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

ZIRUI LIU*, Peking University, China
HAILIN ZHANG*, Peking University, China
BOXUAN CHEN*, Peking University, China
ZIHAN JIANG, Peking University, China
YIKAI ZHAO, Peking University, China
YANGYU TAO, Tencent Inc., China
TONG YANG†, Peking University, China
BIN CUI†, Peking University, China

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 achieve 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 meets the above requirements. The design philosophy of CAFE+ is to dynamically allocate more memory to important features, and less to unimportant ones. We assign unique embedding to important feature and allow multiple unimportant features sharing one embedding. We propose a fast and lightweight feature monitor, to real-time capture feature importance and report important features. We theoretically analyze the accuracy of our feature monitor, and prove the superiority of CAFE+ from the aspect of model convergence. Extensive experiments show CAFE+ outperforms existing embedding compression methods, yielding 3.94% and 3.94% superior testing AUC on Criteo Kaggle dataset and CriteoTB dataset at a compression ratio of 10000×. Building on our conference version [114], this journal version introduces several novel designs (Implicit Importance Attenuation, Adaptive Threshold Adjustment, and ColdSifter) that enable CAFE+ to more effectively adapt to long-term online learning and achieve better model quality. All codes are available at GitHub [112].

CCS Concepts: • **Computing methodologies → Artificial intelligence**; • **Information systems → Online advertising**; • **Theory of computation → Sketching and sampling**.

Additional Key Words and Phrases: 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 information retrieval community, such as question answering [48, 67, 74, 117], semantic understanding [2, 99, 120, 124],

*Zirui Liu, Hailin Zhang, and Boxuan Chen contribute equally to this work.
†Tong Yang (yangtong@pku.edu.cn) and Bin Cui (bin.cui@pku.edu.cn) are corresponding authors.

The preliminary version of this paper titled "CAFE: Towards Compact, Adaptive, and Fast Embedding for Large-scale Recommendation Models" [114] is published in the 2024 ACM International Conference on Management of Data (SIGMOD), Santiago, Chile, June 9-14, 2024.

Authors' addresses: Zirui Liu, zirui.liu@pku.edu.cn, National Key Laboratory for Multimedia Information Processing, School of Computer Science, Peking University, China; Hailin Zhang, z.hl@pku.edu.cn, School of Computer Science & Key Lab of High Confidence Software Technologies, Peking University, China; Boxuan Chen, 2100012923@stu.pku.edu.cn, School of Computer Science & Key Lab of High Confidence Software Technologies, Peking University, China; Zihan Jiang, Jumbo0715@outlook.com, National Key Laboratory for Multimedia Information Processing, School of Computer Science, Peking University, China; Yikai Zhao, zyk@pku.edu.cn, National Key Laboratory for Multimedia Information Processing, School of Computer Science, Peking University, China; Yangyu Tao, brucetao@tencent.com, Tencent Inc., China; Tong Yang, yangtong@pku.edu.cn, National Key Laboratory for Multimedia Information Processing, School of Computer Science, Peking University, China; Bin Cui, bin.cui@pku.edu.cn, School of Computer Science & Key Lab of High Confidence Software Technologies, Peking University, China .

entity resolution [15, 16, 109], document retrieval [30, 58, 72], graph learning [28, 86, 91, 119], and recommendation systems [4, 23, 95, 108, 111, 121, 127, 129], 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 [10, 85, 98], and consume more than 50% training and 80% inference cycles at Meta's data centers in 2020 [26, 65].

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 [8, 27, 60, 66, 79, 107]. 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 [64, 104]. 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[1].

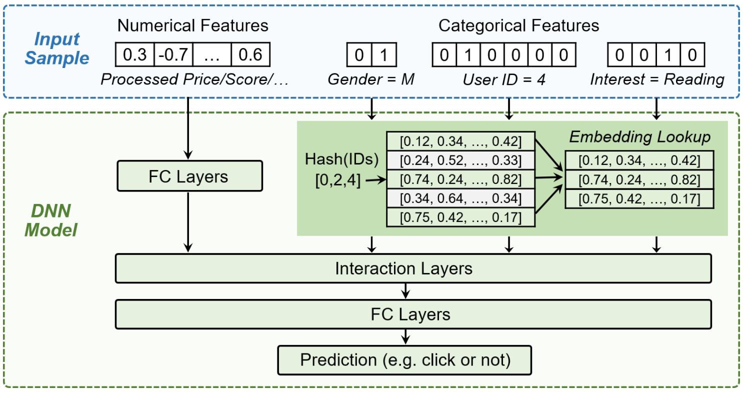


Fig. 1. Overview of DLRM.

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 focuses on the scenario of online training because it is more difficult. Generally, compression methods for online training can be directly applied to offline training. Embedding compression for online training has the following three important design requirements:

- **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 [63, 93]. Furthermore, training and deployment of embedding tables often occur on edge or end devices with small storage capacities, making the distributed solutions not always applicable [68]. As the model quality directly impacts profits, even a small change of 0.001 in DLRM's AUC (area under the ROC curve) is considerable [22]. Existing compression methods often lead to severe model degradation when memory constraints are small [122], emphasizing the need for memory-efficient compression methods that maintain model quality.

[1]Based on previous research works [34, 47, 64, 97, 123], we consider DLRMs with more than 100 million parameters as large-scale, and DLRMs with more than 10 billion parameters as extremely large-scale.

- **Low latency.** Low latency is a vital requirement in practical applications, as latency is a key metric of service quality [25]. Embedding compression methods must be fast enough to avoid introducing significant latency during embedding lookup.
- **Adaptability to dynamic data distribution.** In online training, the data distribution is not fixed as in offline training because the interest of users may change from time to time. For example, before and during the FIFA World Cup, users are likely to search extensively for match schedules, team merchandise, and related news. However, once the tournament ends, the demand for World Cup related content significantly decreases. To better illustrate this issue, we calculate the Kullback-Leibler (KL) divergence (an asymmetric measure of how one probability distribution is different from another probability distribution) 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. We can see that there is a significant difference between the feature distribution across different days, and generally the greater the temporal distance between the days, the greater the difference. Inspired by the observation that feature popularity distributions are often highly skewed [104, 116], existing methods compress the embedding table based on pre-captured feature importance distribution [20, 39, 110, 125]. However, most of them rely on fixed data distributions and cannot handle dynamic data distributions in online training [37, 104, 110], demanding new adaptive compression method.

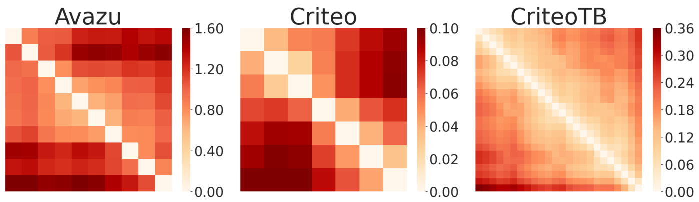


Fig. 2. Kullback-Leibler (KL) divergence between distributions on each day.

1.2 Limitations of Prior Art

Existing embedding compression methods can be categorized into two orthogonal types: row compression and column compression. Row compression aims to reduce the number of embeddings, while column compression seeks to decrease the size (length, precision, sparsity) of individual embeddings. As it is difficult for column compression to precisely control the compression ratio, this paper primarily focuses on row compression. Actually, as row compression and column compression are orthogonal, methods of column compression (such as low-precision quantization [44, 100], dimension reduction [20, 55, 125], and sparsification [13, 39]) can also be incrementally applied to our framework. Generally, there are two kinds of row compression methods: hash-based methods and adaptive methods.

Hash-based methods. These methods use hash functions to map a large number of features into a small number of embeddings with collisions [81, 94, 101]. Despite their simplicity and convenience, which have resulted in widespread industry use, these methods struggle to simultaneously achieve small memory overhead and high model quality. This is because hash collisions distort the semantic information of features. When the embeddings of multiple important features collide, the model is unable to distinguish between these important features, resulting in deviations from the optimal convergence direction. Although there are some hash-based solutions integrating offline feature frequency information to enhance model quality [110], they cannot be applied to online training.

Adaptive methods. Adaptive methods automatically distinguish and track important features throughout the online training process. AdaEmbed [42] logs the importance scores of all features, dynamically allocates embeddings for important features, and discards the embeddings of unimportant features. However, AdaEmbed needs to store the importance scores of all features, which increases linearly with the number of features. On the other hand, AdaEmbed directly discards the embeddings of unimportant features, which may also be valuable, leaving further room for model quality improvement. Therefore, AdaEmbed cannot compress embedding tables to an extremely small memory budget while guaranteeing model quality. Additionally, AdaEmbed also needs to periodically check the recorded scores to determine important features, which can impact the overall efficiency of training and serving.

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 simultaneously meets the three requirements.

1.3 Our Proposed Method

This paper presents **CAFE+**, a **C**ompact, **A**daptive, and **F**ast **E**mbedding compression framework. We observe that in most training data, the feature importance follows a highly skewed distribution, where only a few features are very important (Figure 3). Inspired by this observation, the key idea of CAFE+ is to dynamically allocate more memory resources to important features and less memory to unimportant ones. Specifically, CAFE+ proposes a light-weight feature monitor called HotSketch to compactly track feature importance and report important features. CAFE+ uses an embedding manager to independently manage the embeddings of different features. The embedding manager dynamically allocates unique embeddings to important features and shared embeddings to less important ones.

CAFE+ simultaneously satisfies the three design requirements. **(1) Memory efficiency:** By dynamically allocates memory resources according to feature importance, CAFE+ maintains good model quality within tight memory constraints. Guided by the idea of dynamically allocating resources according to feature importance, we further propose the Multi-layer Hash Embedding technique to optimize the memory allocation for unimportant features. Additionally, the feature monitor of CAFE+, namely HotSketch, also achieves small memory footprint, enabling high compression ratios. We also propose the ColdSifter technique to further optimize the memory usage of the feature monitor. **(2) Low latency:** The feature monitor of CAFE+ is small enough to be held in high-level CPU cache, and its lookup operation requires only a single hash calculation and memory access, thereby achieving small latency. On the other hand, the lookup operation to the embedding table of CAFE+ necessitates just several additional hash calculations and at most one extra memory access, incurring small latency during training and serving. **(3) Adaptability to dynamic data distribution:** CAFE+ makes many efforts to adapt to dynamic data distributions. CAFE+ introduces an embedding migration mechanism to migrate the embeddings between the unique embedding table and the shared embedding table according to real-time feature importance. The feature monitor of CAFE+ incorporates an Adaptive Threshold Adjustment mechanism to automatically adjust the importance threshold for important features according to current data distribution. We also devise an Implicit Importance Attenuation mechanism, where we define a time-aware feature importance score to make CAFE+ focus more on recent features and less on older ones, thereby incorporating the temporal information into CAFE+.

1.4 Key Contribution

- We propose CAFE+, a compact, adaptive, and fast embedding compression framework for online training.

- We introduce a time-aware feature importance score and devise HotSketch, a small and fast sketch algorithm to discern important features.
- We theoretically analyze the effect of HotSketch on identifying important features, and explain the reasons why CAFE+ can achieve better model quality through convergence analysis.
- We extensively evaluate CAFE+ across representative DLRM models and datasets. The results show that CAFE+ achieves 3.94% ∼ 5.39% higher testing AUC and 3.54% ∼ 11.26% lower training loss at 10000× compression ratio compared to existing methods.
- Building on CAFE in our conference version [114], CAFE+ includes the following novel designs: 1) The Implicit Importance Attenuation technique that enables CAFE+ to adapt to the continuously changing data distribution in online training (Section 3.7); 2) The Adaptive Threshold Adjustment mechanism to automatically tune the thresholds of CAFE+ during training (Section 3.6); 3) The ColdSifter optimization to improve the accuracy of the feature monitor (Section 3.5). With these designs, CAFE+ can more effectively adapt to long-term online training and achieve better model quality (Section 5.2.6).

Table 1. Symbols frequently used in this paper.

Symbol	Meaning	Symbol	Meaning
f_i	ID of a categorical feature	$\mathcal{S}$	Hot feature threshold
$s_{i,t}$	Increase of f_i's importance score in iteration t	$\mathcal{M}$	Medium feature threshold
S_i	Importance score of f_i ($S_i := \sum_t s_{i,t}$)	λ	Parameter controlling Threshold Adjustment
$\mathcal{U}$	*Uncompressed Embedding Table*	$\mathcal{H}_i$	The i-layer *Hash Embedding Table*
k	Size of *Uncompressed Embedding Table*	m_i	Size of the i-layer *Hash Embedding Table*
$\mathcal{H}$	*Hash Embedding Table*	$h_i(\cdot)$	Hash function mapping features into $\mathcal{H}_i$
m	Size of *Hash Embedding Table*	C	Bucket array of ColdSifter
$h(\cdot)$	Hash function mapping features into $\mathcal{H}$	$h_c(\cdot)$	Hash function mapping features into C
$\mathcal{B}$	Bucket array of HotSketch	$\mathcal{P}$	Cold feature threshold of ColdSifter
c	# slots per bucket of HotSketch/ColdSifter	α	Decay factor of feature importance
w	# buckets of HotSketch/ColdSifter	$\mathcal{A}$	Threshold controlling the range of α^{-t}
$h_b(\cdot)$	Hash function mapping features into $\mathcal{B}$	CR	Compression ratio
p_i	Embedding index of hot feature f_i in $\mathcal{U}$		

2 PRELIMINARY

In this section, we discuss the architecture of DLRMs, formulate the problem of embedding compression, and discuss the research context of online training and streaming recommendation systems. The symbols frequently used in this paper is listed in Table 1.

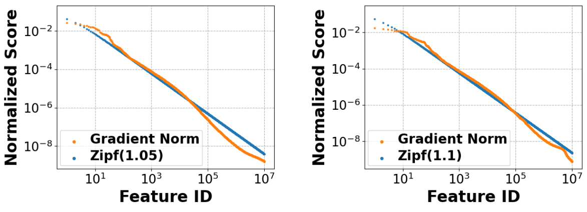
DLRM architecture: Figure 1 illustrates the overall architecture of DLRM. A training sample of DLRM has several categorical feature fields and numerical feature fields. For example, in Figure 1, gender, user ID and interest are categorical fields, while price and score are numerical fields. Each field has a certain number or a certain range of possible values, called features. During the training process, 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 (*e.g.*, WDL [8], DCN [88], DIN [128]) exploring different forms of interaction layers and neural network layers to enhance model performance, but all of them use the same embedding architecture.

The model parameters of DLRMs can be divided into two parts: the huge embedding table (tens of GBs) and the small neural network (several to hundreds of MBs). The former contains embeddings for all categorical features, *i.e.*, one unique embedding per feature if uncompressed. The latter is a network that interacts these embeddings and outputs predictions. In DLRMs, the size of the neural network part (just a few layers of matrix multiplication) is negligible compared to large embedding tables [42, 57]. It is highly desired to efficiently compress the embedding tables while simultaneously maintaining model accuracy.

Embedding compression: As storing the unique embedding for each categorical feature can lead to an unacceptably large model size, in recent years, there have been many research works focusing on compressing the embedding table [13, 20, 39, 42, 44, 55, 81, 94, 100, 101, 125]. Suppose there are n unique categorical features. The most classic *hash embedding* compress their embeddings into an array $\mathcal{H}$ of m embeddings ($m \ll n$) [81, 94, 101]. It uses a hash function $h(\cdot)$ to map each feature f_i into one embedding $\mathcal{H}[h(f_i)]$. In this way, multiple features can share the same embedding. We will discuss more works on embedding compression in Section 6.

Given an embedding compression solution, we formally define its *compression ratio CR* as the size of the uncompressed embedding table divided by the size of the compressed embedding table. For example, the compression ratio of the above *hash embedding* is $CR = \frac{n}{m}$. In practice, 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.

