zhaotong@pku.edu.cn,
yangtongemail@gmail.com, bin.cui@pku.edu.cn

ABSTRACT

Recently, the growing memory demands of embedding tables in Deep Learning Recommendation Models (DLRMs) pose great challenges for model training and deployment. Existing embedding compression solutions cannot simultaneously meet three key design requirements: memory efficiency, low latency, and adaptability to dynamic data distribution. This paper presents **CAFE**, a **C**ompact, **A**daptive, and **F**ast **E**mbedding compression framework that addresses the above requirements. The design philosophy of CAFE is to dynamically allocate more memory resources to important features (called hot features), and allocate less memory to unimportant ones. In CAFE, we propose a fast and lightweight sketch data structure, named HotSketch, to capture feature importance and report hot features in real time. For each reported hot feature, we assign it a unique embedding. For the non-hot features, we allow multiple features to share one embedding by using hash embedding technique. Guided by our design philosophy, we further propose a multi-level hash embedding framework to optimize the embedding tables of non-hot features. We theoretically analyze the accuracy of HotSketch, and analyze the model convergence against deviation. Extensive experiments show that CAFE significantly outperforms existing embedding compression methods, yielding 3.92% and 3.68% superior testing AUC on Criteo Kaggle dataset and CriteoTB dataset at a compression ratio of 10000×. The source codes of CAFE are available at GitHub [1].

KEYWORDS

Embedding; Deep Learning Recommendation Model; Sketch

1 INTRODUCTION

1.1 Background and Motivation

In recent years, embedding techniques are widely applied in various fields in database community, such as cardinality estimation [2, 3], query optimization [4, 5], language understanding [6], entity resolution [7, 8], document retrieval [9], graph learning [10, 11], and advertising recommendation [12], to learn the semantic representations of categorical features. Among these fields, Deep Learning Recommendation Models (DLRMs) are one of the most important applications of embedding techniques: they account for 35% of Amazon's revenue in 2018 [13–15], and consume more than 50% training and 80% inference cycles at Meta's data centers in 2020 [16, 17].

As shown in Figure 1, a typical DLRM vectorizes categorical features into learnable embeddings, and then feeds these embeddings into downstream neural networks along with other numerical features [18–23]. Recently, with the exponential increase of categorical features in DLRM, the memory requirements of embedding tables have also skyrocketed, which creates formidable storage challenges in various applications [24, 25]. Therefore, it is highly desired to devise a framework that can effectively compress the embedding tables into limited storage space without compromising model accuracy. In this paper, we focus on compressing the embedding tables of extremely large-scale DLRMs.

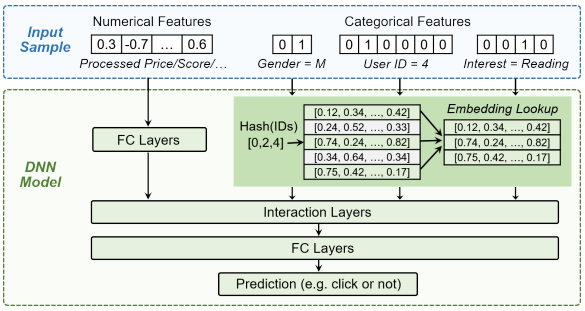


Figure 1: Overview of DLRM.

DLRM has two training paradigms: offline training and online training. (1) In offline training, the training data is collected in advance, and the model is deployed for use after the entire training process. (2) In online training, the training data is generated in real time, and the model simultaneously updates parameters and serves requests. This paper mainly focuses on the scenario of online training as it is more difficult. Generally, compression methods for online training can be directly applied to offline training. Embedding compression in online training has three important design requirements, which are as follows:

- **Memory efficiency.** For extremely large-scale DLRMs, it is challenging to maintain model quality within memory constraints. While distributed instances can help manage large-scale embedding tables, they come with a significant communication overhead [26, 27]. Furthermore, training and deployment of embedding tables often occur on edge or end devices with smaller storage capacities, making the memory issue even worse [28]. On the other hand, since model quality directly impacts profits, even a small change of 0.001 in DLRM's AUC (area under the ROC curve) is considerable [29]. Existing compression methods often lead to severe model degradation when memory constraints are small [30], emphasizing the need for memory-efficient compression methods that maintain model quality.
- **Low latency.** Low latency is a vital requirement in practical applications, as latency is a key metric of service quality [31]. Embedding compression methods must be fast enough not to introduce significant latency.

* Equal contribution.

- **Adaptability to dynamic data distribution.** In online training, the data distribution is not fixed as in offline training. We calculate the KL divergence (an asymmetric measure of the distance between distributions) between the feature distributions on each day within three common public datasets, and plot the heatmaps in Figure 2. In each heatmap, the block in row i, column j shows the KL divergence between the distributions on day i and day j. There is a significant difference between the feature distributions, and generally the greater the number of days between, the greater the difference. Existing advanced compression methods often exploit feature frequencies explicitly [32, 33] or capture feature importance implicitly [34, 35], which are inspired by the observation that feature popularity distributions are highly skewed, fitting zipfian [24] or powerlaw distributions [36]. However, most of them rely on fixed data distributions and cannot be applied to dynamic data distributions, demanding new adaptive compression method suitable for online training.

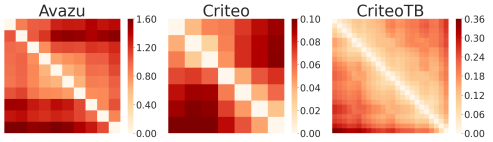


Figure 2: KL divergence between distributions on each day.

1.2 Limitations of Prior Art

Existing embedding compression methods can be generally categorized into two types: row compression and column compression. As column compression primarily aims at enhancing model quality rather than compressing to a specific memory limit, our focus is on row compression, including hash-based and adaptive methods.

Hash-based methods. These methods utilize simple hash functions to map features into embeddings with collisions [37–39]. They restrict the number of embedding vectors to fit within the memory budget, causing different features to potentially share an embedding vector when a hash collision occurs. Despite their simplicity and convenience, which have resulted in widespread industry use, these methods are not very memory-efficient. Pre-determined hash functions distort the semantic information of features, often leading to a substantial decline in model accuracy. For each feature, the gradients of other hash-collided features will be updated to the same embedding, resulting in deviations from the original convergence direction. Integrating feature frequency information [32] can enhance hash-based methods' model quality in offline training but cannot be applied to online training.

Adaptive methods. To accommodate online training, adaptive methods distinguish and track important features throughout the training process. AdaEmbed [40] logs the importance scores of all features, dynamically reallocates embedding vectors for critical features, and discards embeddings of less important features. While it can adapt to data distribution, its compression ratio is constrained by the storage of importance scores, which increases linearly with the total number of features. Thus, it cannot compress embedding tables to a small memory budget and still needs distributed training for large models, resulting in low memory efficiency. It also needs to sample and check data to determine whether to migrate embeddings, which can increase overall latency.

In summary, existing methods fail to meet all three critical requirements for DLRM: memory efficiency, low latency, and adaptability. In this paper, we aim to propose an embedding compression method that is memory-efficient, adaptive, and ensures low latency.

1.3 Our Proposed Method

We introduce **CAFE**, a **C**ompact, **A**daptive, and **F**ast **E**mbedding compression method, which, to our knowledge, is the first to satisfy all three design requirements. **(a) Memory efficiency:** CAFE allocates unique embedding vectors to important features and shared embedding vectors to less important features, thereby preserving model quality. A light-weight sketch, HotSketch, distinguishes these features, with its memory usage being linear to the number of important features, enabling high compression ratios. Consequently, CAFE manages to maintain good model quality within tight memory constraints. **(b) Low latency:** CAFE entails only several hash processes and potentially one additional embedding lookup, incurring negligible time overhead beyond the standard embedding layers and thus maintaining low latency during serving. **(c) Adaptability to dynamic data distribution:** CAFE incorporates an embedding migration process that takes effect when a feature's importance score changes, ensuring that vital features are always identified even when data distribution changes during online training. On Criteo dataset, compared to existing methods, CAFE improves the model AUC by 1.79% and reduces the training loss by 2.31% on average.

To achieve a high compression ratio without compromising model quality, we utilize a sketch structure to distinguish and record important features from a highly skewed Zipf distribution. Sketches are a class of probabilistic data structures for processing large-scale data streams, and are naturally suitable for handling streamed features in online training. Specifically, we extend Space-Saving Sketch [41], an advanced sketch algorithm with small error, to design HotSketch, a less memory-consuming structure to store important DLRM features with a theoretically guaranteed error bound. Being a light-weight data structure, HotSketch incurs negligible time overhead, facilitating fast training and inference. Since HotSketch's memory usage is only linear to the number of important features, CAFE can compress to any given memory constraints. With HotSketch, we allocate unique embeddings to a handful of important features and shared embeddings to a vast majority of long-tail features, achieving memory efficiency.

To adapt CAFE to online training, where important features can change dynamically, we enable features to migrate between unique and shared embedding tables. If a feature's importance score exceeds a relative threshold in HotSketch, it is deemed important and allocated a unique embedding. Conversely, if a feature's importance score drops below a relative threshold, its unique embedding migrates to the shared embedding table.

To further optimize CAFE, we divide features into more groups by importance scores. While the most critical features are still allocated unique embeddings, other features are assigned a varying number of hashed embedding vectors. This multi-level design further improves the model AUC by 0.08% on Criteo dataset.

1.4 Main Contribution

- We introduce CAFE, a compact, adaptive, and fast embedding compression method.

- We propose HotSketch, a light-weight sketch structure to discern and record features' importance scores.
- We provide a theoretical analysis of HotSketch's effectiveness, and elucidate how CAFE's design contributes to the convergence of compressed DLRMs.
- We evaluate CAFE on representative DLRM datasets, achieving 3.92%, 3.68%, 5.16% higher testing AUC and 4.61%, 3.24%, 11.21% lower training loss at 10000× compression ratio compared to existing method.

2 PRELIMINARY

In this section, we elaborate on the architecture of DLRMs in Section 2.1 and provide a formal definition of the embedding compression problem in Section 2.2.

2.1 DLRM

Figure 1 illustrates the overall architecture of DLRM. Each dataset of DLRM has several categorical feature fields and numerical feature fields. For example, in Figure 1, gender, user ID and interest are categorical fields, while price and score are numerical fields. Each field has a certain number or a certain range of possible values, called features. Categorical and numerical features are transformed into representations using embedding vectors and fully-connected layers, respectively. The representations are then fed into interaction layers and fully-connected layers for final predictions. The prediction may be a category for classification tasks such as click-through-rate and conversion-rate prediction, or a score for regression tasks such as score prediction. There are many variants of DLRM, such as WDL [18], DCN [42], DIN [43]; while they all utilize the same embedding layer, they explore different forms of interaction layers and neural network layers to enhance model performance.

The size of a DLRM does not depend on the model structure, but on the number of unique categorical features in the dataset. The model parameters of DLRMs can be divided into two parts: the embedding table and the neural network. The former contains embeddings for all categorical features, *i.e.*, one embedding per feature if uncompressed. The latter is a network that interacts these embeddings and outputs predictions. The number of parameters in the embedding table depends on the dataset: if there are n unique categorical features in the dataset, and the dimension of embeddings is d, then the number of parameters is $n \times d$. In DLRMs, the size of the neural network part (just a few layers of matrix multiplication) is negligible compared to large embedding tables. Based on previous research works [25, 44–47], we consider DLRMs with more than 100 million parameters as large-scale, and DLRMs with more than 10 billion parameters as extremely large-scale.

In DLRMs, categorical features are viewed as one-hot vectors where only the i-th position is set to 1 and the rest are set to 0, facilitating the retrieval of the corresponding row vector from the embedding table. Each input data, sampled from distribution $\mathcal{D}$, contains categorical features x_{cat}, numerical features x_{num}, and a label y. We denote E as the embedding tables and f as the other neural network layers, then the process of minimizing the loss can be formulated as follows:

$$\min_{E,f} \mathbb{E}_{(x_{cat},x_{num},y)\sim\mathcal{D}} \mathcal{L}(y, f(E(x_{cat}), x_{num})). \tag{1}$$

After the forward pass, an optimizer such as Adam [48] is applied to iteratively update the embedding table and other parameters. Frequently used notations in this paper are detailed in Table 1.

Table 1: Symbols frequently used in this paper.