Online training and streaming recommendation systems: Online training refers to the process of learning a model in real-time using data that is available during the training procedure. Unlike offline training, where the model is trained on a pre-collected dataset, online training continuously updates the model with new incoming data, allowing it to dynamically adapt to changing environments or tasks. There are many online embedding frameworks used in streaming recommendation systems [24, 42, 73, 77, 87, 90]. We further illustrate the overall online training procedure using some examples. 1) AdaEmbed [42] is an online framework that automatically identifies important features and reallocates embeddings for them in real time. During online training, it records the importance of all features given streaming data, and dynamically maintains a set of the important features. The embeddings of these important features are recorded, while that of the unimportant features are discarded. 2) Another state-of-the-art work SCALL [77] uses a reinforcement learning (RL) model to periodically generate the optimal embedding configuration and uses this configuration to adjust the embedding allocation. Similar to SCALL, many other online training frameworks in streaming recommendation also periodically update the embedding configuration at the end of each session [24, 73, 87]. Like post-training pruning (PTP) schemes [50, 51], these solutions cannot achieve memory savings during model training. On the contrary, they require more memory and computation consumption during training, and have longer training time and more hyperparameters to be manually adjusted. Nonetheless, our results show that CAFE+ still outperforms SCALL (Figure 21) because it can adapt to the model performance at runtime and continuously optimize the embedding allocation throughout the entire training process. 3) As an online training framework, our CAFE+ also dynamically allocates more space to important features in real time. During online training, CAFE+ receives real-time streaming data and uses a feature monitor to continuously record feature importance. It reports current important features, and dynamically maintains the embeddings of important and unimportant features in two separate tables. In each training iteration, CAFE+ looks up the embeddings of important and unimportant features in the two tables respectively, and feeds these embeddings into downstream neural network for model training. The specific online training workflow of CAFE+ is shown in Algorithm 1, which will be described later in Section 3.1.



(a) Criteo. (b) CriteoTB.

Fig. 3. Comparing gradient norm and Zipf distributions.

Algorithm 1: Pseudo-code of CAFE+'s online training workflow

```
while not converged do
    // Step 1: fetch input data (feature IDs)
    feature_ids,labels ← fetch_next_data_batch();
    // Step 2: identify hot features and reports their indices in Uncompressed Embedding Table
    (feature_is_hot,hot_feature_indices) ← HotSketch.report(feature_ids);
    // Step 3: lookup embeddings for hot and non-hot features respectively
    feature_embeddings ← Embedding.lookup(feature_ids, feature_is_hot,
      hot_feature_indices);
    // Step 4: feed embeddings into downstream neural network for model training
    feature_gradients ← model.train_step(feature_embeddings,labels);
    feature_scores ← ||feature_gradients||_2;
    // Step 5 & Step 6: update feature importance and migrate embeddings
    HotSketch.update(feature_ids,feature_scores);
```

3 CAFE+ DESIGN

3.1 CAFE+ Overview

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. As in previous works [21, 38, 42], 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 3.8.2. 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 few features are very important. Figure 3 shows that the feature importance distributions in the Criteo dataset and the CriteoTB dataset are highly consistent with the Zipf distributions of parameters 1.05 and 1.1, respectively. Intuitively, we should allocate more memory to the embedding of each hot feature and less memory to that of non-hot feature. By strategically optimizing memory allocation based on feature importance, 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 feature monitor called HotSketch, to capture feature importance and report top-k hot features in real time[2]. We manage feature embeddings with two tables. For each hot feature, we allocate it a unique embedding in the *Uncompressed Embedding Table*. For each non-hot feature, we hash it into an embedding in the *Hash Embedding Table*, where multiple features can share one embedding. Algorithm 1 shows the overall online training workflow of CAFE+. In each training iteration, we first fetch data samples from the input training data

[2]We will further optimize the feature monitor with a structure called ColdSifter in Section 3.5.

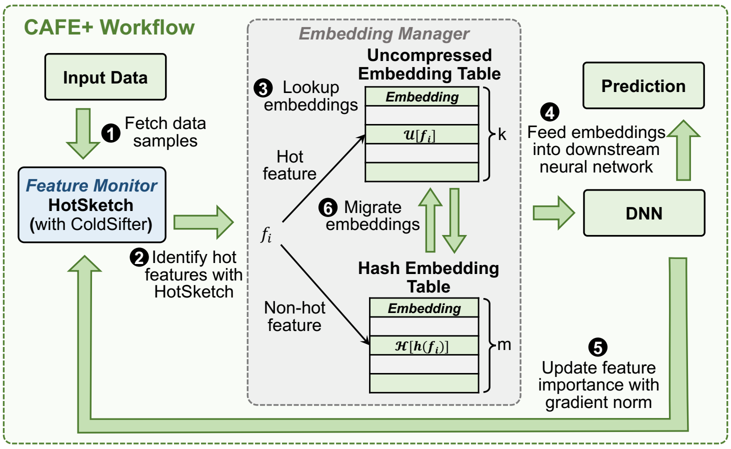


Fig. 4. CAFE+ workflow.

(step 1). Then we query each feature from these samples in HotSketch (step 2). For each feature, HotSketch reports whether it is a hot feature. For each hot feature, HotSketch also reports the index to its embedding in the *Uncompressed Embedding Table*. We lookup the embeddings for hot and non-hot features respectively (step 3). For a hot feature, we retrieve its unique embedding in the *Uncompressed Embedding Table*. For a non-hot feature, we retrieve its hashed embedding in the *Hash Embedding Table*. Afterwards, we feed the embeddings into the downstream neural network for prediction and get the gradient norm for each feature (step 4). Finally, we update the importance scores of these features in HotSketch using their gradient norms (step 5). During the update procedure, if a non-hot feature becomes a hot feature, we migrate its embedding from the *Hash Embedding Table* to the *Uncompressed Embedding Table* (step 6).

This section is organized as follows. We first describe the feature monitor in Section 3.2, where we explain how to identify hot features with HotSketch. Then we describe the embedding manager in Section 3.3, where we discuss how to lookup and migrate embeddings in the two tables. Guided by our design philosophy of dynamically allocating more memory to more important features, we further propose the Multi-layer Hash Embedding technique to better embrace the skewed feature importance distribution in Section 3.4. We further optimize the design of memory manager and feature monitor in Section 3.5-3.6. We design a novel implicit importance attenuation mechanism to perfectly adapt CAFE+ to online training in Section 3.7. Finally, we theoretically analyze the accuracy of HotSketch and explain the convergence advantages of CAFE+ in Section 3.8.

3.2 Feature Monitor: Identify Important Features with HotSketch

Problem statement: Given a feature f_i, we define its importance score S_i as the sum of the L2-norm of its gradient over all iterations, *i.e.*, $S_i := \sum_t ||\nabla g_t(f_i)||_2$ where $||\nabla g_t(f_i)||_2$ is the L2-norm of f_i's gradient in iteration t. As discussed in Section 3.1, the distribution of feature importance is highly skewed. Our feature monitor aims at identifying top-k important features, where k is the size of the *Uncompressed Embedding Table*. Specifically, given an incoming feature f_i, we should report whether f_i has the top-k largest importance score.

Design rationale: Towards the above goal, we design a novel sketch algorithm, called HotSketch, to capture top-k hot features in real time. This is essentially a problem of finding top-k items (features) in streaming data. Currently, Space-Saving [61] 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

Algorithm 2: Pseudo-code of the feature monitor (HotSketch)

```
Function HotSketch.report(feature_ids):
    feature_is_hot ← [];
    hot_feature_indices ← [];
    for $f_i$ in feature_ids do
        if $f_i$ is in $\mathcal{B}[h_b(f_i)]$ and $\hat{S}_i > \mathcal{S}$ then
            feature_is_hot.append(True);
            hot_feature_indices.append($p_i$); // $p_i$ is $f_i$'s embedding index in the Uncompressed Embedding Table
        else
            feature_is_hot.append(False);
            hot_feature_indices.append(Null);
    return (feature_is_hot,hot_feature_indices);
Function HotSketch.update(feature_ids,feature_scores):
    for ($f_i$,$s_{i,t}$) in (feature_ids,feature_scores) do
        if $f_i$ is in $\mathcal{B}[h_b(f_i)]$ then
            $\hat{S}_i \leftarrow \hat{S}_i + s_{i,t}$;
            if $\hat{S}_i \geq \mathcal{S}$ and $\hat{S}_i - s_{i,t} < \mathcal{S}$ then
                allocate a unique embedding in Uncompressed Embedding Table for $f_i$, initialize it with $\mathcal{H}[h(f_i)]$, and record its index $p_i$ in $\mathcal{B}[h_b(f_i)]$; // embedding migration (Section 3.3)
        else if $f_i$ is not in $\mathcal{B}[h_b(f_i)]$ and $\mathcal{B}[h_b(f_i)]$ is not full then
            insert ($f_i$, $s_{i,t}$) into an empty slot in $\mathcal{B}[h_b(f_i)]$;
        else if $f_i$ is not in $\mathcal{B}[h_b(f_i)]$ and $\mathcal{B}[h_b(f_i)]$ is full then
            find the slot with the smallest score ($f_{min}$, $\hat{S}_{min}$), and update it to ($f_i$, $\hat{S}_{min} + s_{i,t}$);
```

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 3.8.1), and empirically validate the advantages of HotSketch (Section 5.5).

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 $h_b(\cdot)$ to map each feature into one bucket. Each bucket contains c slots. Each slot stores a feature ID f_i and its estimated importance score $\hat{S}_i$. In our implementation, each slot also stores an index p_i to one embedding in the *Uncompressed Embedding Table*, by which the reported top-k feature can retrieve its unique embedding.

Update (step 5 in Figure 4; Algorithm 2): For each incoming feature f_i with score $s_{i,t} = ||\nabla g_t(f_i)||$, we first calculate the hash function to locate the hashed bucket $\mathcal{B}[h_b(f_i)]$. Then, we check $\mathcal{B}[h_b(f_i)]$ and there are three cases: (1) f_i is recorded in $\mathcal{B}[h_b(f_i)]$. We add $s_{i,t}$ to its importance score. Note that if f_i becomes a hot feature after updating the importance score, we migrate its embedding from the *Hash Embedding Table* to the *Uncompressed Embedding Table*, and record its index p_i in $\mathcal{B}[h_b(f_i)]$ (discussed in Section 3.3). (2) f_i is not recorded in $\mathcal{B}[h_b(f_i)]$ and there exists an empty slot in $\mathcal{B}[h_b(f_i)]$. We insert f_i into the empty slot by setting this slot to $(f_i, s_{i,t})$. (3) f_i is not recorded in $\mathcal{B}[h_b(f_i)]$ and $\mathcal{B}[h_b(f_i)]$ is full. We find the feature with the smallest score $(f_{min}, \hat{S}_{min})$ in $\mathcal{B}[h_b(f_i)]$, replace f_{min} with f_i, and add $s_{i,t}$ to $\hat{S}_{min}$. In other words, we set the slot $(f_{min}, \hat{S}_{min})$ to $(f_i, \hat{S}_{min} + s_{i,t})$. Figure 5 shows the examples of the above three cases.

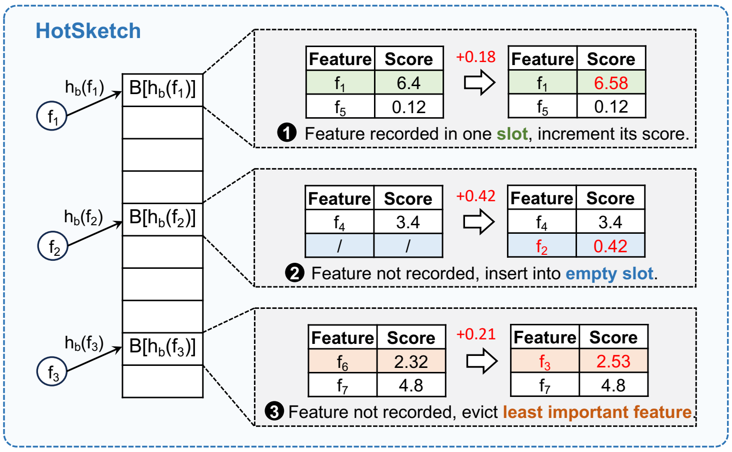


Fig. 5. Feature monitor: the HotSketch algorithm ($c = 2$).

Report (step 2 in Figure 4; Algorithm 2): For each incoming feature f_i, we calculate the hash function to locate its hashed bucket $\mathcal{B}[h_b(f_i)]$. If f_i is recorded in $\mathcal{B}[h_b(f_i)]$ and its importance score $\hat{S}_i$ exceeds a predefined threshold $\mathcal{S}$, we report f_i as a hot feature and return its embedding index p_i in the *Uncompressed Embedding Table*. Otherwise, if the importance score of f_i does not exceeds $\mathcal{S}$ or f_i is not in $\mathcal{B}[h_b(f_i)]$, we report f_i as a non-hot feature. In practice, the threshold $\mathcal{S}$ should closely matches the true importance score of the k^{th} hot features in real time. Therefore, using a fixed threshold $\mathcal{S}$ is not a wise choice. We will further design a mechanism to automatically adjust threshold $\mathcal{S}$ in Section 3.6.

Discussion on time- and space- overhead of HotSketch: (1) HotSketch has fast processing speed. It processes each incoming feature in a one-pass manner with one memory access and $O(1)$ time complexity. The processing procedure can be further accelerated with multi-threading and SIMD instructions [31]. (2) HotSketch is memory-efficient. Its bucket structure is neat and efficient, which avoids to store the complex pointers in Space-Saving [61]. In addition, after a brief cold start, all slots in HotSketch will be occupied, minimizing its memory waste.

Discussion on how to handle new features: In the training procedure of CAFE+, new features initially undergo a cold start phase. For a new feature f_i, it is initially reported by HotSketch as a non-hot feature and thus shares an embedding with other non-hot features in the *Hash Embedding Table*. As training progresses, the importance score of this feature f_i accumulates. If f_i eventually enters HotSketch and its estimated importance score $\hat{S}_i$ surpasses hot feature threshold $\mathcal{S}$, it is then reported by HotSketch as a hot feature and assigned a unique embedding in the *Uncompressed Embedding Table*. Afterwards, if f_i becomes less important again during subsequent training iterations[3], it will return to the *Hash Embedding Table* and resume sharing embeddings with other non-hot features. In other words, CAFE+ can always automatically identify current hot features and allocate them unique embeddings during the online training process (Figure 20(f)).

3.3 Embedding Manager: Manage Feature Embeddings in Two Tables

Data structure: As shown in Figure 4, CAFE+ stores the embeddings of hot features and non-hot features in two tables: *Uncompressed Embedding Table* $\mathcal{U}$ and *Hash Embedding Table* $\mathcal{H}$. The *Uncompressed Embedding Table* can store k unique embeddings for the top-k hot features. The *Hash*

[3]We will introduce the adaptive threshold adjustment mechanism and the time-decaying feature importance score in Section 3.6 and Section 3.7, respectively. With these two designs, a hot feature could become non-hot again.

Algorithm 3: Pseudo-code of the embedding manager

```
Function Embedding.lookup(feature_ids,feature_is_hot,hot_feature_indices):
    feature_embeddings ← [];
    for ($f_i$, $flag_i$, $p_i$) in (feature_ids,feature_is_hot,hot_feature_indices) do
        if $flag_i$ is True then
            feature_embeddings.append($\mathcal{U}[p_i]$);
        else
            feature_embeddings.append($\mathcal{H}[h(f_i)]$);
    return feature_embeddings;
```

Embedding Table stores m embeddings shared by the non-hot features, and it is associated with a hash function $h(\cdot)$ mapping features into embeddings in it.

Embedding lookup (step 3 in Figure 4; Algorithm 3): For each incoming feature f_i, if f_i is a hot feature reported by feature monitor, we retrieve its embedding in the *Uncompressed Embedding Table*. We use the index p_i recorded in HotSketch to directly retrieve this embedding. Otherwise, if f_i is a non-hot feature, we retrieve its embedding in the *Hash Embedding Table*. Specifically, we calculate hash function $h(f_i)$ and report embedding $\mathcal{H}[h(f_i)]$.