Symbols	Meaning
$\mathcal{D}$	Distribution of input data
$\mathcal{D}_t$	Shifting distribution of input data at time t
n	Number of unique categorical features
d	Embedding dimension
x_{cat}	Categorical feature
x_{num}	Numerical feature
y	Ground truth label
$\hat{y}$	Prediction
E	Embedding table
E^*	Compressed embedding table
f	Neural network
θ	Learnable parameters
α	Learning rate
g	Standard gradient without compression
$\widetilde{g}$	Gradient in compressed DLRM
$\mathcal{L}$	Loss function
$\mathcal{M}$	Memory usage (of the embedding table)
M	Memory budget
CR	Compression ratio
w	Number of buckets in HotSketch
c	Number of slots in each bucket in HotSketch
k	Number of hot features

2.2 Embedding Compression

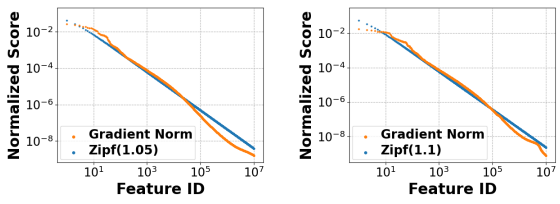
Embedding compression is mainly conducted within a memory constraint. Denoting M as the memory budget of the embedding table, $\mathcal{M}$ as the memory function mapping an embedding table to corresponding memory usage, and E^* as the compressed embedding table, the optimization of DLRM within a memory constraint is formulated as follows:

$$\begin{aligned} &\min_{E^*,f} \mathbb{E}_{(x_{cat},x_{num},y)\sim\mathcal{D}} \mathcal{L}(y, f(E^*(x_{cat}), x_{num})), \\ &\text{s.t.} \mathcal{M}(E^*) \le M. \end{aligned} \tag{2}$$

The memory function excludes the memory usage of neural networks since it is fixed and negligible compared to the memory usage of embedding tables.

We define compression ratio as the multiple of the original memory to the compressed memory, to reflect the degree of compression: $CR = \frac{\mathcal{M}(E)}{\mathcal{M}(E^*)}$. In practical applications, using a compression ratio of 10× can reduce the cost of distributed deployment, 100× to 1000× can allow for single-device deployment, and an extreme compression ratio of 10000× can enable DLRMs on edge devices.

For online training, the fixed data distribution $\mathcal{D}$ in the above definition can be modified to a variable distribution $\mathcal{D}_t$, which is continuously evolving over time t.



(a) Criteo. (b) CriteoTB.

Figure 3: Comparing gradient norm and Zipf distributions.

3 CAFE DESIGN

3.1 CAFE Overview

Rationale: We design CAFE, an efficient embedding framework that is simultaneously compact, adaptive, and fast. The key idea of CAFE is to dynamically distinguish important features (called hot features) from unimportant ones (called non-hot features), and allocate more resources to hot features. Specifically, we define the importance score of a feature using the L2-norm of its gradient, which is proven to have good theoretical properties in Section 3.5.2 and also in previous works [40, 49, 50]. We further experimentally demonstrate the effectiveness of gradient norms in Section 5.3. We observe that in most training data, the feature importance follows a highly skewed distribution, where most features have small importance scores and only a small fraction of hot features are very important. For example, Figure 3 illustrates that the feature importance distributions in Criteo dataset and CriteoTB dataset are highly consistent with Zipf distributions of parameters 1.05 and 1.1, respectively. Therefore, if we can allocate more memory to the embedding of hot features and less memory to that of non-hot features, it is possible to significantly improve the model quality under the same memory usage of embedding tables.

As shown in Figure 4, in CAFE, we propose a novel sketch algorithm, called HotSketch, to capture feature importance and report top-k hot features in real time (Section 3.2). In each training iteration, we first fetch data samples from the input training data, and query each feature from these samples in HotSketch. For each feature, HotSketch reports its current importance score, and if the score exceeds a predefined hot feature threshold, we regard this feature as a hot feature. We then lookup the embeddings for hot features and non-hot features respectively. In CAFE, for each hot feature, we allocate a unique embedding, and we store the pointer to this embedding in HotSketch. For the non-hot features, we use hash embedding tables where multiple features can share one embedding. We will discuss how to migrate embeddings between the tables of hot features and non-hot features in Section 3.3. Guided by our design philosophy, we further propose a multi-level hash embedding framework to better embrace the skewed feature importance distribution (Section 3.4). Afterwards, we feed the embeddings into the downstream neural network for prediction and get the gradient norm for each feature. Finally, we update the importance of these features in HotSketch using their gradient norms.

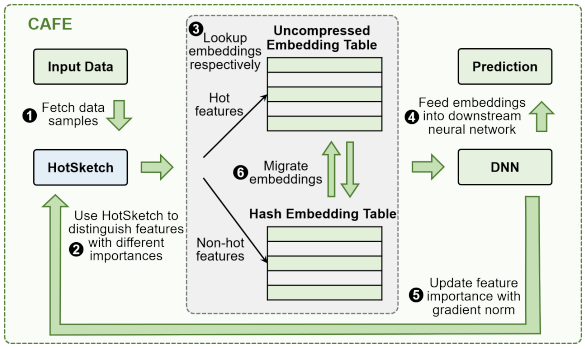


Figure 4: Overview of CAFE.

3.2 The HotSketch Algorithm

Rationale: We design HotSketch to capture hot features with high importance scores in a single pass, which is essentially a problem of finding top-k frequent items (features) in streaming data. Currently, Space-Saving [51] is the most recognized algorithm for solving top-k problem. It maintains frequent items in sorted doubly linked list and uses a hash table to index this list. However, this hash table not only doubles the memory usage but also imposes time inefficiency due to numerous memory accesses caused by pointer operations. Based on the idea of Space-Saving, we propose HotSketch, which removes the hash table while still maintaining the $O(1)$ time complexity. We theoretically prove that our HotSketch well inherits the theoretical results of Space-Saving (Section 3.5), and empirically validate the performance of HotSketch (Section 5.6).

Data structure: As depicted in Figure 5, HotSketch consists of an array of w buckets $\mathcal{B}[1], \cdots, \mathcal{B}[w]$. We use a hash function $h(\cdot)$ to map each feature into one bucket. Each bucket contains c slots. Each slot stores a feature ID and its importance score.

Insertion: For each incoming feature f_i associated with an importance score s_i, we first calculate the hash function to locate a bucket $\mathcal{B}[h(f_i)]$, termed as the hashed bucket of f_i. Then, we check bucket $\mathcal{B}[h(f_i)]$ and encounter three possible scenarios: (1) f_i is recorded in $\mathcal{B}[h(f_i)]$. We add s_i to its importance score. (2) f_i is not recorded in $\mathcal{B}[h(f_i)]$ and there exists an empty slot in $\mathcal{B}[h(f_i)]$. We insert f_i into the empty slot by setting this slot to (f_i, s_i). (3) f_i is not recorded in $\mathcal{B}[h(f_i)]$ and $\mathcal{B}[h(f_i)]$ is full. We locate the feature with the smallest score (f_{min}, s_{min}), replace f_{min} with f_i, and add s_i to s_{min}. In other words, we set the slot (f_{min}, s_{min}) to $(f_i, s_{min} + s_i)$. Figure 5 shows an example of insertion.

Discussion: HotSketch has the following advantages: (1) HotSketch has fast insertion speed. It processes each feature in a one-pass manner and has an $O(1)$ time complexity. In addition, HotSketch avoids complicated pointer operations and has only one memory access. (2) HotSketch is memory-efficient. It does not store pointers, and there are no empty slots in HotSketch after a brief cold start. (3) HotSketch is hardware-friendly and can be accelerated with multi-threading and SIMD, thereby achieving superior data parallelism.

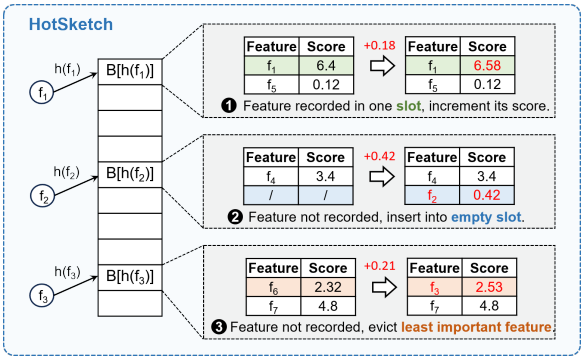


Figure 5: The HotSketch algorithm.

3.3 Migration Strategy

During the online training of DLRMs, the distribution of feature importance fluctuates with data distribution changes, meaning that the hot features are not constant throughout the training process. Since HotSketch already records the feature importance during

training, it can naturally support dynamic hot features by embeddings migration between uncompressed and hash embedding tables.

In HotSketch, we set a threshold to distinguish hot features, and the entry and exit of hot features occur throughout the training process. Almost every feature that reaches HotSketch for the first time is considered a non-hot feature with a low importance score. When a non-hot feature's importance score surpasses the threshold, it transitions into a hot feature, and its embedding migrates from the shared table to the uncompressed table as initialization, ensuring the feature's representation remains smooth throughout the training process. Conversely, when a hot feature's importance score drops below the threshold, it becomes a non-hot feature, and its embedding is discarded from the uncompressed table. Considering that the newly migrated non-hot feature is no longer important, its original exclusive embedding is simply ignored and the shared embedding is used instead. The threshold is meticulously set, allowing HotSketch to always saturate with hot features and adapt to distribution changes. If the importance scores alter rapidly, we can also decay the scores periodically.

During training, it's vital to maintain an appropriate migration frequency. If the migration occurs too frequently, the learning process may not be smooth enough due to the replacement of embeddings, and the migration will generate substantial delay. Conversely, if the migration occurs too infrequently, HotSketch cannot capture changes in the distribution, leading to a decline in model quality. By setting a suitable threshold in HotSketch, a moderate migration frequency can enable the model to adapt to changes in distribution without negatively impacting convergence and latency.

3.4 Multi-level Hash Embedding

In HotSketch, features are categorized into hot and non-hot features, with the latter outnumbering the former, considering typical compression ratios ranging from 10× to 1000×. A substantial number of non-hot features are treated identically in HotSketch, sharing a hash embedding table with the same rate of collisions. Given that these features' importance scores also conform to a highly skewed Zipf distribution, it's logical to further segregate non-hot features based on their importance scores and assign different hash embedding tables to them. Therefore, we integrate multi-level hash embedding, as shown in Figure 6.

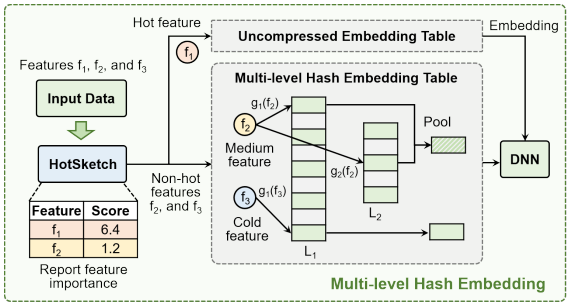


Figure 6: Overview of multi-level hash embedding.

With multi-level hash embedding, we partition non-hot features into more refined categories of different importance levels, and assign to them different number of embeddings from multiple tables. For simplicity, we focus on 2-level hash embedding, where non-hot features are divided into medium features and cold features. We expand the functionality of HotSketch to identify medium features by estimating their importance scores. Since medium features are more significant, they reference 2 embedding vectors from 2 distinct hash embedding tables, while cold features only look up a single embedding vector. This design draws inspiration from prior hash-based methods [32] that also adopt multiple embedding vectors to enrich representations and boost model quality.

We illustrate the multi-level embedding process using an example in Figure 6. (1) Input features f_1, f_2, f_3 are fed into HotSketch. Among them f_1 has a score larger than the hot threshold, f_2 has a score above the medium threshold, and f_3 has a score lower than the thresholds, so they are classified as hot, medium, and cold features respectively. (2) Hot and cold features look up the embedding vectors as before. (3) Medium feature f_2 looks up two embedding vectors from two hash embedding tables, and obtains the final embedding through a pooling process. To ensure that the training process remains smooth, the hash function is combined with hash tables. When a feature is migrated between middle and cold classes, it always retrieves the same embedding vector from the first embedding table. For pooling operation, in practice, we find that simple summation of embeddings performs well, since a feature's embedding vectors are always updated in the same direction.

The design of the multi-level hash embedding is based on the observation that a unique embedding is a comprehensive representation with no information loss, whereas, for hash embeddings, the larger the number of embeddings involved, the fewer the collisions and the more information a feature can retain. Through experiments detailed in Section 5.4, we find that multi-level hash embedding performs better, with a reduction of 0.25% in training loss and an increase of 0.08% in testing AUC.