Embedding migration (step 6 in Figure 4): HotSketch uses a predefined threshold $\mathcal{S}$ to identify hot features. 1) After the update procedure (step 5 in Figure 4), if the importance score of a non-hot feature f_i surpasses threshold $\mathcal{S}$, it becomes a hot feature. In this case, we allocate a unique embedding in the *Uncompressed Embedding Table* for f_i, and initialize this embedding with $\mathcal{H}[h(f_i)]$. We store the index p_i to this allocated embedding in the slot of f_i in $\mathcal{B}[h_b(f_i)]$. This procedure is shown in the "update" function of Algorithm 2. 2) If after update, a hot feature f_i is evicted from HotSketch, or its importance score drops below threshold $\mathcal{S}$[4], we do nothing, meaning that we directly discard its embedding $\mathcal{U}[p_i]$. Actually, we can also use the embedding of f_i in the *Uncompressed Embedding Table* $\mathcal{U}[p_i]$ to cover $\mathcal{H}[h(f_i)]$. But since f_i is no longer important, it is unnecessary to preserve its embedding $\mathcal{U}[p_i]$ and use it to cover the embeddings of other non-hot features, because we want to treat all non-hot features equally. In our experiments, we find that the simplest solution of directly discarding the embedding of f_i can already attain good accuracy. We will further design a mechanism to avoid frequent migration operations in Section 3.6, thereby ensuring a smooth learning process.

3.4 Embedding Manager Optimization: Multi-layer Hash Embedding

Motivation: In the basic workflow of CAFE+, we categorize features into hot and non-hot ones with HotSketch. All non-hot features share one hash embedding table, meaning that they are treated equally. However, it is important to note that the importance of non-hot features also follows a highly skewed distribution (Section 3.2). As the key idea of CAFE+ is to dynamically allocate more resources to more important features, it is reasonable to further divide non-hot features based on their importance and assign them different resources accordingly. Towards this goal, we propose the multi-layer hash embedding table to further optimize the embedding manager.

Multi-layer Hash Embedding: As shown in Figure 6, we extend the *Hash Embedding Table* $\mathcal{H}$ for non-hot features (Figure 4) into a *Multi-layer Hash Embedding Table* consisting of 2 layers[5] $\mathcal{H}_1$

[4]In the basic version of feature monitor (Section 3.2), threshold $\mathcal{S}$ is set to a fixed value, so this case will not happen. We will introduce a mechanism to automatically tune threshold $\mathcal{S}$ (Section 3.6), and introduce the importance attenuation mechanism (Section 3.7). In such circumstances, the importance score can drop below threshold $\mathcal{S}$ after update.

[5]In this paper, we only build a 2-layer hash embedding table. It is also feasible to build an embedding table with more layers.

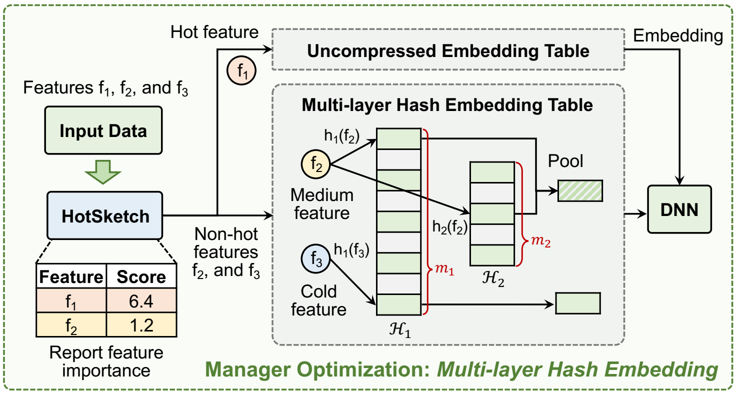

Hot feature
Uncompressed Embedding Table
Embedding
Features f1, f2, and f3
Input Data
HotSketch
Multi-layer Hash Embedding Table
h1(f2)
f2
Medium feature
Pool
DNN
h2(f2)
Non-hot features f2, and f3
h1(f3)
f3
Cold feature
m1
m2
H1
H2
Feature Score
f1 6.4
f2 1.2
Report feature importance
Manager Optimization: Multi-layer Hash Embedding

Fig. 6. Manager optimization: multi-layer hash embedding.

and $\mathcal{H}_2$. We further divide the non-hot features into medium features and cold features based on their importance scores. We extend the functionality of HotSketch to identify medium features by introducing another threshold $\mathcal{M}$[6]. Given an incoming feature f_i. If f_i is a cold feature, we lookup its embedding in the first layer and return $\mathcal{H}_1[h_1(f_i)]$. If f_i is a medium feature, we lookup its embeddings in both two layers and returns the pooling result $Pool(\mathcal{H}_1[h_1(f_i)], \mathcal{H}_2[h_2(f_i)])$. Here, the pooling operation $Pool(\cdot)$ can take various forms. In practice, we find that simple summation of embeddings performs well, as the embedding vectors of a feature are consistently updated in the same direction. Figure 6 illustrates the lookup procedure of multi-layer hash embedding: 1) For the hot feature f_1, we lookup its embedding in the *Uncompressed Embedding Table*. 2) For the medium feature f_2, we lookup its two embeddings in the two *Hash Embedding Tables*, and return the final embedding via a pooling operation. 3) For the cold feature f_3, we lookup its embedding in the first *Hash Embedding Table*.

Discussion: In this way, we achieve a smooth training process. Initially, a new feature is a cold feature, and its embedding is stored only in the first layer of the *Hash Embedding Table*. As the importance of this feature gradually increases and it becomes a medium feature, its embedding is then stored across two layers of the *Hash Embedding Table*. Once the feature becomes a hot feature, we allocate a unique embedding for it in the *Uncompressed Embedding Table* and migrate its embedding from the *Hash Embedding Table* to this location. Our experimental results show that with the multi-layer hash embedding optimization, we can reduce 0.25% training loss and increase 0.08% testing AUC.

3.5 Feature Monitor Optimization: Filter Cold Features with ColdSifter

Motivation: CAFE+ uses a feature monitor called HotSketch to distinguish features according to their importance. It is important for the feature monitor to maintain a small memory footprint. On the one hand, the small memory footprint facilitates CAFE+ to achieve larger compression ratios[7], which aids in its deployment on various edge devices with limited memory resources [68]. On the other hand, when deployed on CPU (Section 4), the small memory footprint of the feature monitor allows it to fit into higher-level CPU cache, thus enabling faster processing speeds in terms of higher throughput and lower latency. Therefore, it is desirable to make the feature monitor as small as possible without compromising its accuracy.

[6]We will also introduce a mechanism to automatically tune $\mathcal{M}$ in Section 3.6.
[7]Although the feature monitor is implemented on CPU memory, our experimental assessments of the *Compression Ratio (CR)* also account for its memory to ensure a fair evaluation (see Section 5.1.4).

Algorithm 4: Pseudo-code of the ColdSifter (replace line 11 in Algorithm 1)

```
Function ColdSifter.update(feature_ids,feature_scores):
    feature_ids_after_ColdSifter ← [];
    feature_scores_after_ColdSifter ← [];
    for ($f_i$, $s_{i,t}$) in (feature_ids,feature_scores) do
        if $f_i$ is in $C[h_c(f_i)]$ and $\hat{S}_i < \mathcal{P}$ then
            if $\hat{S}_i + s_{i,t} \geqslant \mathcal{P}$ then
                feature_ids_after_ColdSifter.append($f_i$);
                feature_scores_after_ColdSifter.append($\hat{S}_i + s_{i,t}$);
                $\hat{S}_i \leftarrow \mathcal{P}$;
            else
                $\hat{S}_i \leftarrow \hat{S}_i + s_{i,t}$;
            move ($f_i$, $\hat{S}_i$) to the front of $C[h_c(f_i)]$;
        else if $f_i$ is in $C[h_c(f_i)]$ and $\hat{S}_i == \mathcal{P}$ then
            feature_ids_after_ColdSifter.append($f_i$);
            feature_scores_after_ColdSifter.append($s_{i,t}$);
            move ($f_i$, $\hat{S}_i$) to the front of $C[h_c(f_i)]$;
        else if $f_i$ is not in $C[h_c(f_i)]$ then
            if $C[h_c(f_i)]$ is full then
                evict the tail of $C[h_c(f_i)]$;
            insert ($f_i$, $s_{i,t}$) to the front of $C[h_c(f_i)]$;
    HotSketch.update(feature_ids_after_ColdSifter,feature_scores_after_ColdSifter);
```

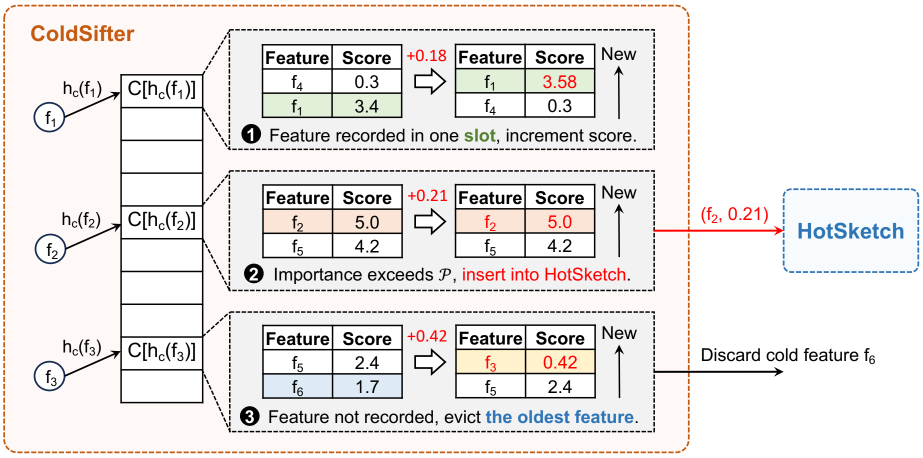

ColdSifter
h_c(f_1)
f_1
C[h_c(f_1)]
h_c(f_2)
f_2
C[h_c(f_2)]
h_c(f_3)
f_3
C[h_c(f_3)]
Feature Score f4 0.3 f1 3.4
+0.18
Feature Score f1 3.58 f4 0.3
New
❶ Feature recorded in one slot, increment score.
Feature Score f2 5.0 f5 4.2
+0.21
Feature Score f2 5.0 f5 4.2
❷ Importance exceeds 𝒫, insert into HotSketch.
(f2, 0.21)
HotSketch
Feature Score f5 2.4 f6 1.7
+0.42
Feature Score f3 0.42 f5 2.4
❸ Feature not recorded, evict the oldest feature.
Discard cold feature f6

Fig. 7. Monitor optimization: the ColdSifter algorithm ($c = 2$, $\mathcal{P} = 5$).

Rationale: In this subsection, we optimize the memory of feature monitor by adding another data structure called ColdSifter before HotSketch. We use ColdSifter to filter out most cold features beforehand and only let the non-cold features enter HotSketch. The rationale behind ColdSifter is to embrace the skewed distribution of feature importance again. We observe that in the basic design of the feature monitor, a large number of cold features unnecessarily consume a lot of space in HotSketch. Since CAFE+ only focuses on hot features (and medium features), it is desired to use a small structure to efficiently keep most cold features outside, thereby eliminating their interference to HotSketch. Towards the above goal, the key idea of ColdSifter is to use a set-associative LRU (Least Recently Used) queue to record the importance scores of recent features [46]. For a feature f_i, if its cumulative importance score exceeds a predefined threshold $\mathcal{P}$ before being evicted from the LRU queue, we consider f_i to be a non-cold feature and insert it into the subsequent HotSketch. Otherwise, we regard f_i as a cold feature and discard it.

Data structure: As shown in Figure 7, similar to HotSketch, the data structure of ColdSifter is also an array of w buckets $C[1], \cdots, C[w]$[8]. We use a hash function $h_c(\cdot)$ to map each feature into one bucket. Each bucket contains c slots. Each slot stores a feature ID and its recent importance score. The c features in a bucket are sorted according to their last accessed time (the LRU rule), with the first feature being the most recently accessed one.

Update (step 5 in Figure 4; Algorithm 4): For each incoming feature f_i with score $s_{i,t}$, we check its hashed bucket $C[h_c(f_i)]$. 1) If f_i is recorded in $C[h_c(f_i)]$ and its recorded score $\hat{S}_i$ is less than $\mathcal{P}$, we regard f_i as a cold feature. We increment its score by $s_{i,t}$ and move it to the front of $C[h_c(f_i)]$ (case 1 in Figure 7). Note that if the importance score of f_i reaches $\mathcal{P}$ after increment ($\hat{S}_i + s_{i,t} \geqslant \mathcal{P}$), we set its score to $\mathcal{P}$ and insert $(f_i, \hat{S}_i + s_{i,t})$ to the subsequent HotSketch. 2) If f_i is recorded in $C[h_c(f_i)]$ and its score is already $\mathcal{P}$, we regard f_i as a non-cold feature. We move f_i to the front of $C[h_c(f_i)]$ and insert $(f_i, s_{i,t})$ to the subsequent HotSketch (case 2 in Figure 7). 3) If f_i is not recorded in $C[h_c(f_i)]$, we insert $(f_i, s_{i,t})$ to the front of $C[h_c(f_i)]$. If $C[h_c(f_i)]$ is already full before insertion, we evict the least recent feature in $C[h_c(f_i)]$ (case 3 in Figure 7).

Discussion: In this way, by efficiently discarding cold features beforehand, our feature monitor achieves higher accuracy and smaller memory overhead. Experimental results show that with the same total memory usage, the optimized feature monitor enhances the recall rate on finding hot features by 4.9% compared to the basic version (Figure 20(c)).

3.6 Optimization: Adaptive Threshold Adjustment

Motivation: In the basic design of CAFE+, we use two fixed thresholds $\mathcal{S}$ and $\mathcal{M}$ to identify hot features and medium features respectively. However, in practice, feature importance always dynamically fluctuates over time. This phenomenon will become more pronounced after we introduce the time-decaying importance score in Section 3.7. Therefore, using fixed thresholds to distinguish important features is not a wise choice. In this subsection, we design an adaptive threshold adjustment mechanism, which can automatically corrects the two thresholds $\mathcal{S}$ and $\mathcal{M}$ to their optimal values. This approach also helps to avoid frequent embedding migrations, thereby ensuring a smooth learning process.

Adaptive adjustment for hot feature threshold $\mathcal{S}$: Ideally, hot feature threshold $\mathcal{S}$ should be set to the importance score of the current k^{th} hottest feature, which always fluctuates over time. We design an adaptive adjustment mechanism to automatically correct $\mathcal{S}$ to its optimal value. This mechanism works by maintaining the number of current hot features reported by HotSketch (denoted as $\mathcal{N}_{hot}$) in real time. In other words, $\mathcal{N}_{hot}$ is the number of features whose estimated importance scores exceed current threshold $\mathcal{S}$. Specifically, after initializing $\mathcal{S}$, we set $\mathcal{N}_{hot}$ to zero. During the update procedure of HotSketch (Step 5 in Figure 4), if a feature's importance score surpasses $\mathcal{S}$ for the first time, we increment $\mathcal{N}_{hot}$ by one. Every time $\mathcal{N}_{hot}$ exceeds $\lambda \times k$, we adjust threshold $\mathcal{S}$ and the features in *Uncompressed Embedding Table* as follows. We traverse all slots in HotSketch to acquire the current top-k hot features (the features with the k largest importance

[8]To avoid introducing more symbols, we use the same symbols to represent the number of buckets w and the number of slots per bucket c for both ColdSifter and HotSketch. Actually, the values of w and c for ColdSifter and HotSketch do not need to be the same.

scores), and set $\mathcal{S}$ to the recorded importance score of the k^{th} hottest feature. Subsequently, we adjust the features in *Uncompressed Embedding Table*, placing the current k^{th} hottest features into *Uncompressed Embedding Table*. After this adjustment, we set $\mathcal{N}_{hot} = k$.
Adaptive adjustment for medium feature threshold $\mathcal{M}$: Medium feature threshold $\mathcal{M}$ should be set to maximize the utilization of the second-layer *Hash Embedding Table* ($\mathcal{H}_2$ in Figure 6). In other words, we expect the number of features entering $\mathcal{H}_2$ to be neither too many nor too few. Assuming $\mathcal{H}_2$ can hold m_2 embeddings (Figure 6), our experimental results with *Hash Embedding* (Figure 10) indicate that the accuracy of *Hash Embedding* significantly declines when compression ratio exceeds 100. Therefore, we aim to automatically adjust threshold $\mathcal{M}$ to ensure that the number of features entering $\mathcal{H}_2$ approximately remains at $100m_2$, so as to fill $\mathcal{H}_2$ as much as possible without reducing much accuracy. Towards this goal, each time we adjust threshold $\mathcal{S}$ and traverse HotSketch, we set threshold $\mathcal{M}$ to the importance score of the $(k + 100m_2)^{th}$ hottest feature in HotSketch[9]. In this way, we ensure that precisely $100m_2$ features enter $\mathcal{H}_2$ and maximize its utilization.
Discussion: Our adaptive threshold adjustment mechanism introduces an extra parameter λ, which controls the frequency of threshold adjustment. A large λ results in infrequent threshold adjustments, preventing $\mathcal{S}$ and $\mathcal{M}$ from being updated to their optimal values in a timely manner. By contrast, a small λ leads to frequent changes in the set of hot features and excessive embedding migrations. We find the optimal value of λ does not change with compression ratio, which is about 1.2 (Figure 18(g)). Therefore, we recommend setting $\lambda = 1.2$ by default. For parameters k (the size of the *Uncompressed Embedding Table* for hot features) and m (the size of the *Hash Embedding Table* for non-hot features), their values are determined by the memory ratio allocated to hot features (*i.e.*, hot percentage). Note that unlike $\mathcal{S}$ and $\mathcal{M}$ whose optimal values change dynamically over the online training process, hot percentage (k and m) determines the allocation of embedding space between hot and non-hot features in CAFE+, and cannot be dynamically adjusted during training. Therefore, we just set it to the empirically optimal value. We find the optimal value of hot percentage varies with compression ratio. We present the optimal hot percentage across compression ratios in Figure 18(c), which can guide the configuration of hot percentage in practice. For example, at a compression ratio of 1000×, we recommend setting hot percentage to 0.7.

3.7 Optimization: Implicit Importance Attenuation

Motivation: In recommendation systems, popularity trends often evolve over time as user preferences and interests shift [35, 82]. These trends reflect the rise or decline in the popularity of specific features (items, categories, or topics) over time. For example, certain products might gain popularity due to seasonal factors, events, or viral marketing campaigns. Due to the pervasive herd mentality among users, their decisions are frequently influenced by the popularity of items at any given moment [54]. This dynamic nature of user engagement makes it crucial to consider recency when selecting important features. Features representing older, less relevant trends may no longer effectively contribute to model performance and could even introduce noise, thus reducing the overall accuracy of recommendations. This effect is particularly apparent for time-sensitive or frequently updated items such as fashionable clothing, movies, and news [35]. For example, the findings from Ji et al. [33] demonstrated that recommending the most popular movies from the past month significantly outperformed recommending movies with the highest global popularity. Therefore, it is crucial for DLRMs to take recency information into consideration. By incorporating a time-aware feature importance score that emphasizes recency, the model can adapt to changing popularity trends and maintain its effectiveness in reflecting current user behaviors and interests. In

[9]We set $\mathcal{M} = 0$ if there are less than $k + 100m_2$ features recorded in HotSketch, meaning that we consider all features that are not filtered out by ColdSifter as medium features.

other words, this recency bias ensures that the model remains responsive to the temporal dynamics of the recommendation environment.

Naive solution: To closely identify current important features in real-time, we should focus more on recent features and less on older ones. A straightforward solution is to periodically decay the importance score of all recorded features in HotSketch, which is adopted in our conference version [114]. Ideally, during each training iteration, we should decay the importance score of each recorded feature in HotSketch by α $(0 < \alpha < 1)$. However, as the decay operation requires traversing the entire HotSketch, which is implemented on CPU (Section 4), performing this decay operation every iteration would significantly degrade training efficiency. Therefore, in our conference version [114], rather than decaying each importance score by α every iteration, we periodically decay each importance score by α^{T_d} every T_d iterations. Here, T_d is a predefined hyperparameter that balances between training efficiency and model performance. Using a small T_d means traversing HotSketch more frequently, which can decrease training efficiency. By contrast, using a large T_d will prevent CAFE+ from catching up with the dynamic variation in feature importance, thereby diminishing model performance.

Implicit importance attenuation: In this paper, we adopt a new way to consider temporal information by introducing a time-aware feature importance score. Our method achieves the same effect as the above ideal solution (decaying each score every iteration) without the need for explicit traversal of HotSketch. First, consider the ideal solution of decaying the importance score of each feature by α $(0 < \alpha < 1)$ every iteration. In this design, suppose the training process has taken T iterations, the importance score of a feature f_i can be written as $S_i = \sum_{t=1}^{T} s_{i,t} \cdot \alpha^{T-t} = \alpha^T \times \sum_{t=1}^{T} s_{i,t} \cdot \alpha^{-t}$. Here, $s_{i,t} = ||\nabla g_t(f_i)||_2$ is the importance increase of f_i in iteration t. Notice that the goal of our HotSketch is to identify the hot features with top-k largest importance scores, rather than to estimate the exact importance scores. We can factor out the common term α^T from all feature scores. This leads to a newly defined importance score $S'_i = \sum_{t=1}^{T} s'_{i,t} = \sum_{t=1}^{T} s_{i,t} \cdot \alpha^{-t}$. During the update procedure of feature monitor (step 5 in Figure 4), we insert the modified importance increase $s'_{i,t} = s_{i,t} \cdot \alpha^{-t}$ into HotSketch (or ColdSifter). In this way, we maintain the automatically decaying importance scores in the feature monitor, thereby achieving implicit importance attenuation.

Counter-normalization technique: After training for a long time, the implicit decay factor α^{-t} and the new feature importance score $S'_i = \sum_{t=1}^{T} s_{i,t} \cdot \alpha^{-t}$ will grow into very large values, eventually leading the counters in the feature monitor to overflow. We propose the counter-normalization technique to address this issue. Specifically, we use a predefined threshold $\mathcal{A}$ to limit the value of the implicit decay factor α^{-t} to the interval $(1, \mathcal{A}]$. Every time the decay factor α^{-t} exceeds threshold $\mathcal{A}$, we divide the decay factor and all counters in HotSketch by $\mathcal{A}$. At the same time, we also divide the hot feature threshold $\mathcal{S}$ and the medium feature threshold $\mathcal{M}$ by $\mathcal{A}$ to ensure a smooth training process. To avoid the inefficient explicit traversal to HotSketch, we implement the counter division operation of HotSketch in a lazy manner. Specifically, we use a global 1-bit flag to record the parity (odd/even) of the number of times the decay factor has been divided. We append a local 1-bit flag to each bucket of HotSketch. During the update procedure of feature monitor (step 5 in Figure 4), every time a bucket is accessed, we first compare its local flag with the global flag. If its local flag differs from the global flag, we update the local flag to the global one and divide the recorded importance score of each feature in this bucket by $\mathcal{A}$. A small design detail is that when performing the Adaptive Threshold Adjustment operation (Section 3.6), we must first perform the division operation on all buckets where the local flag differs from the global flag. This ensures that all local flags match the global flag before proceeding with the top-k feature selection and thresholds setting. In this way, with the counter-normalization technique, we not only avoid the potential counter overflow error but also reduce the counter size of HotSketch to save memory.

3.8 Theoretical Analysis

In this section, we first analyze the accuracy of our feature monitor (HotSketch) in identifying hot features in Section 3.8.1. Then we theoretically explain the advantages of the embedding management framework of CAFE+ over traditional hash embedding from the aspect of model convergence in Section 3.8.2. The detailed proofs of the theorems in this section can be found in our supplementary material [113].

3.8.1 *Effect of HotSketch on Identifying Hot Features.*

In this subsection, we theoretically analyze the performance of HotSketch in identifying hot features. Based on the theoretical properties of Space-Saving [61], we first derive a relatively loose lower bound of the probability that a hot feature is recorded in HotSketch in Theorem 3.1.

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 a hot feature with a total score larger than* $\gamma\|a\|_1$, *it can be held in HotSketch with probability at least:* $\Pr > 1 - \frac{1-\gamma}{(c-1)\gamma w}$.

REMARK. Theorem 3.1 provides the accuracy lower bound of HotSketch in identifying a certain hot feature f_i in a data stream of arbitrary distribution. In this theorem, c (number of slots per bucket) and w (number of buckets) control the size of HotSketch, and γ controls the importance of the hot feature f_i to be identified. We can see that larger c, w, and γ correspond to higher accuracy. Therefore, this theorem tells us the following implications of HotSketch under arbitrary data distribution: 1) Larger memory usage of HotSketch goes with higher accuracy in identifying hot features; 2) It is easier for HotSketch to identify the features with higher importance scores. These implications are consistent with our intuition and design goal.

We further derive a tighter lower bound of the probability that a hot feature is recorded in HotSketch in Theorem 3.3, which is based on Lemma 3.2.

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 vector 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. Suppose we would like to check whether the* k' *hottest features can be hashed into the buckets. Then the mathematical expectation of the score sum of the non-hot features entering each bucket is:* $\mathbb{E}[\hat{f}] \leqslant \frac{\|a\|_1 \cdot k'^{1-z}}{w}$ *with probability at least* $3^{-\frac{k'}{w}}$ *for* $z > 1$ *and* $n \longrightarrow +\infty$.

REMARK. Lemma 3.2 provides an upper bound on the expected importance score sum $\mathbb{E}[\hat{f}]$ for non-hot features (non-top-k' features) within each bucket of HotSketch. Here, non-hot features can be considered as interference to HotSketch, because its mission is to record hot features. The importance score sum $\hat{f}$ can be viewed as the degree of the interference. Lemma 3.2 tells us that when the data stream distribution (*i.e.*, feature distribution) is sufficiently skewed ($z > 1$) and the number of features is large enough ($n \longrightarrow +\infty$), the interference from non-hot features remains low and has an analytical upper bound. This upper bound forms the basis for the subsequent Theorem 3.3.

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. Let* $k' = \eta w$. *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* $n \longrightarrow +\infty$.

Remark. Theorem 3.3 improves the accuracy lower bound in Theorem 3.1 under the data stream of Zipfian distribution, which also provides the accuracy lower bound of HotSketch in identifying a certain hot feature f_i. We can see that larger c, w, γ, and z correspond to higher identification accuracy. Here, larger c and w means more memory usage of HotSketch; larger γ means higher importance score of f_i; and larger z means the data stream distribution is more skewed. This theorem tells us the following implications of HotSketch under Zipfian data distribution: 1) Larger memory usage of HotSketch goes with higher accuracy in identifying hot features. 2) It is easier for HotSketch to identify the features with higher importance scores. 3) HotSketch achieves higher accuracy under skewed data distributions. The more skewed the distribution, the higher the accuracy. These implications are consistent with our intuition and design goal. 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. The simulation results are shown in Figure 8, where 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.

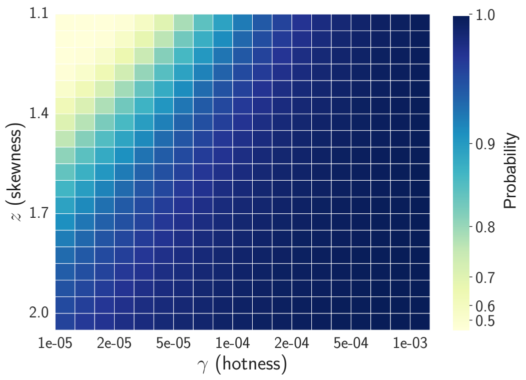


Fig. 8. Numerical analysis for the probability of HotSketch in identifying hot features, where the x-axis represents the hotness of the feature and the y-axis represents the skewness of the feature hotness/importance distribution (Theorem 3.3).

Based on Theorem 3.3, we finally analyze how the data distribution and the parameters of HotSketch affect its performance in Corollary 3.4, and derive the theoretically optimal parameters for HotSketch in Corollary 3.5.

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.*

Remark. This corollary formally summarizes the three implications of Theorem 3.3 discussed above, which are consistent with our intuition and design goal.

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

Remark. Corollary 3.5 gives the optimal setup of parameter c (number of slots per bucket) for HotSketch under Zipfian data distribution. 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 Space-Saving [61]. 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 the optimal parameters in Corollary 3.5 is consistent with our experimental results in Figure 20(a).

3.8.2 ***Convergence Analysis against Deviation.***

In this subsection, we try to theoretically explain the advantage of CAFE+ over traditional hash embedding [81, 94, 101] from the aspect of model convergence. As discussed in Section 1.2, in hash-based methods, hash collisions between different features can lead to embedding deviation that hinders model convergence. Next, we prove that CAFE+ can reduce the deviation of embedding gradients, and thus reduce the deviation of embedding parameters. We analyze how embedding deviation affects the convergence of SGD algorithm by studying the following (non-convex) empirical risk minimization problem:

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

where α is learning rate, $\boldsymbol{g}_{\boldsymbol{i_t}} = \nabla f_i(\boldsymbol{\theta}_{i_t})$ is the standard gradient without compression, $\widetilde{\boldsymbol{g}}_{\boldsymbol{i_t}}$ is the real gradient with compression[10]. Our analysis is based on Assumption 1 [3, 19] and Theorem 3.6.

Assumption 1. *For $\forall i \in \{1, 2, ..., N\}$, $\boldsymbol{\theta}, \boldsymbol{\theta}' \in \mathbb{R}^D$, we make the following assumptions:*
(1. L-Lipschitz) $\|\nabla f_i(\boldsymbol{\theta}) - \nabla f_i(\boldsymbol{\theta}')\| < L\|\boldsymbol{\theta} - \boldsymbol{\theta}'\|$;
(2. Bounded moment) $\mathbb{E}\left[\|\nabla f_i(\boldsymbol{\theta})\|\right] < \sigma_0$, $\mathbb{E}[\|\nabla f(\boldsymbol{\theta})\|] < \sigma_0$;
(3. Bounded variance) $\mathbb{E}[\|\nabla f_i(\boldsymbol{\theta}) - \nabla f(\boldsymbol{\theta})\|] < \sigma$;
(4. Existence of global minimum) $\exists f^*,\ s.t. f(\boldsymbol{\theta}) \geqslant f^*$.

Theorem 3.6. *Suppose we run SGD optimization with CAFE+ on DLRMs satisfying the assumptions above, with $\epsilon_t = \|\widetilde{\boldsymbol{g}}_{\boldsymbol{i_t}} - \boldsymbol{g}_{\boldsymbol{i_t}}\|$ as the deviation of embedding gradients. Assume the learning rate α satisfies $\alpha < \frac{1}{L}$. After T steps, for $\overline{\boldsymbol{\theta}}_T$ which is randomly selected from $\{\boldsymbol{\theta}_0, \boldsymbol{\theta}_1, \cdots, \boldsymbol{\theta}_{T-1}\}$, we have:*

$$\mathbb{E}[\|\nabla f(\overline{\boldsymbol{\theta}}_T)\|^2] \leqslant \frac{f(\boldsymbol{\theta}_0) - f^*}{T\alpha(1-\alpha L)} + \frac{\alpha(2L\sigma^2 + \sigma_0^2)}{2(1-\alpha L)} + \frac{(1+\alpha^2 L)\sum_{t=0}^{T-1}\mathbb{E}[\epsilon_t^2]}{2T\alpha(1-\alpha L)}$$

Remark. We use Theorem 3.6 to explain the advantage of CAFE+ over traditional hash embedding. We can see that as T increases, with a proper learning rate $\alpha = O\left(\frac{1}{\sqrt{T}}\right)$, the first two terms at the right hand side of the inequality in Theorem 3.6 tend to 0, and thus the convergence of SGD is mainly influenced by the deviation ϵ_t. Although there is no bound for the deviation ϵ_t, CAFE+ can still achieve smaller deviation than hash embedding. CAFE+ allocates unique embeddings for hot features, and thus their deviation can be ignored. For the non-hot features, their deviation is incurred by 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. For basic hash embedding, the weight deviation of a feature comes from the gradients of any other features. By contrast, as CAFE+ uses gradient norm as the importance scores of

[10]To adhere to common symbols in theoretical machine learning, the meanings of some symbols used in this subsection differ from those in Table 1. Explanations of all symbols in this subsection can be found in our supplementary materials [113].