3.5 Theoretical Analysis

3.5.1 *Accuracy of HotSketch*.

In this section, we theoretically analyze the performance of HotSketch in finding top-k important features. We derive the probability that a feature with top-k importance score is recorded in HotSketch.

Theorem 3.1. *Given a data stream with n features, and suppose their importance score vector is* $a = \{a_1, a_2, \cdots, a_n\}$, *where* $a_1 \geqslant a_2 \geqslant \cdots \geqslant a_n$. *Suppose that our HotSketch has w buckets, and each bucket contains c cells. Without distribution assumption, for an important feature with a total score larger than* $\gamma\|a\|_1$, *it can be held in HotSketch with probability at least:* $\Pr > 1 - \frac{1-\gamma}{(c-1)\gamma w}$.

Proof. The expected score sum of the other features $\hat{f}$ entering the same bucket is: $E\left[\hat{f}\right] = \frac{(1-\gamma)\|a\|}{w}$.

By following the properties of SpaceSaving algorithm, if the score $\hat{f}$ of the other features entering the bucket is no more than $(c-1)\gamma\|a\|_1$, then the feature must be held in the bucket. Using Markov inequality, we have $\Pr\left(\hat{f} > (c-1)\gamma\|a\|_1\right) \leqslant \frac{1-\gamma}{(c-1)\gamma w}$, which means that $\Pr > 1 - \frac{1-\gamma}{(c-1)\gamma w}$. □

Lemma 3.2. *Given a data stream with score vector* $a = \{a_1, a_2, \cdots, a_n\}$, *where* $a_1 \geqslant a_2 \geqslant \cdots \geqslant a_n$. *Suppose that a follows a Zipfian distribution with parameter z, meaning that* $a_i = \frac{a_1}{i^z}$. *Suppose our HotSketch has w buckets, and each bucket contains c cells.*

Then the mathematical expectation of the score sum of the features entering each bucket is: $E[\hat{f}] \leqslant \frac{\|a\|_1 \cdot k^{1-z}}{w}$ *with probability at least* $3^{-\frac{k}{w}}$ *for* $z > 1$ *and* $n \to +\infty$.

Proof. The probability that the k hottest features are not hashed into this bucket is: $\left(1-\frac{1}{w}\right)^k = \left(\left(1-\frac{1}{w}\right)^w\right)^{\frac{k}{w}} > 3^{-\frac{k}{w}}$.

When $w \geqslant 6$, $\left(1-\frac{1}{w}\right)^w$ increases monotonically with w. The expected score sum of the non-hot features entering this bucket is:

$$\begin{aligned} E[\hat{f}] =& \frac{\sum_{i=k+1}^{n} a_i}{w} = \frac{\sum_{i=k+1}^{n} \frac{a_1}{i^z}}{w} = \frac{\|a\|_1}{w} \cdot \left(\sum_{i=k+1}^{n} i^{-z}\right) \cdot \frac{1}{\sum_{i=1}^{n} i^{-z}} \\ \leqslant& \frac{\|a\|_1}{w} \cdot \left(\int_k^{+\infty} x^{-z} dx\right) \cdot \left(\int_1^{+\infty} x^{-z} dx\right)^{-1} \\ \leqslant& \frac{\|a\|_1}{w} \cdot \frac{k^{1-z}}{z-1} \cdot (z-1) = \frac{\|a\|_1 k^{1-z}}{w} \end{aligned}$$

for $z > 1$ and $n \to +\infty$. □

Theorem 3.3. *Given a data stream with score vector* $a = \{a_1, a_2, \cdots, a_n\}$, *where* $a_1 \geqslant a_2 \geqslant \cdots \geqslant a_n$. *Suppose that* a *follows a Zipfian distribution with parameter* z. *Suppose that our HotSketch has* w *buckets, and each bucket contains* c *cells. Then for a hot feature with a score larger than* $\gamma\|a\|_1$, *it can be held in the sketch with probability at least:* $\Pr > \sup_{\eta>0}\left(3^{-\eta} \cdot \left(1 - \frac{\eta}{(c-1)\gamma(\eta w)^z}\right)\right)$ *for* $z > 1$ *and* $n \to +\infty$.

Proof. The condition C that none of the k hottest features collide with this item holds with probability at least $3^{-\frac{k}{w}}$.

By following the properties of SpaceSaving algorithm, if the scores $\hat{f}$ of the other features entering the bucket is no more than $(c-1)\gamma\|a\|_1$, then the feature must be held in the bucket.

Using Markov inequality, we have

$$\Pr\left(\hat{f} > (c-1)\gamma\|a\|_1 \mid C\right) \leqslant \frac{\frac{\|a\|_1 \cdot k^{1-z}}{w}}{(c-1)\gamma\|a\|_1} = \frac{k^{1-z}}{(c-1)\gamma w}.$$

Then we have

$$\begin{aligned} \Pr\left(\hat{f} > (c-1)\gamma\|a\|_1\right) \leqslant& \Pr\left(\hat{f} > (c-1)\gamma\|a\|_1, C\right) + \Pr(\neg C) \\ \leqslant& 3^{-\frac{k}{w}} \cdot \left(\frac{k^{1-z}}{(c-1)\gamma w} - 1\right) + 1. \end{aligned}$$

Let $k = \eta w$, we have

$$\Pr\left(\hat{f} > (c-1)\gamma\|a\|_1\right) \leqslant 3^{-\eta} \cdot \left(\frac{1}{\eta^{z-1}(c-1)\gamma w^z} - 1\right) + 1.$$

And we have the probability that this feature must be held greater than

$$\Pr > \sup_{\eta>0}\left(3^{-\eta} \cdot \left(1 - \frac{\eta}{(c-1)\gamma(\eta w)^z}\right)\right).$$

□

Corollary 3.4. *The larger the parameter* c, w, z, *and* γ, *the larger the probability that the feature with score larger than* $\gamma\|a\|_1$ *be held in the sketch. The larger* c *and* w *means the larger memory used by sketch, the larger* z *means the more skew the data stream is, and the larger* γ *means the hotter the feature is.*

Proof. The following formula monotonically decreases with parameter c, w, z, and γ: $\frac{\eta}{(c-1)\gamma(\eta w)^z}$. □

Corollary 3.5. *To let the feature with score larger than* $\gamma\|a\|_1$ *be held with maximum probability in a fixed memory budget, the more skew the data stream is, the less cells per bucket should be used. Specifically, we recommend to use* $c = 1 + \frac{1}{z-1}$.

Proof. With a fixed memory budget $M = cw$, to minimize $\frac{\eta}{(c-1)\gamma(\eta w)^z}$, we should maximize $(c-1)w^z = \left(\frac{M}{w} - 1\right) w^z$.

As it has a derivative function

$$\left[\left(\frac{M}{w} - 1\right) w^z\right]' = ((z-1)M - zw)w^{z-2},$$

the optimal w should be $\frac{z-1}{z}M$, and the optimal c^* should be

$$c^* = \frac{z}{z-1} = 1 + \frac{1}{z-1}.$$

□

Discussion: From Corollary 3.5, we can see that under fixed memory usage ($M = cw$), the optimal c is affected by data distribution. Under non-skewed data distribution (small z), we should use larger c and smaller w to better approximate the results of basic SpaceSaving. Under highly skewed data distribution (large z), we should use smaller c and larger w to lower the impact of hash collisions between hot features. This might be because under highly skewed data, using small c can already guarantee us to find hot features with high probability. In this scenario, the performance of HotSketch is mainly affected by hash collisions between hot features. We surprisingly find that this corollary is consistent with our experimental results in Figure 18(a).

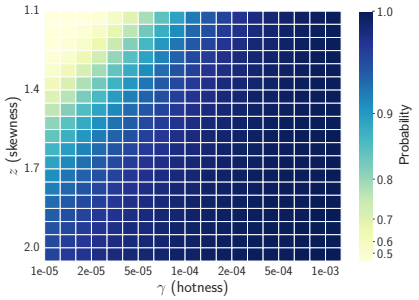


Figure 7: Numerical analysis for the probability of HotSketch identifying hot features, where the x-axis represents the hotness of the feature and the y-axis represents the skewness of the feature hotness distribution (Theorem 3.3).

Experimental analysis (Figure 7): Although we cannot directly obtain the analytical solution of Pr from Theorem 3.3, we can give the numerical solution of Pr under different γ and z by numerical simulation. In our simulation, we set $w = 10000$ and $c = 4$. We can see that larger z goes with higher Pr, showing that HotSketch is more suitable for capturing top-k features on skewed data distribution. In addition, larger γ also goes with higher Pr, showing that hotter features have larger probability of being captured by HotSketch. The results are consistent with our design goal.

3.5.2 ***Convergence Analysis against Deviation.***
As mentioned in Section 1.2, in hash-based methods, there will be deviations that can hinder the convergence of embeddings. CAFE aims to minimize the deviation of embedding gradients, which indeed reflects the deviation of embedding parameters. In this section, we analyze how this deviation affects the convergence of SGD algorithm. We study the following (non-convex) empirical risk minimization problem:

$$\min_{\theta \in \mathbb{R}^D} f(\theta) = \frac{1}{N} \sum_{i}^{N} f_i(\theta), \theta_{t+1} = \theta_t - \alpha \widetilde{g}_{i_t}$$

where α is learning rate, $g_{i_t}^i = \nabla f_i(\theta_{i_t})$ is the standard gradient without compression, $\widetilde{g}_{i_t}$ is the practical gradient with compression. We make the assumption below following [52].

Assumption. For $\forall i \in \{1, 2, ..., N\}, \theta, \theta' \in \mathbb{R}^D$, we make the following assumptions:

(1. L-Lipschitz) $\|\nabla f_i(\theta) - \nabla f_i(\theta')\| < L\|\theta - \theta'\|$;
(2. Bounded moment) $\mathbb{E}[\|\nabla f_i(\theta)\|] < \sigma_0, \mathbb{E}[\|\nabla f(\theta)\|] < \sigma_0$;
(3. Bounded variance) $\mathbb{E}[\|\nabla f_i(\theta) - \nabla f(\theta)\|] < \sigma$;
(4. Existence of global minimum) $\exists f^* s.t. f(\theta) \geq f^*$.

Theorem 3.6. *Suppose we run SGD optimization with CAFE on DLRMs satisfying the assumption above, with $\epsilon_t = \|\widetilde{g}_{i_t} - g_{i_t}\|$ as the deviation of embedding gradients. Assume the learning rate α satisfy $\alpha < \frac{1}{L}$. After T steps, for $\overline{\theta}_T$ which is randomly selected from $\{\theta_0, \theta_1, ..., \theta_{T-1}\}$, we have:*

$$\mathbb{E}[\|\nabla f(\overline{\theta}_T)\|^2] \leq \frac{f(\theta_0) - f^*}{T\alpha(1 - \alpha L)} + \frac{\alpha(2L\sigma^2 + \sigma_0^2)}{2(1 - \alpha L)} + \frac{(1 + \alpha^2 L) \sum_{t=0}^{T-1} \mathbb{E}[\epsilon_t^2]}{2T\alpha(1 - \alpha L)}$$

The proof is in the supplementary file on our GitHub page [1].

As T increases, with a proper learning rate $\alpha = O(\frac{1}{\sqrt{T}})$, the first two terms at the right hand side of above inequality tends to 0, and the convergence of SGD is mainly influenced by the deviation ϵ_t. In the scenario of compression, there is no bound for this deviation; yet the design of CAFE is proposed to minimize the deviation.

Since we assign those importance features with exclusive embedding parameters, their parameters have little deviation; for features sharing embeddings with each other, the deviation is introduced by the hash collisions. Generally, we cannot directly obtain the deviation of gradients, but according to the L-Lipschitz assumption, the deviation of gradients is bounded by the deviation of weights. As non-hot features share embeddings with each other, the deviation of weights comes from other features' gradients, which may disturb the learning direction. Based on this observation, CAFE choose to use gradient norm as the importance of features. For less important features, their gradient norms are relatively small, which limits the deviation of weights to some extent.

4 IMPLEMENTATION

We implement CAFE as a plug-in embedding layer module based on PyTorch. It can directly replace the original Embedding module in any PyTorch-based recommendation models with minor changes. Usage examples can be found on our GitHub page [1]. We consider extending CAFE to other frameworks (Hetu [53], etc.) in the future.

CAFE Backend: We implement the HotSketch algorithm in C++ to reduce the overall latency, and implement the rest of CAFE using PyTorch operators. For HotSketch, we set the number of bucket in HotSketch to the pre-determined number of hot features, with 4 slots per bucket. We use one sketch structure for all feature fields instead of one sketch per field, because the distribution of hot features across fields is unclear, which is better handled directly with importance scores.

Fault Tolerance: We register all HotSketch's states as buffers in CAFE's PyTorch module, so that the states can be saved and loaded alongside model parameters. This simple design requires no additional modifications and enables DLRM with CAFE to use checkpoints for training and inference. When training resumes with checkpoints, parameters and states are reloaded simultaneously.

Memory Management: We place the whole HotSketch structure on CPU, since it is not compute-intensive. Built upon PyTorch operators, CAFE's embedding module can run on any accelerators (including CPU, GPU) where PyTorch is supported.

5 EXPERIMENTAL RESULTS

In this section, we conduct experiments on four widely used DLRM datasets and compare our CAFE with existing embedding memory compression methods. We experimentally show that CAFE satisfies all three requirements. We also design experiments to reflect the effectiveness of HotSketch.

5.1 Experimental Settings

5.1.1 Models and Datasets. We conduct the experiments on three representative recommendation models DLRM [19] [1], WDL [18], and DCN [42]. These models are popular in both academia and industry. All models follow the architecture discussed in Section 2.1, with slight differences in the neural network part. In DLRM, a cross layer performs dot operations between embeddings, producing cross terms for subsequent fully-connected (FC) layers; in WDL, embeddings are fed into a wide network (1 FC layer) and a deep network (several FC layers), and finally the results are summed together for predictions; in DCN, cross layers multiply the embeddings with their projected vectors, producing element-level cross terms for subsequent FC layers. Since our method is essentially an embedding layer plugin, the conclusions can be generalized to other recommendation models with little effort. We set the configurations of the models as in the original paper.

We train on three large-scale datasets Avazu [54], Criteo [55], KDD12 [56], and an extremely large-scale dataset CriteoTB [57]. Criteo Kaggle Display Advertising Challenge Dataset (Criteo) [55] and Criteo Terabytes Click Logs (CriteoTB) [57] contain 7 and 24 days of ads click-through rate (CTR) prediction data respectively, which are adopted in MLPerf [58]. Each data sample has 13 numerical fields and 26 categorical fields. For CriteoTB, we set the field's maximum cardinality to 4e7, the same as in the MLPerf configuration. Avazu Click-Through Rate Prediction Dataset (Avazu) [54] and KDD Cup 2012, Track 2 (KDD12) [56] are another two widely-used CTR datasets. They have no numerical field. Avazu contains 10 days of CTR data with 22 categorical fields. KDD12 has no temporal information, and has 11 categorical fields. For each dataset, we use the appropriate embedding dimension based on the benchmarks [19, 59] or our experiments on the uncompressed models. The statistics of the datasets are listed in Table 2. Since the numerical field is not our focus, we omit it from the table. In Section 5.5,

[1] In this section, we use the term "DLRM" to refer to this specific model, rather than the abbreviation of general "Deep Learning Recommendation Model".

we construct a new dataset with a more significant shift in data distribution to further validate CAFE's ability to adapt to changes in data distribution.

Table 2: Overview of the datasets.

Dataset	#Samples	#Features	#Fields	Dim	#Param
Avazu	40,428,967	9,449,445	22	16	150M
Criteo	45,840,617	33,762,577	26	16	540M
KDD12	149,639,105	54,689,798	11	64	3.5B
CriteoTB	4,373,472,329	204,184,588	26	128	26B

5.1.2 Baselines. We compare CAFE with Hash Embedding [37], Q-R Trick [38], and AdaEmbed [40]. Hash embedding is a simple baseline using only one hash function, providing a lower bound for all compression methods. Q-R Trick is an improved hash-based method, using multiple hash functions and complementary embedding tables to reduce the overall collisions. AdaEmbed is an adaptive method, recording all features' importance scores and dynamically allocates embedding vectors only for important features. We also compare with uncompressed embedding tables. In Section 5.2.4, we compare CAFE with a column compression method MDE [33]. If not specified, the hyperparameters of the baselines are the same as in the original paper or code.

5.1.3 Hardware Environment. We conduct all experiments on NVIDIA RTX TITAN 24 GB GPU cards. Since we focus on embedding compression with large compression ratios, we do not incur distributed training or inference.

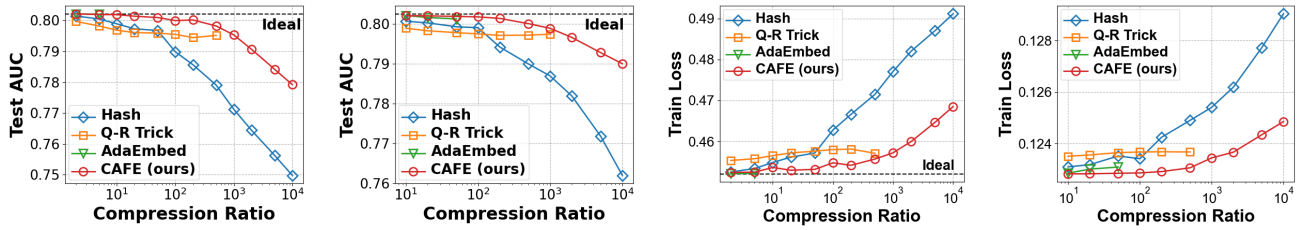
5.1.4 Metrics. We employ training loss and testing AUC (area under the ROC curve) to measure model quality. Specifically, we use the data samples except the last day as the training set, and the data samples of the last day as the testing set. We use the testing AUC on the last day as the metric for offline training, and the average loss during training as the metric for online training. We train one epoch on the training set in chronological order, which is common in industry. Since KDD12 has no temporal information, we randomly shuffle the data and select 90% for training and the rest for testing. For memory usage, besides embedding tables, we also consider the memory of additional structures to achieve a fair judgment on memory efficiency. We use throughput to measure the speed of each method.

5.2 End-to-end Comparison

In this section, we compare CAFE with baseline methods in an end-to-end manner. For large-scale datasets, we train with compression ratios ranging from 2× to 10000×, while for the CriteoTB dataset, we train with compression ratios ranging from 10× to 10000×, ensuring the model fits in the memory.

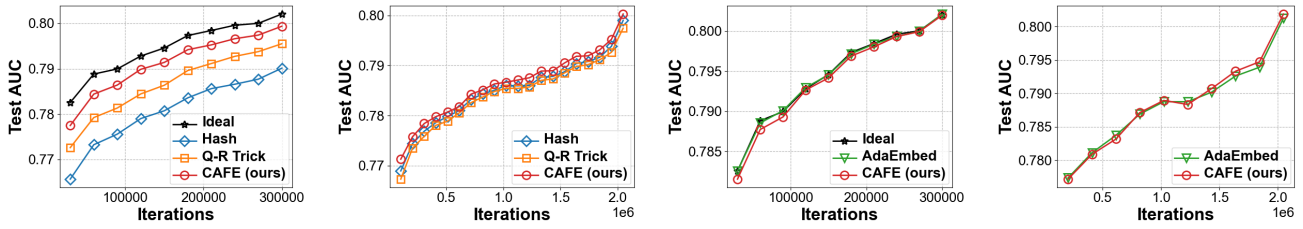
5.2.1 Metrics v.s. compression ratios. We conduct the main experiments on DLRM. The testing AUC and the training loss of Criteo and CriteoTB under different compression ratios are plotted in Figure 8, representing the performance of offline and online training respectively. For KDD12, we only plot the testing AUC in Figure 10(a) since it does not contain temporal information for online training. For Avazu, given the significant changes in distributions between days as shown in Figure 2, we focus on the online training performance and plot the training loss in Figure 10(b). Only CAFE and Hash can compress the embedding tables to extreme 10000× compression ratio, while Q-R Trick can only compress to around 500× due to its complementary index design, and AdaEmbed can only compress to 5× in Avazu and Criteo with dimension 16, 20× in KDD12 with dimension 64, and 50× in CriteoTB with dimension 128. Compared to Hash and Q-R Trick, CAFE is always closer to ideal result that uses uncompressed embedding tables, showing excellent memory efficiency. When varying the compression ratio, on Criteo dataset CAFE improves the testing AUC by 1.79% and 0.55% compared to Hash and Q-R Trick respectively on average; on CriteoTB dataset the improvement is 1.304% and 0.427%; on KDD12 dataset the improvement is 1.86% and 3.80%. CAFE also reduces the training loss by 2.31%, 0.72% on Criteo dataset, 1.35%, 0.59% on CriteoTB dataset, and 3.34%, 0.76% on Avazu dataset compared to Hash and Q-R Trick, exhibiting better performance for both offline and online training. The training loss of Hash fluctuates with the increase of CR on KDD12, which may be due to the instability of the Hash method and a certain degree of randomness in its embedding sharing. The improvement of CAFE over Hash is greater with larger compression ratio. Compared to Hash, at 10000× compression ratio, CAFE improves 3.92%, 3.68%, and 5.16% testing AUC on Criteo, CriteoTB, KDD12; CAFE reduces 4.61%, 3.24%, and 11.21% training loss on Criteo, CriteoTB, Avazu. Compared to AdaEmbed, CAFE reaches nearly the same testing AUC and training loss on Criteo dataset, achieves an increase of 0.04% testing AUC and a decrease of 0.12% training loss on CriteoTB dataset, achieves an increase of 0.82% testing AUC on KDD12 dataset and a decrease of 0.83% training loss on Avazu dataset. AdaEmbed can distinguish hot features with no errors, but it uses much memory for storing importance information of all features, with less memory for embedding vectors compared to CAFE, leading to comparable results at small compression ratios.

5.2.2 Metrics v.s. iterations. We check the convergence process of different methods. Figure 9 shows the metrics on Criteo and CriteoTB throughout iterations during training. Figure 10(c) shows the training loss on Avazu throughout iterations. We do not plot uncompressed embeddings trained on CriteoTB because the model cannot be held in our limited memory space. In Figure 9(a)-9(d), the testing AUC curves tend to increase because the model continues to learn during training and the data distribution gradually approaches the distribution of the last day testing data. CAFE has consistently better AUC during training compared to Hash and Q-R Trick. However, CAFE does not show better performance at the beginning of training compared to AdaEmbed, mainly because CAFE has a cold-start process to populate HotSketch, where all features are initially non-hot features. As training progresses, CAFE gradually achieves an AUC comparable to or better than AdaEmbed. In Figure 9(e)-9(h), and 10(c), the training loss curves fluctuate due to changes in data distributions. CAFE always has a closer training loss to ideal result than Hash and Q-R Trick on Criteo and Avazu datasets, showing better online training ability. The training curves of CAFE and AdaEmbed roughly coincide, since they are both designed for online training. The CriteoTB dataset is large enough to adequately train various methods, resulting in the loss curves of different methods being indistinguishable.

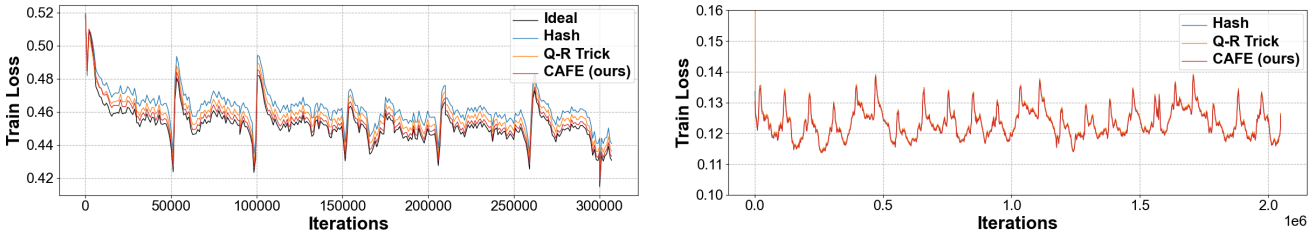


(a) Test AUC v.s. CR on Criteo. (b) Test AUC v.s. CR on CriteoTB. (c) Train loss v.s. CR on Criteo. (d) Train loss v.s. CR on CriteoTB.