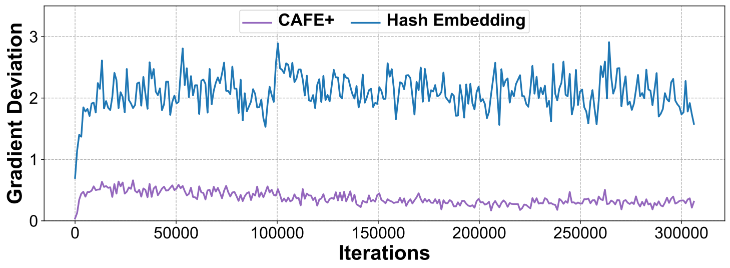


Fig. 9. Experimental comparison of the gradient deviation between CAFE+ and hash embedding.

features, the gradient norms of non-hot features are relatively small, which limits their overall weight deviation.

Experimental analysis (Figure 9): We further empirically compare the gradient deviation between CAFE+ and hash embedding. We set the compression ratio of CAFE+ and hash embedding to 10×. For CAFE+, we allocate 50% memory to its feature monitor and the embeddings of hot features. Figure 9 shows the gradient deviation over the training process. The results show that the gradient deviation of CAFE+ is always 5× ∼ 6× smaller than that of hash embedding. These experimental results complement our theoretical analysis, demonstrating that the embedding architecture of CAFE+ (*i.e.*, separating hot features from non-hot features) is superior to basic hash embedding.

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 [112]. We will also extend CAFE+ to other frameworks (TensorFlow, Hetu [62], etc.) in the future.

CAFE+ backend: We implement the feature monitor module of CAFE+ (including HotSketch and ColdSifter) with C++ to reduce the overall latency, and implement the rest modules of CAFE+ (including embedding manager, downstream DNN, *etc.*) using PyTorch operators. By default, we set the number of slots per bucket in HotSketch (and ColdSifter) to $c = 4$ and control the total number of buckets in HotSketch to the pre-determined number of hot features ($w = k$). We use one feature monitor to handle all categorical feature fields instead of using one monitor per field. This is because the distribution of hot features across different fields is unclear, and it is reasonable to directly handle them with importance scores in a unified manner. We also 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.

Fault tolerance: We register the states of CAFE+'s feature monitor as buffers in PyTorch module, so that these states can be saved and loaded alongside model parameters. This simple design requires no additional modifications to existing DLRM, and enables CAFE+ to use checkpoints for training and inference. When training is resumed with checkpoints, model parameters and CAFE+'s states are reloaded simultaneously.

Memory management: We place the feature monitor module of CAFE+ on CPU. This is because the feature monitor of CAFE+ is not compute-intensive and can be further accelerated with the parallel computing techniques of CPU (multi-threading, SIMD [31]). Built upon PyTorch operators, the embedding manager module of CAFE+ 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 recommendation datasets and compare CAFE+ (and CAFE) with existing embedding compression methods. We experimentally show that CAFE+ satisfies all three requirements. We also design experiments to validate the effectiveness of HotSketch. In this section, label "CAFE" and "CAFE+" refer to our proposed embedding framework without and with the three added optimizations in our journal version respectively, namely the optimizations of feature monitor (Section 3.5), adaptive threshold (Section 3.6), and implicit importance attenuation (Section 3.7).

5.1 Experimental Settings

5.1.1 Models and Datasets. In Section 5.2-5.5, we conduct experiments on three representative recommendation models for click-through rate (CTR) prediction task: DLRM [66][11], WDL [8], and DCN [88]. These models are popular in both academia and industry. All models follow the architecture discussed in Section 2, which consists of a large embedding table for categorical features and a downstream neural network part. Different models have slight differences in neural network parts. We plot these model structures in our supplementary materials [113]. 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. In Section 5.6-5.7, we conduct experiments on two graph-based models for top-N recommendation task: LightGCN [29] and PinSage [106]. Both models have embedding tables for users and items[12]. LightGCN uses multiple graph convolution layers to aggregate user and item embeddings from their neighbors in a user-item bipartite graph. PinSage further combines graph convolutions with a random walk-based sampling technique to aggregate neighborhood embeddings. The final user and item embeddings after aggregation are used to compute the recommendation score. We use CAFE+ to optimize the embedding tables in LightGCN and PinSage. As an embedding layer plugin, our method can be easily generalized to other recommendation models with little effort. We set the configurations of these models as in their original papers.

Table 2. Statistics of the CTR 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

In Section 5.2-5.5, we conduct experiments on three large-scale Click-Through Rate (CTR) datasets Avazu [89], Criteo [41], KDD12 [1], and an extremely large-scale CTR dataset CriteoTB [40]. Criteo Kaggle Display Advertising Challenge Dataset (Criteo) [41] and Criteo Terabytes Click Logs (CriteoTB) [40] contain 7 and 24 days of ads click-through rate (CTR) prediction data respectively, which are adopted in the MLPerf benchmark [70]. Each data sample has 13 numerical fields and 26

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

[12]PinSage did not explicitly implement the embedding table of users. For each user, PinSage recommends the top-N items that are closest in the embedding space to one of the most recently interacted items by the user. We follow this implementation as described in the original paper of PinSage.

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) [89] and KDD Cup 2012, Track 2 (KDD12) [1] 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 [66, 131] or our experiments on the uncompressed models. The statistics of these datasets are listed in Table 2. Since the numerical field is not our focus, we omit it from the table. In Section 5.4, we further 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. In Section 5.6-5.7, we conduct experiments on two datasets for top-N recommendation: Yelp2020 and #nowplaying-RS. Yelp2020 dataset is a subset of Yelp's businesses, reviews, and user data. We obtain the Yelp2020 dataset from the official code repository of HGCF [83][13]. #nowplaying-RS [71] is a large-scale benchmark for music recommendation, containing music listening events of users and tracks collected from Twitter. We obtain the #nowplaying-RS dataset from its official code repository[14]. The detailed statistics of the two ranking datasets are summarized in Table 3.

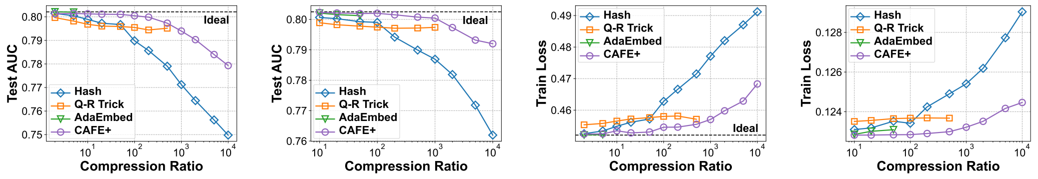
Table 3. Statistics of the datasets for top-N recommendation.

Dataset	**#User**	**#Item**	**#Interaction**	**Density**
Yelp2020	71,135	45,063	1,782,999	0.056%
#nowplaying-RS	138,721	346,122	11,609,883	0.024%

5.1.2 Baselines. We compare CAFE+ with Hash Embedding [94], Q-R Trick [81], and AdaEmbed [42]. Hash embedding is a simple baseline using only one hash function, which serves as a performance 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 CAFE+ with uncompressed embedding tables (ideal). In Section 5.2.4, we compare CAFE+ with a column compression method MDE [20]. In Section 5.6, we compare CAFE+ with three recent embedding compression frameworks in top-N recommendation tasks: PEP [51], CERP [50], and SCALL [77]. PEP [51] is a post-training pruning scheme that adaptively learns the pruning thresholds and uses them to prune the embedding table during deployment. CERP [50] is also a post-training pruning scheme. It represents each entity by combining a pair of embeddings from two independent meta-embedding tables (like Q-R [81]), which are then jointly pruned via learnable element-wise thresholds. SCALL [77] is a state-of-the-art AutoML dimension search-based algorithm that uses an reinforcement learning model to determine the optimal embedding dimension size and periodically adjust the embedding allocation in a streaming setting (discussed in Section 2). Note that these three baselines need to retrain the model after pruning/re-allocating the embedding tables. By default, the hyperparameters of these baselines are the same as in the original paper or code.

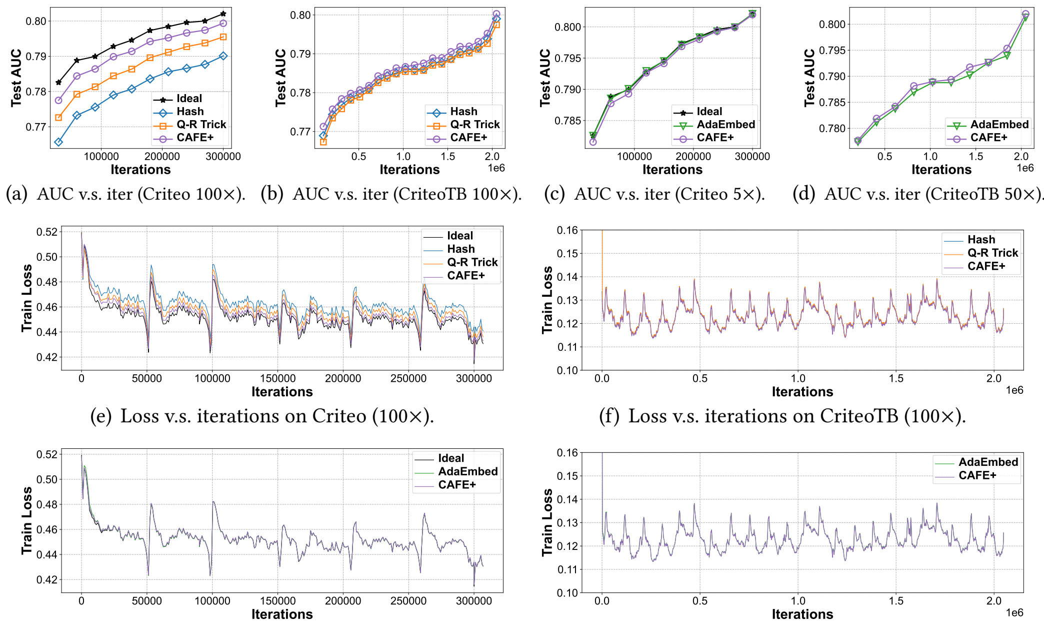
5.1.3 Hardware Environment. We conduct all experiments on NVIDIA RTX TITAN 24 GB GPU cards. Since this paper focuses on embedding compression with large compression ratios, we do not incur distributed training or inference.

[13]https://github.com/layer6ai-labs/HGCF
[14]https://github.com/asmitapoddar/nowplaying-RS-Music-Reco-FM



(a) AUC v.s. CR on Criteo. (b) AUC v.s. CR on CriteoTB. (c) Loss v.s. CR on Criteo. (d) Loss v.s. CR on CriteoTB.

Fig. 10. Metrics v.s. compression ratios.



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

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

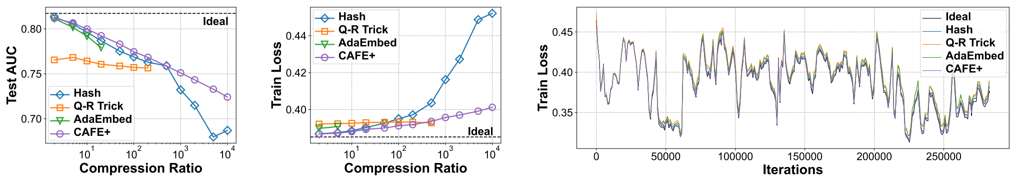
(g) Loss v.s. iterations on Criteo (5×). (h) Loss v.s. iterations on CriteoTB (50×).

Fig. 11. Metrics v.s. iterations.

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 (including the feature monitor on CPU memory) to achieve a fair judgment on memory efficiency. We use latency and throughput to measure the speed of each method. For the top-N recommendation tasks in Section 5.6-5.7, we use *NDCG@N* [92] and *Recall@N* as evaluation metrics with N set to $\{10, 20\}$. These two metrics are widely used in recent researches on top-N recommendation [29, 49–51, 77].

5.2 End-to-end Comparison

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.

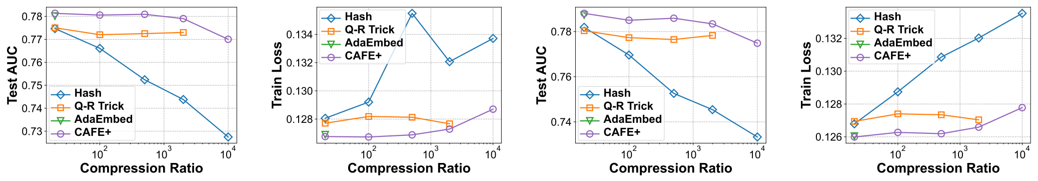


(a) AUC v.s. CR on KDD12. (b) Loss v.s. CR on Avazu. (c) Loss v.s. iterations on Avazu (5×).

Fig. 12. Performance on KDD12 and Avazu.

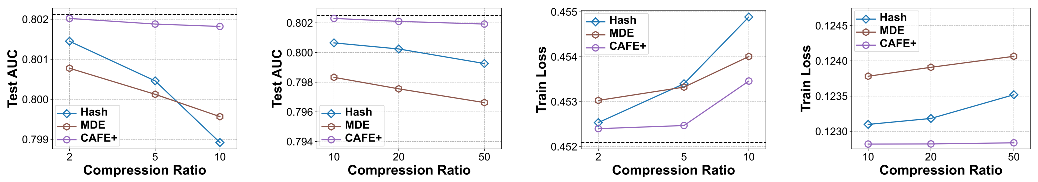
5.2.1 Metrics v.s. Compression Ratios. We conduct the main experiments on DLRM. The testing AUC and the training loss of Criteo and CriteoTB under different compression ratios are plotted in Figure 10, representing the performance of offline and online training respectively. For KDD12, we only plot the testing AUC in Figure 12(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 12(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 1.74% and 0.50% compared to Hash and Q-R Trick respectively on average; on CriteoTB dataset the improvement is 1.385% and 0.483%; on KDD12 dataset the improvement is 1.94% and 3.84%. CAFE+ also reduces the training loss by 2.36%, 0.74% on Criteo dataset, 1.44%, 0.612% on CriteoTB dataset, and 3.36%, 0.78% 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 3.94%, 3.94%, and 5.39% testing AUC on Criteo, CriteoTB, KDD12; CAFE+ reduces 4.65%, 3.54%, and 11.26% 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 0.06% testing AUC and a decrease of 0.13% training loss on CriteoTB dataset, achieves an increase of 0.84% testing AUC on KDD12 dataset and a decrease of 0.85% 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 11 shows the metrics on Criteo and CriteoTB throughout iterations during training. Figure 12(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 11(a)-11(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 11(e)-11(h), and 12(c), the training loss curves fluctuate due to changes in



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

Fig. 13. WDL and DCN performance on CriteoTB.



(a) AUC v.s. CR on Criteo. (b) AUC v.s. CR on CriteoTB. (c) Loss v.s. CR on Criteo. (d) Loss v.s. CR on CriteoTB.

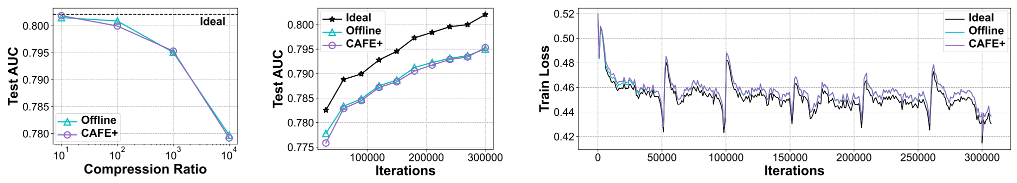
Fig. 14. Comparison with MDE.

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.

5.2.3 Experiments on WDL and DCN. We use another two models, WDL [8] and DCN [88], to experiment on the extremely large-scale dataset CriteoTB. The results are shown in Figure 13. 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.