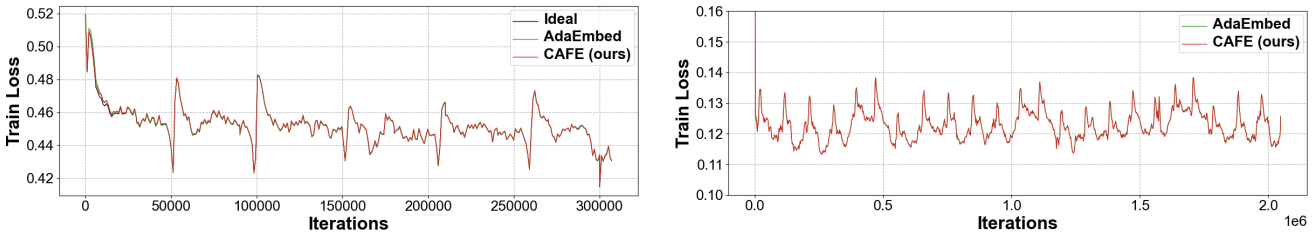
Figure 8: Metrics v.s. compression ratios.



(a) AUC v.s. iterations on Criteo (100×). (b) AUC v.s. iterations on CriteoTB (100×). (c) AUC v.s. iterations on Criteo (5×). (d) AUC v.s. iterations on CriteoTB (50×).

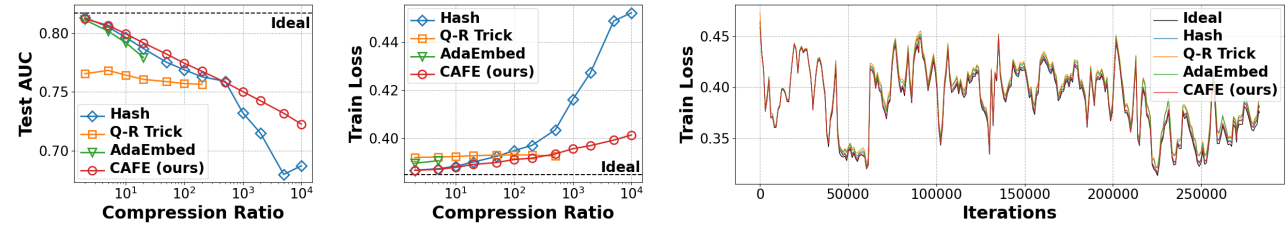


(e) Train loss v.s. iterations on Criteo (100×). (f) Train loss v.s. iterations on CriteoTB (100×).



(g) Train loss v.s. iterations on Criteo (5×). (h) Train loss v.s. iterations on CriteoTB (50×).

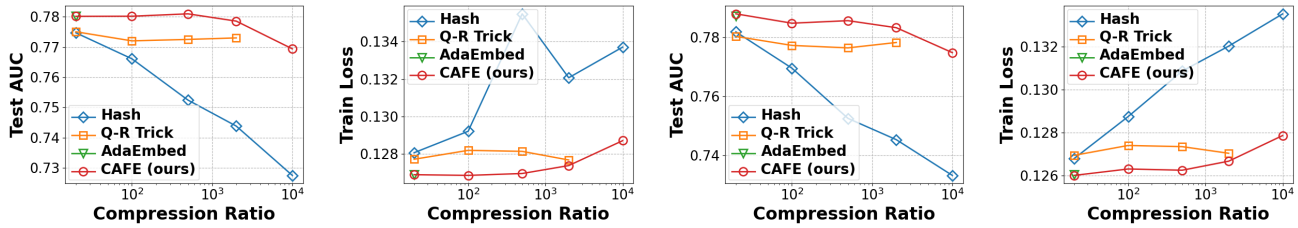
Figure 9: Metrics v.s. iterations.



(a) Test AUC v.s. CR on KDD12. (b) Train loss v.s. CR on Avazu. (c) Train loss v.s. iterations on Avazu (5×).

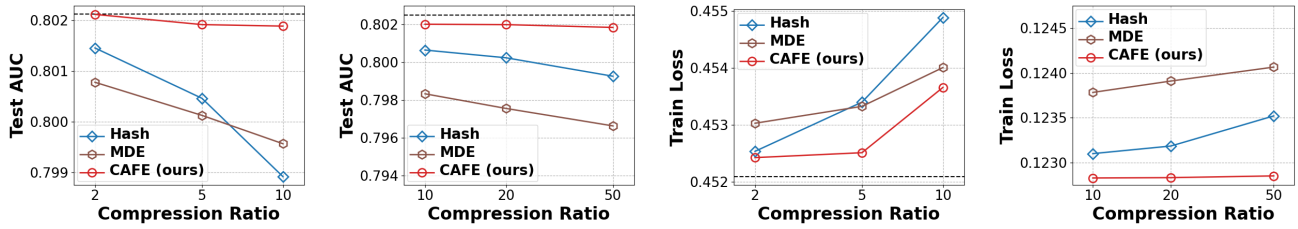
Figure 10: Performance on KDD12 and Avazu.

5.2.3 Experiments on WDL and DCN. We use another two models, WDL [18] and DCN [42], to experiment on the extremely large-scale dataset CriteoTB. The results are shown in Figure 11. Similar to DLRM, CAFE consistently outperforms Hash and Q-R Trick at different compression ratios in both testing AUC and training loss. AdaEmbed is the most advanced compression method for small compression ratios, and CAFE achieves comparable performance to AdaEmbed. The training loss of Hash is not stable in WDL, possibly due to the instability of the Hash method itself and a certain degree of randomness in its embedding sharing.



(a) WDL, Test AUC v.s. CR. (b) WDL, Train loss v.s. CR. (c) DCN, Test AUC v.s. CR. (d) DCN, Train loss v.s. CR.

Figure 11: WDL and DCN performance on CriteoTB.



(a) Test AUC v.s. CR on Criteo. (b) Test AUC v.s. CR on CriteoTB. (c) Train loss v.s. CR on Criteo. (d) Train loss v.s. CR on CriteoTB.

Figure 12: Comparison with MDE.

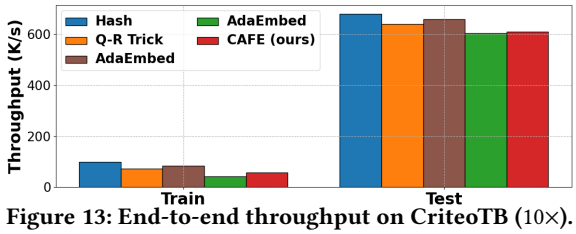
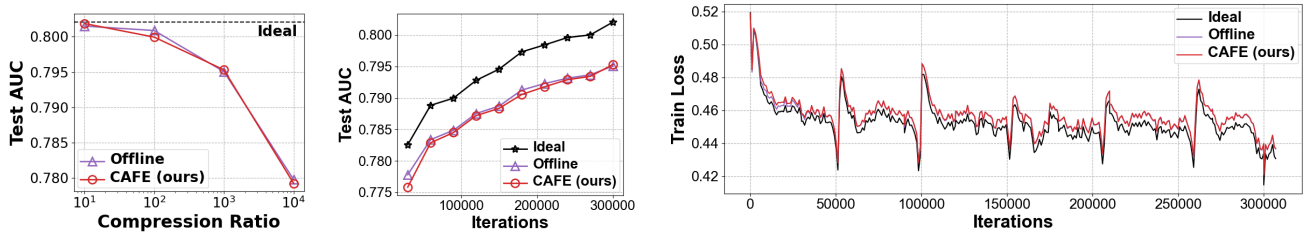


Figure 13: End-to-end throughput on CriteoTB (10×).

5.2.4 Comparison with Column Compression. We also compare CAFE with MDE [33], a method that compresses columns of embedding tables instead of rows as in CAFE and other baselines. It introduces frequency information to allocate different embedding dimensions for different features, and then uses a trainable matrix to project the raw embeddings to the same final dimension. Since MDE does not compress the rows, and each feature needs to be assigned at least one dimension, the overall compression ratio is limited by the original embedding dimension. We plot the results in Figure 12. We also include a simple row compression baseline Hash for comparison. MDE's performance is comparable to Hash on Criteo, but it drops dramatically on CriteoTB. To reduce the number of projection matrices, MDE simply uses the feature cardinality of the field to derive the frequency instead of using the actual frequency, which does not effectively utilize important features. It also significantly reduces the rank of the embedding matrix at large compression ratios, causing the embedding to lose semantic information. According to the experimental results, CAFE always outperforms MDE.

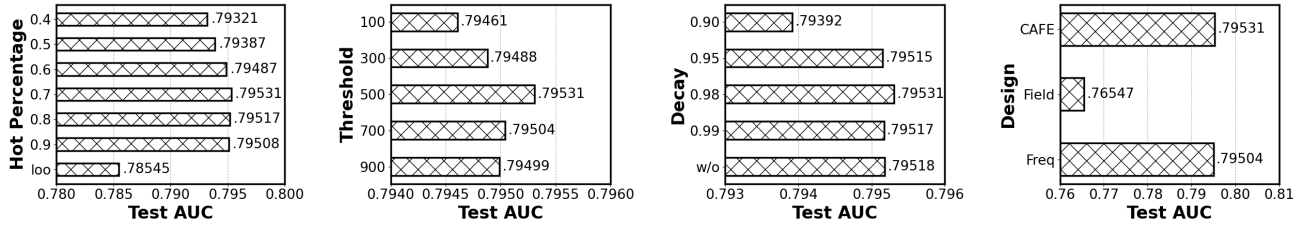
5.2.5 Throughput. We test the throughput of each method in Figure 13. The experiments is conducted on CriteoTB dataset with a compression ratio of 10×. We use 2048 batch size for training and 16384 batch size for inference, which is common in real applications. As the data loading time and the DNN computing time is the same across different methods, the difference lies in the operations of the embedding layer. Hash requires only an additional modulo operation compared to uncompressed embedding operations, and is therefore the fastest method in both training and inference. Q-R Trick also has large throughput, because it only additionally introduces hash processes and the aggregation of embedding vectors. Although MDE introduces matrix multiplication, it requires fewer memory accesses to obtain the embedding parameters, resulting in high throughput. AdaEmbed and CAFE incur reallocation or migration of embeddings, which are inevitable for dynamic adjustments, leading to lower throughputs. AdaEmbed regularly samples thousands of data to determine whether to reallocate, which introduces a large time overhead. In contrast, CAFE determines the migraion in HotSketch with negligible time overhead. Compared to AdaEmbed, CAFE has 33.97% higher training throughput and 1.20% higher inference throughput. Through the further experimental results in Section 5.6, we can see that HotSketch's $O(1)$ operation time only accounts for a small fraction of the overall time.

5.2.6 Comparison with offline separation. We also compare CAFE with an offline feature separation version on Criteo dataset. The offline separation version collects all data and makes statistics, separates hot and non-hot features according to frequency instead of gradient norms, and assigns embedding tables respectively. It uses the same embedding memory as in CAFE for hot and non-hot features. As shown in Figure 14(a), two versions achieve nearly the same testing AUC under several compression ratios. Compared to CAFE, the offline version has no errors in distinguishing hot features, but it can only use frequency, resulting in comparable performance. Figure 14(b) and Figure 14(c) show the testing AUC and the training loss throughout the training process at 1000× compression ratio. At the beginning of training, the offline version has better testing AUC and training loss, because CAFE has a cold-start process to fill in the slots. When the training process becomes stable, the two training loss curves almost completely coincide. The offline version, however, cannot be used in practical applications. First, it cannot adapt to online training, where the frequency information is unknown without recording. Second, in offline training, memory storage and additional data traversal process are required for statistics, causing much overhead. In contrast, CAFE naturally



(a) Test AUC v.s. CR. (b) Test AUC v.s. iterations (1000×). (c) Train loss v.s. iterations (1000×).

Figure 14: CAFE v.s. offline feature separation on Criteo dataset.



(a) Memory for hot features. (b) Threshold of hot features. (c) Decay of scores. (d) Design details.

Figure 15: Experiments of configuration sensitivity on Criteo dataset (1000×).

supports online and offline training without storing all importance information, so it can be directly applied in the industry.

5.3 Configuration Sensitivity

In this section, we study the impact of configurations in CAFE. We test different configurations on the Criteo dataset with a fixed compression ratio of 1000×, as shown in Figure 15.

Memory for hot features. Given a limited memory constraint, we need to split the memory into three parts: sketch structure, hot features, and non-hot features. We define the term "hot percentage" as the percentage of memory occupied by sketch structure and hot features, while the rest is used for non-hot features. Since HotSketch stores 4 times the slots of the number of hot features, with each slot 3 attributes, the ratio of memory usage between HotSketch and d dimension exclusive embeddings is $12 : d$. In the Criteo dataset, the dimension is set to 16, so HotSketch occupies 3/7 of memory in hot percentage. Figure 15(a) shows the testing AUC under different hot percentages, where “loo” means “leave-one-out”, leaving only one embedding for non-hot features. A small hot percentage has low memory overhead of HotSketch, and allocate more memory for non-hot features, while a large hot percentage allocate more memory for hot features. As hot percentage gradually increases from 0.4 to 1, the testing AUC first rises then drops. When the hot percentage is small, enlarging hot percentage enables more hot features, contributing to model quality; when the hot percentage reaches 0.7, CAFE reaches the best testing AUC; when the hot percentage exceeds 0.7, HotSketch brings much overhead, and collisions of non-hot features increase dramatically, making the testing AUC drop. At the extreme case "leave-one-out", all the non-hot features share only one embedding, leading to very bad model performance. In practice, we find that setting hot percentage to around 0.7 is good enough for nearly all compression ratios.

Threshold of hot features. Hot features are distinguished in HotSketch if their importance scores exceed the threshold. We test different thresholds, and the experimental results are shown in Figure 15(b). The testing AUC is bad when the threshold is set too high or too low. If the threshold is set too high, the memory space allocated for hot features cannot be saturated, resulting in waste of memory and more non-hot features sharing hash embeddings. If the threshold is set too low, the entry and exit of features will be too frequent, leading to unstable training process. When threshold is set to 500, CAFE reaches the best model AUC.

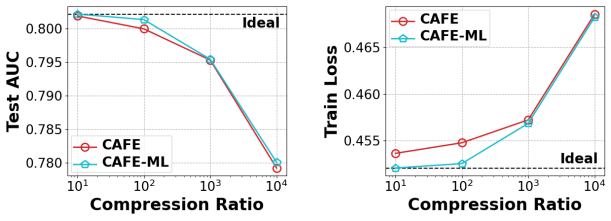
Decay of scores. The decay coefficient in HotSketch determines the exit of features. All the importance scores in HotSketch, after a certain number of iterations, will be multiplied the decay coefficient to adapt to temporal variation of data distribution. We test different decay coefficients in Figure 15(c). In general, smaller the coefficient, easier the hot features to dropped out as non-hot features. In experiments, we find that 0.98 is a proper value for decay coefficient in Criteo dataset, while smaller or larger decay coefficient both have poor performance. When the decay coefficient is too small, hot features cannot stay long in HotSketch, makes HotSketch not saturated and hot features mis-classified to non-hot features. When the decay coefficient is too large, features continuously occupy slots in HotSketch, even if they are no longer hot features.

Other design details. We experiment on other design details of HotSketch. Currently we maintain only one exclusive embedding table for all fields, instead of maintaining one embedding table per field. This design makes hot features more flexible, distributed among fields only according to importance scores rather than cardinality. Figure 15(d) shows that maintaining only one exclusive embedding table leads to a substantial increase in model AUC. We also check the effect of using frequency information as importance score, with a worse testing AUC than gradient norm. Although frequency is a good indicator of feature importance, it has been proved theoretically and experimentally that gradient norm is better.

5.4 Multi-level Hash Embedding

In this section, we study the effect of multi-level hash embedding. The experimental results are shown in Figure 16, where CAFE-ML means CAFE combined with multi-level hash embedding. Under different compression ratios, CAFE-ML always performs better

than CAFE, achieving 0.08% better testing AUC and reducing 0.25% training loss. CAFE-ML performs especially well with smaller compression ratios, causing nearly no degradation at 100× compression ratio. This is because CAFE-ML allocates more memory for multi-level hash embedding tables at small compression ratios, making the representation of medium features more precise.



(a) Test AUC v.s. CR. (b) Train loss v.s. CR.

Figure 16: Multi-level hash embedding on Criteo dataset.

5.5 Performance on Processed Dataset

In this section, we construct a new dataset with a more significant shift in data distribution to further validate CAFE's ability to adapt to changes in data distribution. Keeping the testing data unchanged, we select the training data of days 1,4,7,...,22 from CriteoTB to form CriteoTB-1/3 dataset. As shown in Figure 2, generally the greater the number of days between, the greater the difference between feature distributions. Therefore, CriteoTB-1/3 has a more significant shift in data distribution. The results are shown in Figure 17. Although all methods exhibit slight performance degradation compared to CriteoTB, CAFE and AdaEmbed can adapt to changing data distributions and achieve relatively good results. Figure 17(c) shows that CAFE and AdaEmbed have almost the same training loss throughout the training process. However, Figure 17(a) and 17(b) indicate that CAFE actually outperforms AdaEmbed with a slight improvement, demonstrating stronger online training capabilities.

5.6 HotSketch Performance

Impact of the number of slots per bucket (Figure 18(a), 18(b)): We record the recall and the throughput of HotSketch with different number of slots per bucket. The experiments use the number of hot features on Criteo dataset (1000×) as k. In Figure 18(a), recall generally increases as memory becomes larger. According to Corollary 3.5, the best number of slots per bucket locates at 11 to 21 given a Zipf distribution of parameter 1.05 to 1.1. Therefore, $c = 8$ and $c = 16$ exhibit a better recall than $c = 4$ and $c = 32$. The throughput of serialized Insert (write) and Query (read) shown in Figure 18(b) is on the order of $1e7$, greater than that of DLRM. Considering that we can parallelize operations in practice, the sketch time is only a small fraction in training and inference. Throughput drops as the number of slots increases, because more time is spent doing comparisons within buckets. Trading-off recall and throughput, we adopt 4 slots per bucket in our implementation, as we find it to be good enough for model quality.

Finding real-time top-k features (Figure 18(c),18(d)): We conduct experiments to evaluate the performance of HotSketch on finding two types of real-time hot features in online training: the up-to-date top-k features, and the top-k features in previous time window. These two types of top-k features change with data distribution during the online training process, and thus can effectively reflect HotSketch's capability to adapt dynamic workloads. The experiments are conducted on Criteo using 6 days of online training data, with a sliding window size of 0.5 day. Figure 18(c) and 18(d) show the real-time Recall Rate of HotSketch during online training under different compression ratios. HotSketch always achieves >90% Recall Rate on finding these two types of top-k features, meaning that it can well catch up with the changing data distribution.

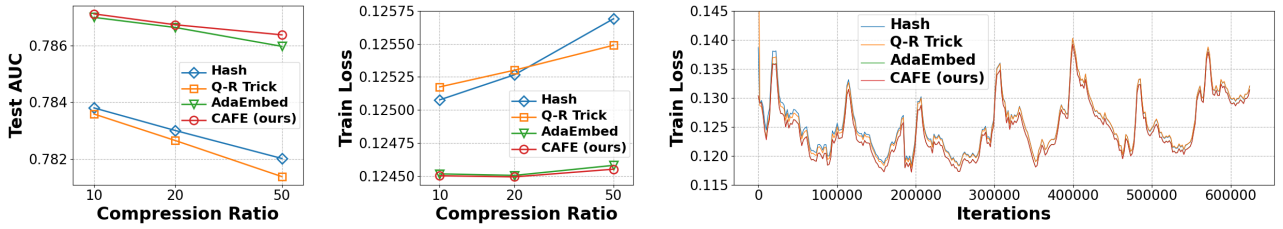
6 RELATED WORK

6.1 Embedding Compression

Numerous compression techniques have been proposed for embedding tables, which can be broadly divided into two categories: row compression and column compression [60]. Row compression methods, including hash-based methods, adaptive methods, and CAFE, reduce the number of rows in embedding tables. Column compression methods, including quantization, pruning, and dimension reduction, compress each unique feature's representation, thereby reducing the number of columns (or the number of bits) in embedding tables. Since two categories are primarily orthogonal, methods of different categories can be further combined in DLRMs.

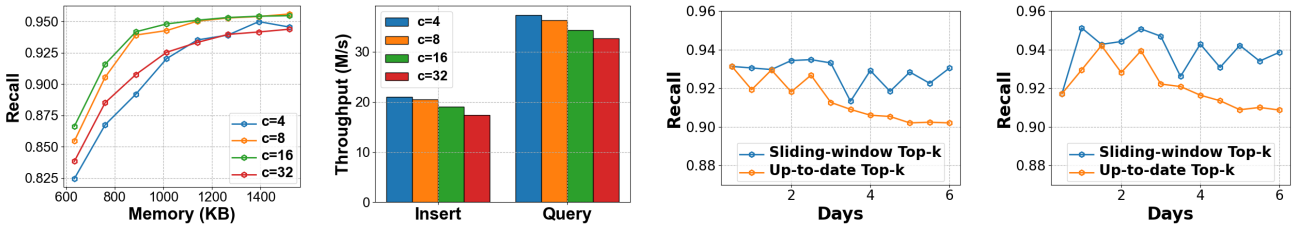
Row compression methods: These methods aim to reduce the number of rows in embedding tables. Initial attempts to accommodate large numbers of embeddings within a limited memory space came from hash-based methods [37–39], which are widely used in real-world applications. They use simple hash functions to map categorical features onto a limited number of embeddings, resulting in different features sharing the same embedding vector in the event of hash collisions. However, hash-based methods do not provide theoretical bounds, which can lead to significant degradation in model quality. AdaEmbed [40] is an adaptive method that identifies and records important features. It dynamically reallocates embeddings for important features during online training, and achieves good model accuracy over time. However, its compression ratio is constrained by the storage of importance information, which scales linearly with the number of features. AdaEmbed's sampling and migration strategy also incurs much latency in online training.

Column compression methods: Methods of this category aim to compress the representation for each unique feature, thereby reducing the number of columns (or the number of bits) in embedding tables. They borrow techniques from traditional deep learning compression , such as quantization [61, 62], pruning [35, 63], and dimension reduction [33, 34, 64]. Except for simple quantization and rule-based dimension reduction, most of these methods incorporate learnable structures to implicitly capture the importance of features, achieving similar or even better model accuracy compared to an uncompressed model. Nevertheless, they are unable to compress the embedding tables to small memory constraints during training. Specifically, quantization has a fixed compression ratio according to the data type; for example, if INT4 is used for compression, the compression ratio is fixed at 8× compared to FLOAT32. Generally, pruning and dimension reduction compress the embedding tables only at inference time, requiring additional memory to store extra structures during training. They are seldom used in industry, as the memory bottleneck during training is more severe due to activations and optimizer states. Most of these methods can only support offline training because they require collected data for multi-stage training, including pre-training, finetuning, and re-training.



(a) Test AUC v.s. CR on CriteoTB-1/3. (b) Train loss v.s. CR on CriteoTB-1/3. (c) Train loss v.s. iterations on CriteoTB-1/3 (50×).

Figure 17: Experiments on CriteoTB-1/3.



(a) Recall. (b) Throughput. (c) Recall v.s. days under 100× CR. (d) Recall v.s. days under 1000× CR.

Figure 18: Experiments on HotSketch.

6.2 Sketching Algorithm

Sketch is an excellent probabilistic data structure that can approximately record the statistics of data streams by maintaining a summary. Thanks to their small memory overhead and fast processing speed, sketches are widely applied in the realm of streaming data mining [65], database [66–68], and network measurement and management [69, 70] to perform various tasks, such as frequency estimation [65, 71, 72], finding top-k frequent items [51, 73–75], and mining special patterns in streaming data [76]. Existing sketches can be classified into two categories: counter-based sketches and KV-based sketches.

Counter-based sketches: Typical counter-based sketches include CM [65], CU [71], Count [77], ASketch [78], and more [72, 79–81]. The data structures of these sketches usually consist of multiple arrays, each containing many counters. Each array is associated with one hash function that maps items into a specific counter in it. For example, the most popular CM sketch [65] comprises d counter arrays $C_1, \cdots, C_d$. For each incoming item e, it is hashed into d counters in the CM sketch $C_1[h_1(e)], \cdots, C_d[h_d(e)]$ with each of the d counters incremented by one. To query item e, CM sketch returns the minimum counter among $C_1[h_1(e)], \cdots, C_d[h_d(e)]$. However, the CM sketch has overestimated errors due to hash collisions. Other sketches propose various strategies to reduce this error. For instance, CU sketch [71] only increments the minimum counter among $C_1[h_1(e)], \cdots, C_d[h_d(e)]$, and Count sketch [77] adds $s(e) \in \{+1, -1\}$ to each mapped counter to achieve unbiased estimation. Despite these improvements, existing counter-based sketches are not memory efficient for finding top-k items. They do not distinguish between frequent and infrequent items, where infrequent items are useless for reporting top-k items, and recording infrequent items only increases the error of frequent items. Moreover, they need multiple memory accesses per insertion, resulting in unsatisfactory insertion speed.