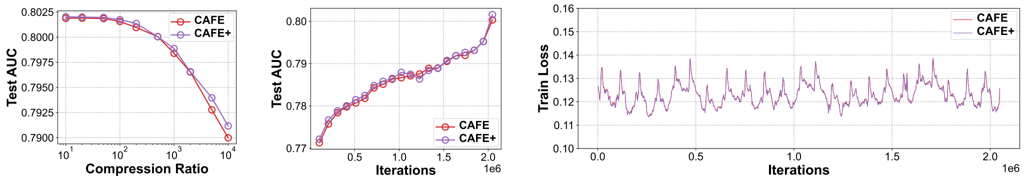
5.2.4 Comparison with Column Compression. We also compare CAFE+ with MDE [20], a method that compresses columns of embedding tables instead of rows as in CAFE+ and other baselines. It introduces frequency information to allocate different embedding dimensions for different features, and then uses a trainable 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 14. 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 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



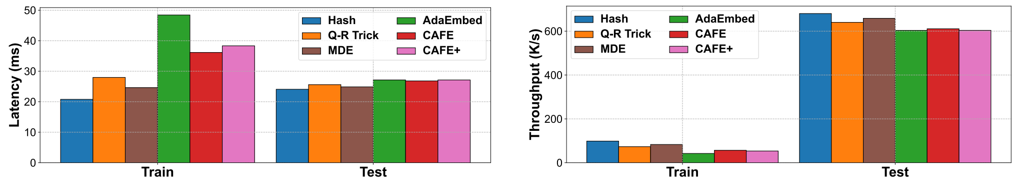
(a) AUC v.s. CR. (b) AUC v.s. iter (1000×). (c) Loss v.s. iterations (1000×).

Fig. 15. CAFE+ v.s. offline feature separation on Criteo dataset.



(a) AUC v.s. CR. (b) AUC v.s. iter. (1000×). (c) Loss v.s. iterations (1000×).

Fig. 16. CAFE v.s. CAFE+ on CriteoTB dataset.



(a) Latency. (b) Throughput.

Fig. 17. Latency and throughput on CriteoTB (10×).

embedding tables respectively. It uses the same embedding memory as in CAFE+ for hot and non-hot features. As shown in Figure 15(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 15(b) and Figure 15(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 supports online and offline training without storing all importance information, so it can be directly applied in the industry.

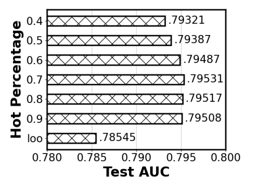
5.2.6 Comparison of CAFE and CAFE+. We compare the basic CAFE with the optimized CAFE (denoted as CAFE+) using the techniques of ColdSifter (Section 3.5), adaptive threshold adjustment (Section 3.6), and implicit importance attenuation (Section 3.7). To better showcase CAFE+'s ability to adapt to the dynamic variation in data distribution, we use the largest dataset, CriteoTB, whose data distribution changes the most drastically. As shown in Figure 16(a), CAFE+ achieves $0.03\% \sim 0.16\%$ improvement on testing AUC compared to the basic CAFE, and the advantage of CAFE+ becomes more apparent under larger compression ratios. In Figure 16(b)-16(c), we fix the compression ratio to 1000× and observe the testing AUC throughout the training process. We can see that CAFE+ finally achieves about 0.15% testing AUC improvement over basic CAFE. These results indicate that CAFE+ can effectively enhance the model quality of CAFE. We will further discuss the effect of the specific optimizations of CAFE+ in Section 5.3.3-5.3.5.

5.2.7 Latency and Throughput. We test the latency and throughput of each method in Figure 17. The experiments are 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 is also fast, 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 low latency and high throughput. AdaEmbed and CAFE incur reallocation or migration of embeddings, which are inevitable for dynamic adjustments, leading to higher latency and lower throughput. AdaEmbed regularly samples thousands of data to determine whether to reallocate, which introduces a large time overhead. By contrast, CAFE automatically identifies hot features using HotSketch in real time. Compared to AdaEmbed, CAFE has 33.97% lower training latency and 1.20% lower inference latency. Through the further experimental results in Section 5.5, we can see that HotSketch's $O(1)$ operation time only accounts for a small fraction of the overall time. We also evaluate the training and inference speed of CAFE+. The results show that the training speed of CAFE+ is slightly slower than CAFE (5.7%) but also significantly faster than AdaEmbed (26.4%), and the inference speed of CAFE+ is slightly slower than CAFE (1.1%) and similar to AdaEmbed.

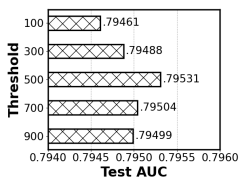
5.3 End-to-end Configuration Sensitivity Study

In this section, we study the impact of configurations on CAFE+ (and CAFE). By default, we conduct the configuration sensitivity studies on Criteo dataset, and we use basic CAFE with a fixed compression ratio of 1000×.

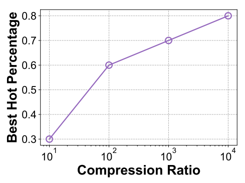
5.3.1 Memory Allocation for Hot Features. We evaluate the impact of memory allocation on model quality. We define the term "hot percentage" as the percentage of memory occupied by the feature monitor (HotSketch) and the embeddings of hot features, while the rest is the embeddings for non-hot features. Recall that by default, HotSketch stores 4 times the slots of the number of hot features (Section 4). As each slot stores 3 attributes, the ratio of memory usage between HotSketch and the d-dimension exclusive embeddings for hot features is $12 : d$. In Criteo dataset, the dimension is $d = 16$, so HotSketch occupies $3/7$ of memory in hot percentage. Figure 18(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 means small memory usage for hot features (and HotSketch), and more memory usage for non-hot features, while a large hot percentage means allocating more memory for hot features. When hot percentage is small, enlarging hot percentage enables more space resources allocated to hot features, contributing to model quality. When hot percentage is about 0.7, CAFE reaches the best testing AUC. When hot percentage exceeds 0.7, enlarging hot percentage reduces model quality because the space for non-hot features is excessively squeezed, and it introduces unnecessarily high overhead for HotSketch. At the extreme case "leave-one-out", all the non-hot features share only one embedding, leading to very bad model performance. We further evaluate the optimal value of hot percentage at different compression ratios in Figure 18(c). The results show that the optimal value of hot percentage increases as the compression ratio increases. In practice, we can configure the hot percentage based on the results shown in Figure 18(c). For example, when the compression ratio is 1000×, we recommend setting the hot percentage to 0.7.



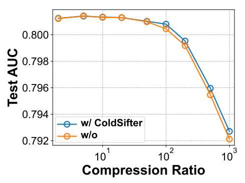
(a) Memory for hot features.



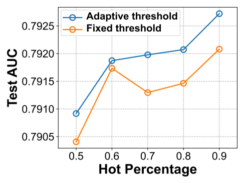
(b) Threshold of hot features.



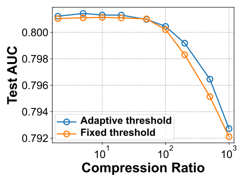
(c) Best hot percentage v.s. *CR*.



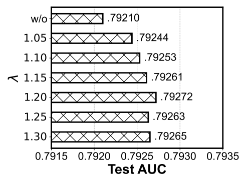
(d) ColdSifter.



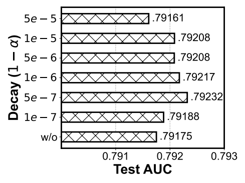
(e) Adaptive threshold.



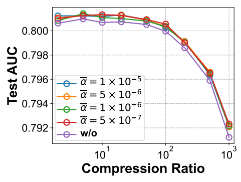
(f) Adaptive threshold.



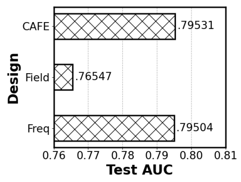
(g) Impact of threshold adjustment parameter (λ).



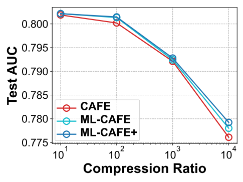
(h) Implicit attenuation.



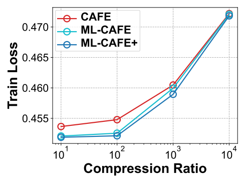
(i) Implicit attenuation.



(j) Other design details.



(k) Multi-layer Hash Emb.



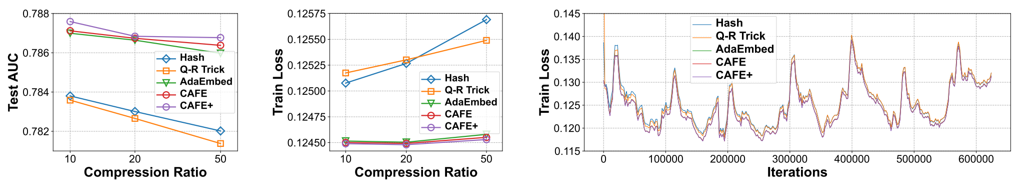
(l) Multi-layer Hash Emb.

Fig. 18. Experiments of configuration sensitivity on Criteo dataset (default $CR = 1000\times$).

5.3.2 Threshold of Hot Features (without Adaptive Threshold Adjustment). We evaluate the impact of hot feature thresholds on model quality (without using the adaptive threshold adjustment mechanism in Section 3.6), and the experimental results are shown in Figure 18(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.

5.3.3 Filter Cold Features with ColdSifter. We empirically evaluate the effectiveness of optimizing the feature monitor by filtering cold features with ColdSifter. In Figure 18(d), we compare the testing AUC of CAFE with and without the ColdSifter optimization, where we fix the total memory of the feature monitor to be the same. The results show that after using ColdSifter, the testing AUC of CAFE improves by approximately 0.05%. This is because after using the ColdSifter optimization, the feature monitor of CAFE can identify hot features with higher accuracy, allowing the uncompressed embedding table to be utilized more efficiently. We will further study the accuracy of the optimized feature monitor in identifying hot features in Section 5.5.

5.3.4 Adaptive Threshold Adjustment. We evaluate the effectiveness of the adaptive threshold adjustment mechanism, and the results are shown in Figure 18(e)-18(f). In Figure 18(e), we fix the compression ratio to 1000× and vary the hot percentage. The results show that the adaptive threshold adjustment mechanism improves the testing AUC by 0.04% $\sim$ 0.09% relatively. In Figure 18(f), we vary the compression ratio and use larger hot percentage (0.3 $\sim$ 0.9) for larger compression ratio. The results show that the adaptive threshold adjustment improves the testing AUC by 0.04% $\sim$ 0.18% relatively. We can see that our adaptive threshold adjustment mechanism can



(a) AUC v.s. CR. (b) Loss v.s. CR. (c) Loss v.s. iterations (50×).

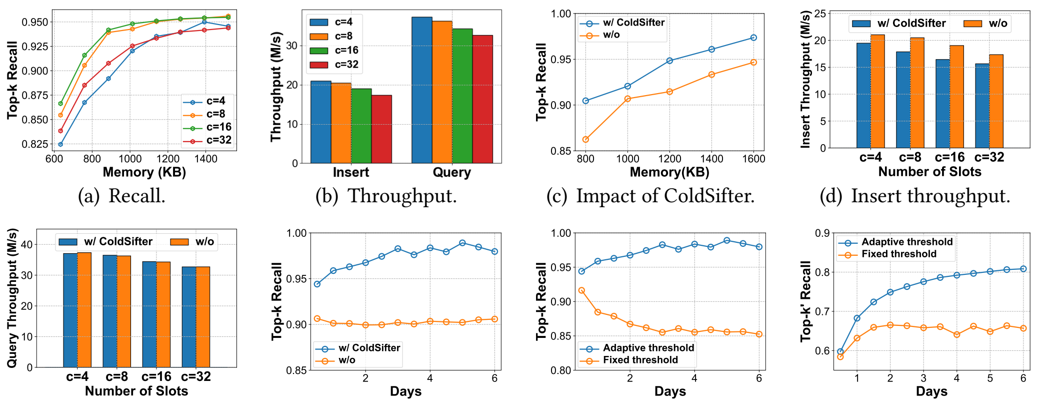
Fig. 19. Experiments on CriteoTB-1/3.

effectively improve model quality. As discussed in Section 3.6, this is because the optimal threshold changes over time, and the adaptive threshold adjustment can always set the threshold to the nearly optimal value according to current data distribution. Figure 18(g) further shows the impact of the threshold adjustment parameter λ on testing AUC. Parameter λ controls the frequency of our adaptive threshold adjustment operation. A large λ may result in delayed threshold adjustment. By contrast, a small λ could cause some features to oscillate between hot features and non-hot features, leading to frequent embedding migrations. Our results show that the optimal value of λ is about 1.2. Therefore, we set $\lambda = 1.2$ in our experiments by default.

5.3.5 *Implicit Importance Attenuation.* We evaluate the effectiveness of the implicit importance attenuation mechanism, and the results are shown in Figure 18(h)-18(i). In Figure 18(h), we evaluate the impact of decay factor α. The results show that the decay factor of $\alpha = 1 - 5e^{-7}$ achieves the best testing AUC, which is 0.072% higher than the basic CAFE without the importance attenuation mechanism. We can see that the decay factor should neither be too large nor too small. A decay factor that is too small prevents the importance score of hot features from accumulating in HotSketch, leading to hot features being misidentified as non-hot features. Conversely, a decay factor that is too large hinders CAFE from catching up with the changing data distribution over time, causing features to continuously occupy slots in HotSketch even if they are no longer hot features. In Figure 18(i), we evaluate the effectiveness of the implicit attenuation mechanism under different compression ratio. The results show that the implicit attenuation mechanism improves the testing AUC by about 0.053% relatively, and the optimal decay factor is also $\alpha = 1 - 5e^{-7}$ ($\overline{\alpha} = 1 - \alpha$).

5.3.6 *Design Details of Embedding Tables and Importance Scores.* We experiment on the design details of the embedding tables. As discussed in Section 4, we maintain only one exclusive embedding table for all fields, instead of maintaining one embedding table per field. Figure 18(j) 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 scores, which shows a worse testing AUC than gradient norms. Therefore, although frequency is also a good indicator of feature importance, it has been proved theoretically and experimentally that gradient norm is better.

5.3.7 *Multi-layer Hash Embedding.* We study the effect of multi-layer hash embedding. The experimental results are shown in Figure 18(k)-18(l), where ML-CAFE means CAFE with multi-layer hash embedding. Under different compression ratios, ML-CAFE always performs better than CAFE, achieving 0.08% better testing AUC and reducing 0.25% training loss. ML-CAFE performs especially well with smaller compression ratios, causing nearly no degradation at 100× compression ratio. This is because ML-CAFE allocates more memory for multi-layer hash embedding tables at small compression ratios, making the representation of medium features more precise. We also evaluate the performance of ML-CAFE with the three optimizations in Section 3.5-3.7, which is denoted as "ML-CAFE+" in Figure 18(k)-18(l). We can see that the three optimizations can further improve the testing AUC of ML-CAFE by 0.15% and reduce the training loss of ML-CAFE by 0.2%.



(a) Recall. (b) Throughput. (c) Impact of ColdSifter. (d) Insert throughput.

(e) Query throughput. (f) Top-k recall vs. days. (g) Top-k recall vs. days. (h) Top-k' recall vs. days.

Fig. 20. Experiments on feature monitor (Criteo).

5.4 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, $\cdots$, 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 19. Although all methods exhibit slight performance degradation compared to CriteoTB, CAFE+ (CAFE) and AdaEmbed can adapt to changing data distributions and achieve relatively good results. Figure 19(c) shows that CAFE+ (CAFE) and AdaEmbed have almost the same training loss throughout the training process. Figure 19(a) and 19(b) indicate that CAFE outperforms AdaEmbed, and CAFE+ further outperforms CAFE because of stronger adaptability to online training. Under 10× compression ratio, CAFE+ achieves 0.060% and 0.076% improvement on testing AUC over CAFE and AdaEmbed, respectively.