KV-based sketches: Common key-value-based sketches include Space-Saving [51], Unbaised Space-Saving [41], Lossy Counting [82], HeavyGuardian [74], and more [69, 70, 73]. These sketches maintain the KV pairs of frequent items in their data structures. For instance, Space-Saving [51] and Unbiased Space-Saving [41] use a data structure called Stream-Summary to record frequent items, which is essentially a doubly-linked list of fixed size, indexed by a hash table. When Stream-Summary is full and an unrecorded item arrives, Space-Saving replaces the least frequent item with the incoming one. Based on Space-Saving, Unbiased Space-Saving [41] replaces the least frequent item with a certain probability, so as to achieve unbiased estimation. Unfortunately, Space-Saving and Unbiased Space-Saving are not memory and time efficient because of the extra hash table and complex pointer operations. Another type of KV-based sketches, such as HeavyGuardian [74] and WavingSketch [73], uses a bucket array data structure, where each bucket stores multiple KV pairs. These sketches provide satisfactory accuracy for reporting top-k items and only require one memory access per insertion, ensuring fast insertion speed.

7 CONCLUSION

In this paper, we propose CAFE, a compact, adaptive, and fast embedding compression method that fulfills three essential design requirements: memory efficiency, low latency, and adaptability to dynamic data distribution. We introduce a light-weight sketch structure, HotSketch, to identify and record the importance scores of features. It incurs negligible time overhead, and its memory consumption is significantly lower than the original embedding tables. By assigning exclusive embeddings to a small set of important features and shared embeddings to other less important features, we achieve superior model quality within a limited memory constraint. To adapt to dynamic data distribution during online training, we incorporate an embedding migration strategy based on HotSketch. We further optimize CAFE with multi-level hash embedding, creating finer-grained importance groups. Experimental results demonstrate that CAFE outperforms existing methods, with 3.92%, 3.68% higher testing AUC and 4.61%, 3.24% lower training loss at 10000× compression ratio on Criteo Kaggle and CriteoTB datasets, exhibiting superior performance in both offline training and online training. The source codes of CAFE are available at GitHub [1].

REFERENCES

[1] Source codes related to CAFE. https://github.com/HugoZHL/CAFE.

[2] Jie Liu, Wenqian Dong, Dong Li, and Qingqing Zhou. Fauce: Fast and accurate deep ensembles with uncertainty for cardinality estimation. *Proceedings of the VLDB Endowment*, 14(11):1950–1963, 2021.

[3] Suyong Kwon, Woohwan Jung, and Kyuseok Shim. Cardinality estimation of approximate substring queries using deep learning. *Proceedings of the VLDB Endowment*, 15(11):3145–3157, 2022.

[4] Yue Zhao, Gao Cong, Jiachen Shi, and Chunyan Miao. Queryformer: A tree transformer model for query plan representation. *Proceedings of the VLDB Endowment*, 15(8):1658–1670, 2022.

[5] Tianyi Chen, Jun Gao, Hedui Chen, and Yaofeng Tu. LOGER: A learned optimizer towards generating efficient and robust query execution plans. *Proceedings of the VLDB Endowment*, 16(7):1777–1789, 2023.

[6] Hyeonji Kim, Byeong-Hoon So, Wook-Shin Han, and Hongrae Lee. Natural language to SQL: where are we today? *Proceedings of the VLDB Endowment*, 13(10):1737–1750, 2020.

[7] Muhammad Ebraheem, Saravanan Thirumuruganathan, Shafiq R. Joty, Mourad Ouzzani, and Nan Tang. Distributed representations of tuples for entity resolution. *Proceedings of the VLDB Endowment*, 11(11):1454–1467, 2018.

[8] Yue Ding, Yuhe Guo, Wei Lu, Hai-Xiang Li, Meihui Zhang, Hui Li, An-Qun Pan, and Xiaoyong Du. Context-aware semantic type identification for relational attributes. *Journal of Computer Science and Technology*, 38(4):927–946, 2023.

[9] Ruihong Huang, Shaoxu Song, Yunsu Lee, Jungho Park, Soo-Hyung Kim, and Sungmin Yi. Effective and efficient retrieval of structured entities. *Proceedings of the VLDB Endowment*, 13(6):826–839, 2020.

[10] Adrian Kochsiek and Rainer Gemulla. Parallel training of knowledge graph embedding models: A comparison of techniques. *Proceedings of the VLDB Endowment*, 15(3):633–645, 2021.

[11] Renchi Yang, Jieming Shi, Xiaokui Xiao, Yin Yang, Juncheng Liu, and Sourav S. Bhowmick. Scaling attributed network embedding to massive graphs. *Proceedings of the VLDB Endowment*, 14(1):37–49, 2020.

[12] Xupeng Miao, Yining Shi, Hailin Zhang, Xin Zhang, Xiaonan Nie, Zhi Yang, and Bin Cui. HET-GMP: A graph-based system approach to scaling large embedding model training. In *Proceedings of the International Conference on Management of Data (SIGMOD)*, 2022.

[13] Michael Chui, James Manyika, Mehdi Miremadi, Nicolaus Henke, Rita Chung, Pieter Nel, and Sankalp Malhotra. Notes from the ai frontier: Insights from hundreds of use cases. *McKinsey Global Institute*, 2, 2018.

[14] Corinna Underwood. Use cases of recommendation systems in business–current applications and methods. *Emerj*, 2019.

[15] Xing Xie, Jianxun Lian, Zheng Liu, Xiting Wang, Fangzhao Wu, Hongwei Wang, and Zhongxia Chen. Personalized recommendation systems: Five hot research topics you must know. *Microsoft Research Lab-Asia*, 2018.

[16] Udit Gupta, Carole-Jean Wu, Xiaodong Wang, Maxim Naumov, Brandon Reagen, David Brooks, Bradford Cottel, Kim M. Hazelwood, Mark Hempstead, Bill Jia, Hsien-Hsin S. Lee, Andrey Malevich, Dheevatsa Mudigere, Mikhail Smelyanskiy, Liang Xiong, and Xuan Zhang. The architectural implications of facebook's dnn-based personalized recommendation. In *IEEE International Symposium on High Performance Computer Architecture (HPCA)*, 2020.

[17] Maxim Naumov, John Kim, Dheevatsa Mudigere, Srinivas Sridharan, Xiaodong Wang, Whitney Zhao, Serhat Yilmaz, Changkyu Kim, Hector Yuen, Mustafa Ozdal, Krishnakumar Nair, Isabel Gao, Bor-Yiing Su, Jiyan Yang, and Mikhail Smelyanskiy. Deep learning training in facebook data centers: Design of scale-up and scale-out systems. *CoRR*, abs/2003.09518, 2020.

[18] Heng-Tze Cheng, Levent Koc, Jeremiah Harmsen, Tal Shaked, Tushar Chandra, Hrishi Aradhye, Glen Anderson, Greg Corrado, Wei Chai, Mustafa Ispir, Rohan Anil, Zakaria Haque, Lichan Hong, Vihan Jain, Xiaobing Liu, and Hemal Shah. Wide & deep learning for recommender systems. In *Proceedings of the 1st Workshop on Deep Learning for Recommender Systems (DLRS@RecSys)*, 2016.

[19] Maxim Naumov, Dheevatsa Mudigere, Hao-Jun Michael Shi, Jianyu Huang, Narayanan Sundaraman, Jongsoo Park, Xiaodong Wang, Udit Gupta, Carole-Jean Wu, Alisson G. Azzolini, Dmytro Dzhulgakov, Andrey Mallevich, Ilia Cherniavskii, Yinghai Lu, Raghuraman Krishnamoorthi, Ansha Yu, Volodymyr Kondratenko, Stephanie Pereira, Xianjie Chen, Wenlin Chen, Vijay Rao, Bill Jia, Liang Xiong, and Misha Smelyanskiy. Deep learning recommendation model for personalization and recommendation systems. *CoRR*, abs/1906.00091, 2019.

[20] Pengyang Shao, Le Wu, Lei Chen, Kun Zhang, and Meng Wang. Faircf: fairness-aware collaborative filtering. *Science China Information Sciences*, 65(12), 2022.

[21] Zhiyang Yuan, Wenguang Zheng, Peilin Yang, Qingbo Hao, and Yingyuan Xiao. Evolving interest with feature co-action network for CTR prediction. *Data Science and Engineering*, 8(4):344–356, 2023.

[22] Xiangfu Meng, Hongjin Huo, Xiaoyan Zhang, Wanchun Wang, and Jinxia Zhu. A survey of personalized news recommendation. *Data Science and Engineering*, 8(4):396–416, 2023.

[23] Teng-Yue Han, Pengfei Wang, and Shaozhang Niu. Multimodal interactive network for sequential recommendation. *Journal of Computer Science and Technology*, 38(4):911–926, 2023.

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

[25] Dheevatsa Mudigere, Yuchen Hao, Jianyu Huang, Zhihao Jia, Andrew Tulloch, Srinivas Sridharan, Xing Liu, Mustafa Ozdal, Jade Nie, Jongsoo Park, Liang Luo, Jie Amy Yang, Leon Gao, Dmytro Ivchenko, Aarti Basant, Yuxi Hu, Jiyan Yang, Ehsan K. Ardestani, Xiaodong Wang, Rakesh Komuravelli, Ching-Hsiang Chu, Serhat Yilmaz, Huayu Li, Jiyuan Qian, Zhuobo Feng, Yinbin Ma, Junjie Yang, Ellie Wen, Hong Li, Lin Yang, Chonglin Sun, Whitney Zhao, Dimitry Melts, Krishna Dhulipala, K. R. Kishore, Tyler Graf, Assaf Eisenman, Kiran Kumar Matam, Adi Gangidi, Guoqiang Jerry Chen, Manoj Krishnan, Avinash Nayak, Krishnakumar Nair, Bharath Muthiah, Mahmoud khorashadi, Pallab Bhattacharya, Petr Lapukhov, Maxim Naumov, Ajit Mathews, Lin Qiao, Mikhail Smelyanskiy, Bill Jia, and Vijay Rao. Software-hardware co-design for fast and scalable training of deep learning recommendation models. In *Proceedings of the 49th Annual International Symposium on Computer Architecture (ISCA)*, 2022.

[26] Xupeng Miao, Hailin Zhang, Yining Shi, Xiaonan Nie, Zhi Yang, Yangyu Tao, and Bin Cui. HET: scaling out huge embedding model training via cache-enabled distributed framework. *Proceedings of the VLDB Endowment*, 15(2):312–320, 2022.

[27] Zehuan Wang, Yingcan Wei, Minseok Lee, Matthias Langer, Fan Yu, Jie Liu, Shijie Liu, Daniel G. Abel, Xu Guo, Jianbing Dong, Ji Shi, and Kunlun Li. Merlin hugectr: Gpu-accelerated recommender system training and inference. In *Proceedings of the 16th ACM Conference on Recommender Systems (RecSys)*, 2022.

[28] Niketan Pansare, Jay Katukuri, Aditya Arora, Frank Cipollone, Riyaaz Shaik, Noyan Tokgozoglu, and Chandru Venkataraman. Learning compressed embeddings for on-device inference. In *Proceedings of Machine Learning and Systems (MLSys)*, 2022.

[29] Huifeng Guo, Ruiming Tang, Yunming Ye, Zhenguo Li, and Xiuqiang He. Deepfm: A factorization-machine based neural network for CTR prediction. In *Proceedings of the 26th International Joint Conference on Artificial Intelligence (IJCAI)*, 2017.

[30] Weijie Zhao, Deping Xie, Ronglai Jia, Yulei Qian, Ruiquan Ding, Mingming Sun, and Ping Li. Distributed hierarchical GPU parameter server for massive scale deep learning ads systems. In *Proceedings of Machine Learning and Systems (MLSys)*, 2020.

[31] Udit Gupta, Samuel Hsia, Vikram Saraph, Xiaodong Wang, Brandon Reagen, Gu-Yeon Wei, Hsien-Hsin S. Lee, David Brooks, and Carole-Jean Wu. Deeprecsys: A system for optimizing end-to-end at-scale neural recommendation inference. In *Proceedings of the 47th Annual International Symposium on Computer Architecture (ISCA)*, 2020.

[32] Caojin Zhang, Yicun Liu, Yuanpu Xie, Sofia Ira Ktena, Alykhan Tejani, Akshay Gupta, Pranay Kumar Myana, Deepak Dilipkumar, Suvadip Paul, Ikuhiro Ihara, Prasang Upadhyaya, Ferenc Huszar, and Wenzhe Shi. Model size reduction using frequency based double hashing for recommender systems. In *Proceedings of the 14th ACM Conference on Recommender Systems (RecSys)*, 2020.

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

[34] Xiangyu Zhao, Haochen Liu, Hui Liu, Jiliang Tang, Weiwei Guo, Jun Shi, Sida Wang, Huiji Gao, and Bo Long. Autodim: Field-aware embedding dimension searchin recommender systems. In *Proceedings of the Web Conference (WWW)*, 2021.

[35] Shuming Kong, Weiyu Cheng, Yanyan Shen, and Linpeng Huang. Autosrh: An embedding dimensionality search framework for tabular data prediction. *IEEE Transactions on Knowledge and Data Engineering*, 35(7):6673–6686, 2023.

[36] Jia-Dong Zhang and Chi-Yin Chow. Geosoca: Exploiting geographical, social and categorical correlations for point-of-interest recommendations. In *Proceedings of the 38th International ACM SIGIR Conference on Research and Development in Information Retrieval (SIGIR)*, 2015.

[37] Kilian Q. Weinberger, Anirban Dasgupta, John Langford, Alexander J. Smola, and Josh Attenberg. Feature hashing for large scale multitask learning. In *Proceedings of the 26th International Conference on Machine Learning (ICML)*, 2009.

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

[39] Bencheng Yan, Pengjie Wang, Jinquan Liu, Wei Lin, Kuang-Chih Lee, Jian Xu, and Bo Zheng. Binary code based hash embedding for web-scale applications. In *Proceedings of the 30th ACM International Conference on Information & Knowledge Management (CIKM)*, 2021.

[40] Fan Lai, Wei Zhang, Rui Liu, William Tsai, Xiaohan Wei, Yuxi Hu, Sabin Devkota, Jianyu Huang, Jongsoo Park, Xing Liu, Zeliang Chen, Ellie Wen, Paul Rivera, Jie You, Chun-cheng Jason Chen, and Mosharaf Chowdhury. Adaembed: Adaptive embedding for large-scale recommendation models. In *17th USENIX Symposium on Operating Systems Design and Implementation (OSDI)*, 2023.

[41] Daniel Ting. Data sketches for disaggregated subset sum and frequent item estimation. In *Proceedings of the International Conference on Management of Data (SIGMOD)*, 2018.

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

[43] Guorui Zhou, Xiaoqiang Zhu, Chengru Song, Ying Fan, Han Zhu, Xiao Ma, Yanghui Yan, Junqi Jin, Han Li, and Kun Gai. Deep interest network for click-through rate prediction. In *Proceedings of the 24th ACM SIGKDD Conference on Knowledge Discovery & Data Mining (KDD)*, 2018.

[44] Biye Jiang, Chao Deng, Huimin Yi, Zelin Hu, Guorui Zhou, Yang Zheng, Sui Huang, Xinyang Guo, Dongyue Wang, Yue Song, Liqin Zhao, Zhi Wang, Peng Sun, Yu Zhang, Di Zhang, Jinhui Li, Jian Xu, Xiaoqiang Zhu, and Kun Gai. Xdl: an industrial deep learning framework for high-dimensional sparse data. In *Proceedings of the 1st International Workshop on Deep Learning Practice for High-Dimensional Sparse Data*, 2019.

[45] Weijie Zhao, Jingyuan Zhang, Deping Xie, Yulei Qian, Ronglai Jia, and Ping Li. Aibox: CTR prediction model training on a single node. In *Proceedings of the 28th ACM International Conference on Information & Knowledge Management (CIKM)*, 2019.

[46] Minhui Xie, Kai Ren, Youyou Lu, Guangxu Yang, Qingxing Xu, Bihai Wu, Jiazhen Lin, Hongbo Ao, Wanhong Xu, and Jiwu Shu. Kraken: memory-efficient continual learning for large-scale real-time recommendations. In *Proceedings of the International Conference for High Performance Computing, Networking, Storage and Analysis (SC)*, 2020.

[47] Xiangru Lian, Binhang Yuan, Xuefeng Zhu, Yulong Wang, Yongjun He, Honghuan Wu, Lei Sun, Haodong Lyu, Chengjun Liu, Xing Dong, Yiqiao Liao, Mingnan Luo, Congfei Zhang, Jingru Xie, Haonan Li, Lei Chen, Renjie Huang, Jianying Lin, Chengchun Shu, Xuezhong Qiu, Zhishan Liu, Dongying Kong, Lei Yuan, Hai Yu, Sen Yang, Ce Zhang, and Ji Liu. Persia: An open, hybrid system scaling deep learning-based recommenders up to 100 trillion parameters. In *Proceedings of the 28th ACM SIGKDD Conference on Knowledge Discovery & Data Mining (KDD)*, 2022.

[48] Diederik P. Kingma and Jimmy Ba. Adam: A method for stochastic optimization. In *3rd International Conference on Learning Representations (ICLR)*, 2015.

[49] Siddharth Gopal. Adaptive sampling for SGD by exploiting side information. In *Proceedings of the 33nd International Conference on Machine Learning (ICML)*, 2016.

[50] Angelos Katharopoulos and François Fleuret. Not all samples are created equal: Deep learning with importance sampling. In *Proceedings of the 35th International Conference on Machine Learning (ICML)*, 2018.

[51] Ahmed Metwally, Divyakant Agrawal, and Amr El Abbadi. Efficient computation of frequent and top-k elements in data streams. In *International Conference on Database Theory*, 2005.

[52] Zeyuan Allen-Zhu. Natasha: Faster non-convex stochastic optimization via strongly non-convex parameter. In *Proceedings of the 34th International Conference on Machine Learning (ICML)*, 2017.

[53] Xupeng Miao, Xiaonan Nie, Hailin Zhang, Tong Zhao, and Bin Cui. Hetu: a highly efficient automatic parallel distributed deep learning system. *Science China Information Sciences*, 66(1), 2023.

[54] Steve Wang and Will Cukierski. Avazu click-through rate prediction. https://kaggle.com/competitions/avazu-ctr-prediction, 2014.

[55] Criteo Labs. Kaggle display advertising challenge dataset. https://labs.criteo.com/2014/02/kaggle-display-advertising-challenge-dataset/, 2014.

[56] Aden and Yi Wang. Kdd cup 2012, track 2. https://kaggle.com/competitions/kddcup2012-track2, 2012.

[57] Criteo Labs. Download terabyte click logs. https://labs.criteo.com/2013/12/download-terabyte-click-logs/, 2013.

[58] NVIDIA AI platform. Mlperf benchmark. https://mlperf.org, 2020.

[59] Jieming Zhu, Jinyang Liu, Shuai Yang, Qi Zhang, and Xiuqiang He. Open benchmarking for click-through rate prediction. In *Proceedings of the 30th ACM International Conference on Information & Knowledge Management (CIKM)*, 2021.

[60] Hailin Zhang, Penghao Zhao, Xupeng Miao, Yingxia Shao, Zirui Liu, Tong Yang, and Bin Cui. Experimental analysis of large-scale learnable vector storage compression. *CoRR*, abs/2311.15578, 2023.

[61] Zhiqiang Xu, Dong Li, Weijie Zhao, Xing Shen, Tianbo Huang, Xiaoyun Li, and Ping Li. Agile and accurate CTR prediction model training for massive-scale online advertising systems. In *Proceedings of the International Conference on Management of Data (SIGMOD)*, 2021.

[62] Shiwei Li, Huifeng Guo, Lu Hou, Wei Zhang, Xing Tang, Ruiming Tang, Rui Zhang, and Ruixuan Li. Adaptive low-precision training for embeddings in click-through rate prediction. In *Proceedings of the AAAI Conference on Artificial Intelligence (AAAI)*, 2023.

[63] Wei Deng, Junwei Pan, Tian Zhou, Deguang Kong, Aaron Flores, and Guang Lin. 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 (WSDM)*, 2021.

[64] Fuyuan Lyu, Xing Tang, Hong Zhu, Huifeng Guo, Yingxue Zhang, Ruiming Tang, and Xue Liu. Optembed: Learning optimal embedding table for click-through rate prediction. In *Proceedings of the 31st ACM International Conference on Information & Knowledge Management (CIKM)*, 2022.

[65] Graham Cormode and S. Muthukrishnan. An improved data stream summary: the count-min sketch and its applications. *Journal of Algorithms*, 55(1):58–75, 2005.

[66] Yesdaulet Izenov, Asoke Datta, Florin Rusu, and Jun Hyung Shin. COMPASS: online sketch-based query optimization for in-memory databases. In *Proceedings of the International Conference on Management of Data (SIGMOD)*, 2021.

[67] Benwei Shi, Zhuoyue Zhao, Yanqing Peng, Feifei Li, and Jeff M. Phillips. At-the-time and back-in-time persistent sketches. In *Proceedings of the International Conference on Management of Data (SIGMOD)*, 2021.

[68] Monica Chiosa, Thomas Preußer, and Gustavo Alonso. SKT: A one-pass multi-sketch data analytics accelerator. *Proceedings of the VLDB Endowment*, 14(11):2369–2382, 2021.

[69] Tong Yang, Jie Jiang, Peng Liu, Qun Huang, Junzhi Gong, Yang Zhou, Rui Miao, Xiaoming Li, and Steve Uhlig. Elastic sketch: adaptive and fast network-wide measurements. In *Proceedings of the 2018 ACM SIGCOMM Conference*, 2018.

[70] Yinda Zhang, Zaoxing Liu, Ruixin Wang, Tong Yang, Jizhou Li, Ruijie Miao, Peng Liu, Ruwen Zhang, and Junchen Jiang. Cocosketch: high-performance sketch-based measurement over arbitrary partial key query. In *Proceedings of the 2021 ACM SIGCOMM Conference*, 2021.

[71] Cristian Estan and George Varghese. New directions in traffic measurement and accounting. *ACM SIGCOMM Computer Communication Review*, 32(4):323–336, 2002.

[72] Fan Deng and Davood Rafiei. New estimation algorithms for streaming data: Count-min can do more. *Webdocs. Cs. Ualberta. Ca*, 2007.

[73] Jizhou Li, Zikun Li, Yifei Xu, Shiqi Jiang, Tong Yang, Bin Cui, Yafei Dai, and Gong Zhang. Wavingsketch: An unbiased and generic sketch for finding top-k items in data streams. In *Proceedings of the 26th ACM SIGKDD Conference on Knowledge Discovery & Data Mining (KDD)*, 2020.

[74] Tong Yang, Junzhi Gong, Haowei Zhang, Lei Zou, Lei Shi, and Xiaoming Li. Heavyguardian: Separate and guard hot items in data streams. In *Proceedings of the 24th ACM SIGKDD Conference on Knowledge Discovery & Data Mining (KDD)*, 2018.

[75] Ankush Mandal, He Jiang, Anshumali Shrivastava, and Vivek Sarkar. Topkapi: Parallel and fast sketches for finding top-k frequent elements. In *Advances in Neural Information Processing Systems 31 (NeurIPS)*, 2018.

[76] Zirui Liu, Chaozhe Kong, Kaicheng Yang, Tong Yang, Ruijie Miao, Qizhi Chen, Yikai Zhao, Yaofeng Tu, and Bin Cui. Hypercalm sketch: One-pass mining periodic batches in data streams. In *39th IEEE International Conference on Data Engineering (ICDE)*, 2023.

[77] Moses Charikar, Kevin C. Chen, and Martin Farach-Colton. Finding frequent items in data streams. In *Automata, Languages and Programming, 29th International Colloquium (ICALP)*, 2002.

[78] Pratanu Roy, Arijit Khan, and Gustavo Alonso. Augmented sketch: Faster and more accurate stream processing. In *Proceedings of the International Conference on Management of Data (SIGMOD)*, 2016.

[79] Guillaume Pitel and Geoffroy Fouquier. Count-min-log sketch: Approximately counting with approximate counters. In *International Symposium on Web AlGorithms*, 2015.

[80] Yao-Chung Fan and Arbee L. P. Chen. Efficient and robust sensor data aggregation using linear counting sketches. In *22nd IEEE International Symposium on Parallel and Distributed Processing (IPDPS)*, 2008.

[81] Tao Li, Shigang Chen, and Yibei Ling. Per-flow traffic measurement through randomized counter sharing. *IEEE/ACM Transactions on Networking*, 20(5):1622–1634, 2012.

[82] Xenofontas A. Dimitropoulos, Paul Hurley, and Andreas Kind. Probabilistic lossy counting: an efficient algorithm for finding heavy hitters. *ACM SIGCOMM Computer Communication Review*, 38(1):5, 2008.