5.5 Performance of Feature Monitor

In this section, we evaluate the performance of the feature monitor in CAFE+. The experiments are conducted on the Criteo dataset, whose feature importance distribution is similar to the Zipf distribution of parameter 1.05 to 1.1. We set the compression ratio to 1000×, hot percentage to 0.7 by default. As discussed in Section 4, we set the number of hot features k to the number of buckets w in HotSketch. We set the number of hot and medium features to $k' = 3k$.

Impact of the number of slots per bucket (Figure 20(a)-20(b)): We evaluate the recall and the throughput of HotSketch with different number of slots per bucket. As shown in Figure 20(a), the recall rate increases as the memory usage of HotSketch 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. From Figure 20(a), we can see that $c = 8$ and $c = 16$ indeed exhibit a better recall than $c = 4$ and $c = 32$. These experimental results are consistent with our theoretical results in Corollary 3.5. The throughput of serialized Insert (write) and Query (read) of HotSketch is shown in Figure 20(b). We can see that the throughput of HotSketch is on the order of $1e7$, larger than that of DLRM. Considering that we can parallelize operations in practice, the time used by HotSketch only accounts for a small fraction in training and inference. In addition, throughput drops as the number of slots increases, because more time is spent doing comparisons within buckets. In our default

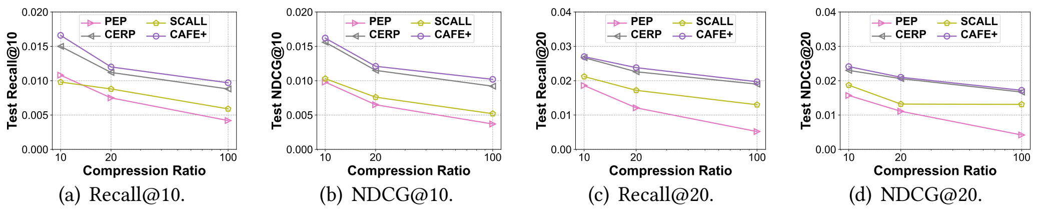


Fig. 21. Performance comparison on top-N recommendation (based on LightGCN [29]).

implementation, we adopt 4 slots per bucket to strike a tradeoff between recall and throughput, and we find this setting is enough for good model quality.

Impact of the ColdSifter optimization (Figure 20(c)-20(e)): We evaluate the performance gain of ColdSifter by comparing the top-k features recall rate of HotSketch with and without the ColdSifter optimization. We ensure that both solutions have the same total memory usage. As shown in Figure 20(c), after filtering cold features with ColdSifter, the recall rate of HotSketch improves 1.44% $\sim$ 4.90%, validating the effectiveness of ColdSifter. From Figure 20(d)-20(e), we can see that after using the ColdSifter optimization, the insert throughput and query throughput slightly drops by 7.4% $\sim$ 12.7% and 0% $\sim$ 0.7% respectively. This is because ColdSifter introduces additional memory accesses. Nevertheless, as discussed in Section 5.2.7, the end-to-end training speed of CAFE+ is still significantly faster than that of AdaEmbed.

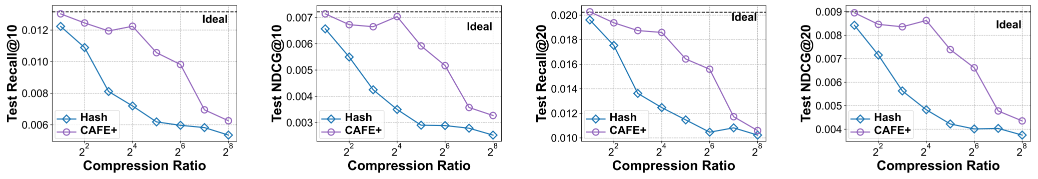
Finding real-time top-k features under implicit importance attenuation (Figure 20(f)): We evaluate the performance of our feature monitor on finding real-time hot features during online training, where we enable the implicit importance attenuation mechanism and set $\alpha = 1 - 5e^{-7}$. In Figure 20(f), we can see that the recall rate is always above 90%, meaning that CAFE+ can well catch up with the changing data distribution. As the real-time top-k features can change with data distribution during the online training process, these results can effectively reflect CAFE+'s capability to adapt to dynamic workloads. In addition, after using ColdSifter, the recall rate further improves 4.89% $\sim$ 9.64%, proving the effectiveness of the ColdSifter optimization.

Impact of adaptive threshold adjustment (Figure 20(g)-20(h)): We evaluate the performance gain of the adaptive threshold adjustment optimization on finding real-time hot/medium features during online training. We also enable the implicit importance attenuation mechanism and set $\alpha = 1 - 5e^{-7}$. From Figure 20(g) and Figure 20(h), we can see that the adaptive threshold adjustment optimization improves the recall rate of finding top-k hot features and top-k' hot and medium features by 3.06% $\sim$ 14.91% and 2.22% $\sim$ 23.10%, respectively. These results show that the adaptive threshold adjustment mechanism can effectively improve the accuracy of HotSketch by setting its thresholds to appropriate values.

5.6 End-to-end Comparison with Baselines on Top-N Recommendation

We compare CAFE+ with three recent embedding compression frameworks for top-N recommendation: PEP [51], CERP [50], and SCALL [77] (described in Section 5.1.2). Following the papers of CERP and SCALL, we use LightGCN (described in Section 5.1.1) as base recommender, use Yelp2020 (described in Section 5.1.1) as dataset, and use $NDCG@N$ [92] and $Recall@N$ scores as evaluation metrics (described in Section 5.1.4).

As shown in Figure 21, across all compression ratios (10×, 20×, 100×), CAFE+ consistently outperforms the three baselines. For example, at the compression ratio of 10×, compared with PEP, CERP, SCALL, CAFE+ achieves 53.70%, 10.67%, and 69.39% improvement on $Recall@10$ (Figure 21(a)), and 65.30%, 3.84%, and 57.28% improvement on $NDCG@10$ (Figure 21(b)). We think the reason why CAFE+ performs better is because it can continuously optimize the embedding allocation



(a) Recall@10. (b) NDCG@10. (c) Recall@20. (d) NDCG@20.

Fig. 22. Performance in graph-based recommender system (based on PinSage [106]).

throughout the entire training process and dynamically adapts to current embedding allocation at runtime. In contrast, post-training pruning schemes PEP and CERP prune the embedding table after training, and SCALL periodically adjusts the embedding allocation. Although the three baselines retrain the model under the final embedding table, they cannot continuously optimize and adapt the embedding allocation throughout the training process. This reason is also discussed in the paper of AdaEmbed [42], which also reports that in-training embedding pruning performs better than post-training pruning counterparts.

In addition, all the three baselines (PEP, CERP, and SCALL) require retraining on the pruned or re-configured embedding table, resulting in more computation consumption. As all the three baselines require maintaining additional structures (pruning masks, RL model) during training for embedding pruning/reallocation, they cannot achieve memory savings during training, and have more hyperparameters to be manually adjusted. Finally, the two post-training pruning schemes (PEP and CERP) cannot adapt to the requirement of online training (defined in Section 2), and thus are often impractical to be deployed in practice as reported by the paper of AdaEmbed [42].

5.7 Performance on Graph-based Recommendation System

To demonstrate the wide adaptability of CAFE+, we further implement it in a graph-based recommendation model PinSage [106] (described in Section 5.1.1), which is widely deployed in the industry for large-scale applications. We conduct experiments on a larger dataset #nowplaying-RS (described in Section 5.1.1), and use *NDCG@N* [92] and *Recall@N* scores as evaluation metrics (described in Section 5.1.4). We compare the performance of CAFE+ with basic hash embedding. As shown in Figure 22, CAFE+ works well on top of PinSage and consistently outperforms basic hash embedding across all compression ratios (2×∼256×). For example, at the compression ratio of 16×, compared with hash embedding, CAFE+ achieves 70.14% improvement on *Recall*@10 (Figure 22(a)) and 101.15% improvement on *NDCG*@10 (Figure 22(b)). When compression ratio is less than 16×, the performance of CAFE+ is close to the ideal result that uses uncompressed embedding tables.

6 RELATED WORK

6.1 Embedding Compression

Recent years have witnessed numerous techniques proposed for compressing embedding tables, which can be broadly divided into two categories: inter-feature compression and intra-feature compression [115]. Inter-feature compression methods (including CAFE+) reduce the number of embeddings and allow embedding sharing among multiple features. Intra-feature compression methods compress individual embeddings through quantization, pruning, and dimension reduction, thereby reducing the size of each embedding. Since two categories are primarily orthogonal, methods of different categories can be further combined in DLRMs. For more related works and specific experimental comparison results, please refer to an earlier experimental analysis paper on embedding compression [115].

Inter-feature compression: These methods keep a small number of embeddings for features to share, and maintain a new mapping from features to embeddings. Based on whether the mapping relationship is fixed during training, these methods can be further divided into static inter-feature compression and dynamic inter-feature compression.

1) Static inter-feature compression: Initial attempts to accommodate large numbers of embeddings within a limited memory space came from hash-based compositional embedding methods [81, 94, 101], which are widely used in real-world applications. They use hash functions to map features into several compositional embeddings, resulting in different features sharing (or partially sharing) the same embedding in the event of hash collisions. However, these methods do not provide theoretical bounds, which can lead to significant degradation in model quality. While early works explored the form of the mapping functions, recent works further explored the structure of embedding layers. For example, some excellent works borrow the idea of tensor-train decomposition (abbreviated as TT) to compress the embedding table [96, 104, 105, 130]. These works factorize the original size of embedding table $m \times d$ into $m = \prod_{i=1}^{t} m_i$ and $d = \prod_{i=1}^{t} d_i$, and decompose the embedding table to $\mathcal{E} = \mathcal{G}_1\mathcal{G}_2 \cdots \mathcal{G}_t$, where $\mathcal{G}_i \in \mathbb{R}^{R_{i-1} \times m_i \times d_i \times R_i}$. To obtain the embedding, TT-based work looks up the tensors and conducts matrix multiplication, which requires more computation overhead [115].

2) Dynamic inter-feature compression: These methods allow the mapping functions to be updated during training, most of which are naturally suitable for online learning [7, 42]. AdaEmbed [42] 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. CEL [7] compresses the embeddings through clustering. During training, CEL dynamically reassigns items to the more proper clusters based on their history interactions, and split a cluster if it is associated with too many interactions. CEL has limited compression ratios due to the need of storing the cluster structures, and takes more training time due to cluster adjustment. LEGCF [49] innovatively introduces a learnable assignment matrix on top of compositional embeddings. As LEGCF requires alternately updating the assignment matrix and the embeddings over multiple epochs, it cannot be deployed in online learning scenarios.

Intra-feature compression: Methods of this category aim to compress the representation for each unique feature, thereby reducing the size (number of bits) of each embedding. They borrow techniques from traditional deep learning compression such as quantization [44, 100], pruning [13, 39, 51, 75], and dimension reduction [20, 55, 125].

1) Quantization: These methods use low-prevision data types to replace the original float32 in the embedding table [44, 100]. Although stable and simple to use, these methods do not support large compression ratios [115]. Even using 4-bit data type, the compression ratio can only reach 8×.

2) Pruning: These methods reduce the amount of embedding parameters by finding a binary mask [13, 39, 51, 75] during training, and pruning the embedding table with this mask. Afterwards, they usually need to retrain the model to fit the sparse embeddings. PEP [51] is a post-training pruning scheme that adaptively learns the pruning thresholds and uses them to prune the embedding table. SSEDS [75] devises a saliency criterion to identify the importance of each embedding dimension for each field, and prunes the embeddings according to this criterion. As reported by AdaEmbed [42] and our results in Section 5.6, many post-training methods actually have lower accuracy than in-training methods (CAFE+, AdaEmbed). In addition, the multi-stage training process of post-training pruning methods is usually time-consuming. These methods also suffer large memory consumption during training, and can hardly adapt to online training scenarios.

3) Dimension reduction: The key idea of these methods is to assign different dimensions for different features. Similar to post-training pruning methods, dimension reduction methods compress the embedding tables only at inference time, and thus also require additional memory to store extra structures during training and multi-stage training. There are a line of excellent research that find the best parameter allocation through the techniques of Neural Architecture Search (NAS), including reinforcement learning [36, 76, 77], evolutionary search [6], bi-level optimizations [126, 127]. Consequently, these NAS-based approaches usually have high computation costs.

However, post-pruning and dimension reduction methods are actually seldom used in industry [42, 115], as the memory bottleneck during training is more severe due to activations and optimizer states. In addition, 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.

There are also some hybrid methods that combine various compression techniques to achieve better performance [50, 56], which is also a promising direction in the future. For example, CERP [50] innovatively combines static compositional embeddings with regularized pruning to achieve high model quality. We can also combine CAFE+ with various compression techniques. For example, the idea of tensor-train decomposition methods [96, 104, 105], clustering methods [7], and quantization methods [44, 100] can also be used to compress the *Hash Embedding Table* in CAFE+.

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 [11], database [9, 32, 53, 80], and network measurement and management [103, 118] to perform various tasks, such as frequency estimation [11, 12, 17], finding top-k frequent items [43, 59, 61, 102], and mining special patterns in streaming data [52]. Existing sketches can be classified into two categories: counter-based sketches and KV-based sketches.

Counter-based sketches: Typical counter-based sketches include CM [11], CU [17], Count [5], ASketch [78], and more [12, 18, 45, 69]. 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 [11] 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 [17] only increments the minimum counter among $C_1[h_1(e)], \cdots, C_d[h_d(e)]$, and Count sketch [5] 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 [61], Unbaised Space-Saving [84], Lossy Counting [14], HeavyGuardian [102], and more [43, 103, 118]. These sketches maintain the KV pairs of frequent items in their data structures. For instance, Space-Saving [61] and Unbiased Space-Saving [84] 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 [84] 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 [102] and WavingSketch [43], 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. 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. We further propose several techniques to further optimize the memory allocation CAFE+ and adapt it to dynamic data distribution during online training. Experimental results demonstrate that CAFE+ outperforms existing methods, with 3.94%, 3.94% higher testing AUC and 4.65%, 3.54% 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 [112].

ACKNOWLEDGMENTS

This work is supported by National Natural Science Foundation of China (NSFC) (No. 62372009, 623B2005, U22B2037, U23B2048), research grant No. IPT-2024JK29, PKU-Tencent joint research Lab, and High-performance Computing Platform of Peking University.

REFERENCES

[1] Aden and Yi Wang. 2012. KDD Cup 2012, Track 2. https://kaggle.com/competitions/kddcup2012-track2.

[2] Maristella Agosti, Stefano Marchesin, and Gianmaria Silvello. 2020. Learning unsupervised knowledge-enhanced representations to reduce the semantic gap in information retrieval. *ACM Transactions on Information Systems (TOIS)* 38, 4 (2020), 1–48.

[3] Zeyuan Allen-Zhu. 2017. Natasha: Faster Non-Convex Stochastic Optimization via Strongly Non-Convex Parameter. In *Proceedings of the 34th International Conference on Machine Learning (ICML)*.

[4] Da Cao, Liqiang Nie, Xiangnan He, Xiaochi Wei, Shunzhi Zhu, and Tat-Seng Chua. 2017. Embedding factorization models for jointly recommending items and user generated lists. In *Proceedings of the 40th international ACM SIGIR conference on research and development in information retrieval*. 585–594.

[5] Moses Charikar, Kevin C. Chen, and Martin Farach-Colton. 2002. Finding Frequent Items in Data Streams. In *Automata, Languages and Programming, 29th International Colloquium (ICALP)*.

[6] Tong Chen, Hongzhi Yin, Yujia Zheng, Zi Huang, Yang Wang, and Meng Wang. 2021. Learning elastic embeddings for customizing on-device recommenders. In *Proceedings of the 27th ACM SIGKDD Conference on Knowledge Discovery & Data Mining*. 138–147.

[7] Yizhou Chen, Guangda Huzhang, Anxiang Zeng, Qingtao Yu, Hui Sun, Heng-Yi Li, Jingyi Li, Yabo Ni, Han Yu, and Zhiming Zhou. 2023. Clustered Embedding Learning for Recommender Systems. In *Proceedings of the ACM Web Conference 2023*. 1074–1084.

[8] 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. 2016. Wide & Deep Learning for Recommender Systems. In *Proceedings of the 1st Workshop on Deep Learning for Recommender Systems (DLRS@RecSys)*.

[9] Monica Chiosa, Thomas Preußer, and Gustavo Alonso. 2021. SKT: A One-Pass Multi-Sketch Data Analytics Accelerator. *Proceedings of the VLDB Endowment* 14, 11 (2021), 2369–2382.

[10] Michael Chui, James Manyika, Mehdi Miremadi, Nicolaus Henke, Rita Chung, Pieter Nel, and Sankalp Malhotra. 2018. Notes from the AI frontier: Insights from hundreds of use cases. *McKinsey Global Institute* 2 (2018). https://www.mckinsey.com/west-coast/~/media/McKinsey/Featured%20Insights/Artificial%20Intelligence/Notes%

20from%20the%20AI%20frontier%20Applications%20and%20value%20of%20deep%20learning/Notes-from-the-AI-frontier-Insights-from-hundreds-of-use-cases-Discussion-paper.pdf

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

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

[13] Wei Deng, Junwei Pan, Tian Zhou, Deguang Kong, Aaron Flores, and Guang Lin. 2021. DeepLight: Deep Lightweight Feature Interactions for Accelerating CTR Predictions in Ad Serving. In *Proceedings of the 14th ACM International Conference on Web Search and Data Mining (WSDM).*

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

[15] Yue Ding, Yuhe Guo, Wei Lu, Hai-Xiang Li, Meihui Zhang, Hui Li, An-Qun Pan, and Xiaoyong Du. 2023. Context-Aware Semantic Type Identification for Relational Attributes. *Journal of Computer Science and Technology* 38, 4 (2023), 927–946.

[16] Muhammad Ebraheem, Saravanan Thirumuruganathan, Shafiq R. Joty, Mourad Ouzzani, and Nan Tang. 2018. Distributed Representations of Tuples for Entity Resolution. *Proceedings of the VLDB Endowment* 11, 11 (2018), 1454–1467.

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

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

[19] Fangcheng Fu, Yuzheng Hu, Yihan He, Jiawei Jiang, Yingxia Shao, Ce Zhang, and Bin Cui. 2020. Don't Waste Your Bits! Squeeze Activations and Gradients for Deep Neural Networks via TinyScript. In *Proceedings of the 37th International Conference on Machine Learning (ICML).*

[20] Antonio A. Ginart, Maxim Naumov, Dheevatsa Mudigere, Jiyan Yang, and James Zou. 2021. Mixed Dimension Embeddings with Application to Memory-Efficient Recommendation Systems. In *IEEE International Symposium on Information Theory (ISIT).*

[21] Siddharth Gopal. 2016. Adaptive Sampling for SGD by Exploiting Side Information. In *Proceedings of the 33nd International Conference on Machine Learning (ICML).*

[22] Huifeng Guo, Ruiming Tang, Yunming Ye, Zhenguo Li, and Xiuqiang He. 2017. DeepFM: A Factorization-Machine based Neural Network for CTR Prediction. In *Proceedings of the 26th International Joint Conference on Artificial Intelligence (IJCAI).*

[23] Lei Guo, Hongzhi Yin, Tong Chen, Xiangliang Zhang, and Kai Zheng. 2021. Hierarchical hyperedge embedding-based representation learning for group recommendation. *ACM Transactions on Information Systems (TOIS)* 40, 1 (2021), 1–27.

[24] Lei Guo, Hongzhi Yin, Qinyong Wang, Tong Chen, Alexander Zhou, and Nguyen Quoc Viet Hung. 2019. Streaming session-based recommendation. In *Proceedings of the 25th ACM SIGKDD international conference on knowledge discovery & data mining*. 1569–1577.

[25] Udit Gupta, Samuel Hsia, Vikram Saraph, Xiaodong Wang, Brandon Reagen, Gu-Yeon Wei, Hsien-Hsin S. Lee, David Brooks, and Carole-Jean Wu. 2020. 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).*

[26] 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. 2020. The Architectural Implications of Facebook's DNN-Based Personalized Recommendation. In *IEEE International Symposium on High Performance Computer Architecture (HPCA).*

[27] Teng-Yue Han, Pengfei Wang, and Shaozhang Niu. 2023. Multimodal Interactive Network for Sequential Recommendation. *Journal of Computer Science and Technology* 38, 4 (2023), 911–926.

[28] Peng-Yi Hao, Si-Hao Liu, and Cong Bai. 2024. Intent-Aware Graph-Level Embedding Learning Based Recommendation. *Journal of Computer Science and Technology* 39, 5 (2024), 1138–1152.

[29] Xiangnan He, Kuan Deng, Xiang Wang, Yan Li, Yongdong Zhang, and Meng Wang. 2020. Lightgcn: Simplifying and powering graph convolution network for recommendation. In *Proceedings of the 43rd International ACM SIGIR conference on research and development in Information Retrieval*. 639–648.

[30] Ruihong Huang, Shaoxu Song, Yunsu Lee, Jungho Park, Soo-Hyung Kim, and Sungmin Yi. 2020. Effective and Efficient Retrieval of Structured Entities. *Proceedings of the VLDB Endowment* 13, 6 (2020), 826–839.

[31] Intel. 2024. Intel instructions. https://software.intel.com/sites/landingpage/IntrinsicsGuide/.

[32] Yesdaulet Izenov, Asoke Datta, Florin Rusu, and Jun Hyung Shin. 2021. COMPASS: Online Sketch-based Query Optimization for In-Memory Databases. In *Proceedings of the International Conference on Management of Data*

(SIGMOD).
[33] Yitong Ji, Aixin Sun, Jie Zhang, and Chenliang Li. 2020. A re-visit of the popularity baseline in recommender systems. In *Proceedings of the 43rd International ACM SIGIR Conference on Research and Development in Information Retrieval.* 1749–1752.
[34] 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. 2019. 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.*
[35] Jiazheng Jing, Yinan Zhang, Xin Zhou, and Zhiqi Shen. 2023. Capturing Popularity Trends: A Simplistic Non-Personalized Approach for Enhanced Item Recommendation. In *Proceedings of the 32nd ACM International Conference on Information and Knowledge Management.* 1014–1024.
[36] Manas R Joglekar, Cong Li, Mei Chen, Taibai Xu, Xiaoming Wang, Jay K Adams, Pranav Khaitan, Jiahui Liu, and Quoc V Le. 2020. Neural input search for large scale recommendation models. In *Proceedings of the 26th ACM SIGKDD International Conference on Knowledge Discovery & Data Mining.* 2387–2397.
[37] Wang-Cheng Kang, Derek Zhiyuan Cheng, Ting Chen, Xinyang Yi, Dong Lin, Lichan Hong, and Ed H. Chi. 2020. Learning Multi-granular Quantized Embeddings for Large-Vocab Categorical Features in Recommender Systems. In *Companion Proceedings of The Web Conference.*
[38] Angelos Katharopoulos and François Fleuret. 2018. Not All Samples Are Created Equal: Deep Learning with Importance Sampling. In *Proceedings of the 35th International Conference on Machine Learning (ICML).*
[39] Shuming Kong, Weiyu Cheng, Yanyan Shen, and Linpeng Huang. 2023. AutoSrh: An Embedding Dimensionality Search Framework for Tabular Data Prediction. *IEEE Transactions on Knowledge and Data Engineering* 35, 7 (2023), 6673–6686.
[40] Criteo Labs. 2013. Download Terabyte Click Logs. https://labs.criteo.com/2013/12/download-terabyte-click-logs/.
[41] Criteo Labs. 2014. Kaggle display advertising challenge dataset. https://labs.criteo.com/2014/02/kaggle-display-advertising-challenge-dataset/.
[42] 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. 2023. AdaEmbed: Adaptive Embedding for Large-Scale Recommendation Models. In *17th USENIX Symposium on Operating Systems Design and Implementation (OSDI).*
[43] Jizhou Li, Zikun Li, Yifei Xu, Shiqi Jiang, Tong Yang, Bin Cui, Yafei Dai, and Gong Zhang. 2020. 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).*
[44] Shiwei Li, Huifeng Guo, Lu Hou, Wei Zhang, Xing Tang, Ruiming Tang, Rui Zhang, and Ruixuan Li. 2023. Adaptive Low-Precision Training for Embeddings in Click-Through Rate Prediction. In *Proceedings of the AAAI Conference on Artificial Intelligence (AAAI).*
[45] Tao Li, Shigang Chen, and Yibei Ling. 2012. Per-Flow Traffic Measurement Through Randomized Counter Sharing. *IEEE/ACM Transactions on Networking* 20, 5 (2012), 1622–1634.
[46] Yuanpeng Li, Feiyu Wang, Xiang Yu, Yilong Yang, Kaicheng Yang, Tong Yang, Zhuo Ma, Bin Cui, and Steve Uhlig. 2023. Ladderfilter: Filtering infrequent items with small memory and time overhead. *Proceedings of the ACM on Management of Data* 1, 1 (2023), 1–21.
[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. 2022. 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).*
[48] Shangsong Liang, Yupeng Luo, and Zaiqiao Meng. 2021. Profiling users for question answering communities via flow-based constrained co-embedding model. *ACM Transactions on Information Systems (TOIS)* 40, 2 (2021), 1–38.
[49] Xurong Liang, Tong Chen, Lizhen Cui, Yang Wang, Meng Wang, and Hongzhi Yin. 2024. Lightweight Embeddings for Graph Collaborative Filtering. In *Proceedings of the 47th International ACM SIGIR Conference on Research and Development in Information Retrieval.* 1296–1306.
[50] Xurong Liang, Tong Chen, Quoc Viet Hung Nguyen, Jianxin Li, and Hongzhi Yin. 2023. Learning compact compositional embeddings via regularized pruning for recommendation. In *2023 IEEE International Conference on Data Mining (ICDM).* IEEE, 378–387.
[51] Siyi Liu, Chen Gao, Yihong Chen, Depeng Jin, and Yong Li. 2021. Learnable embedding sizes for recommender systems. *ICLR* (2021).
[52] Zirui Liu, Chaozhe Kong, Kaicheng Yang, Tong Yang, Ruijie Miao, Qizhi Chen, Yikai Zhao, Yaofeng Tu, and Bin Cui. 2023. HyperCalm Sketch: One-Pass Mining Periodic Batches in Data Streams. In *39th IEEE International Conference*

on Data Engineering (ICDE).

[53] Zirui Liu, Yixin Zhang, Yifan Zhu, Ruwen Zhang, Tong Yang, Kun Xie, Sha Wang, Tao Li, and Bin Cui. 2023. TreeSensing: Linearly Compressing Sketches with Flexibility. In *Proceedings of the International Conference on Management of Data (SIGMOD).*

[54] Mary Loxton, Robert Truskett, Brigitte Scarf, Laura Sindone, George Baldry, and Yinong Zhao. 2020. Consumer behaviour during crises: Preliminary research on how coronavirus has manifested consumer panic buying, herd mentality, changing discretionary spending and the role of the media in influencing behaviour. *Journal of risk and financial management* 13, 8 (2020), 166.

[55] Fuyuan Lyu, Xing Tang, Hong Zhu, Huifeng Guo, Yingxue Zhang, Ruiming Tang, and Xue Liu. 2022. OptEmbed: Learning Optimal Embedding Table for Click-through Rate Prediction. In *Proceedings of the 31st ACM International Conference on Information & Knowledge Management (CIKM).*

[56] Fuyuan Lyu, Xing Tang, Hong Zhu, Huifeng Guo, Yingxue Zhang, Ruiming Tang, and Xue Liu. 2022. Optembed: Learning optimal embedding table for click-through rate prediction. In *Proceedings of the 31st ACM International Conference on Information & Knowledge Management.* 1399–1409.

[57] Kaihao Ma, Xiao Yan, Zhenkun Cai, Yuzhen Huang, Yidi Wu, and James Cheng. 2023. FEC: Efficient Deep Recommendation Model Training with Flexible Embedding Communication. *Proceedings of the ACM on Management of Data* 1, 2 (2023), 1–21.

[58] Sean MacAvaney, Andrew Yates, Arman Cohan, and Nazli Goharian. 2019. CEDR: Contextualized embeddings for document ranking. In *Proceedings of the 42nd international ACM SIGIR conference on research and development in information retrieval.* 1101–1104.

[59] Ankush Mandal, He Jiang, Anshumali Shrivastava, and Vivek Sarkar. 2018. Topkapi: Parallel and Fast Sketches for Finding Top-K Frequent Elements. In *Advances in Neural Information Processing Systems 31 (NeurIPS).*

[60] Xiangfu Meng, Hongjin Huo, Xiaoyan Zhang, Wanchun Wang, and Jinxia Zhu. 2023. A Survey of Personalized News Recommendation. *Data Science and Engineering* 8, 4 (2023), 396–416.

[61] Ahmed Metwally, Divyakant Agrawal, and Amr El Abbadi. 2005. Efficient Computation of Frequent and Top-k Elements in Data Streams. In *International Conference on Database Theory.*

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

[63] Xupeng Miao, Hailin Zhang, Yining Shi, Xiaonan Nie, Zhi Yang, Yangyu Tao, and Bin Cui. 2022. HET: Scaling out Huge Embedding Model Training via Cache-enabled Distributed Framework. *Proceedings of the VLDB Endowment* 15, 2 (2022), 312–320.

[64] 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. 2022. 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).*

[65] 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. 2020. Deep Learning Training in Facebook Data Centers: Design of Scale-up and Scale-out Systems. *CoRR* abs/2003.09518 (2020).

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

[67] Liqiang Nie, Yongqi Li, Fuli Feng, Xuemeng Song, Meng Wang, and Yinglong Wang. 2020. Large-scale question tagging via joint question-topic embedding learning. *ACM Transactions on Information Systems (TOIS)* 38, 2 (2020), 1–23.

[68] Niketan Pansare, Jay Katukuri, Aditya Arora, Frank Cipollone, Riyaaz Shaik, Noyan Tokgozoglu, and Chandru Venkataraman. 2022. Learning Compressed Embeddings for On-Device Inference. In *Proceedings of Machine Learning and Systems (MLSys).*

[69] Guillaume Pitel and Geoffroy Fouquier. 2015. Count-Min-Log sketch: Approximately counting with approximate counters. In *International Symposium on Web AlGorithms.*

[70] NVIDIA AI platform. 2020. MLPerf Benchmark. https://mlperf.org.
[71] Asmita Poddar, Eva Zangerle, and Yi-Hsuan Yang. 2018. nowplaying-RS: A New Benchmark Dataset for Building Context-Aware Music Recommender Systems. In *Proceedings of the 15th Sound Music Computing Conference* (2018-07-04). Limassol, Cyprus. http://mac.citi.sinica.edu.tw/~yang/pub/poddar18smc.pdf code at https://github.com/asmitapoddar/nowplaying-RS-Music-Reco-FM.
[72] Xue-Yang Qin, Li-Shuang Li, Jing-Yao Tang, Fei Hao, Mei-Ling Ge, and Guang-Yao Pang. 2024. Multi-task visual semantic embedding network for image-text retrieval. *Journal of Computer Science and Technology* 39, 4 (2024), 811–826.
[73] Ruihong Qiu, Hongzhi Yin, Zi Huang, and Tong Chen. 2020. Gag: Global attributed graph neural network for streaming session-based recommendation. In *Proceedings of the 43rd international ACM SIGIR conference on research and development in information retrieval.* 669–678.
[74] Chen Qu, Liu Yang, Minghui Qiu, W Bruce Croft, Yongfeng Zhang, and Mohit Iyyer. 2019. BERT with history answer embedding for conversational question answering. In *Proceedings of the 42nd international ACM SIGIR conference on research and development in information retrieval.* 1133–1136.
[75] Liang Qu, Yonghong Ye, Ningzhi Tang, Lixin Zhang, Yuhui Shi, and Hongzhi Yin. 2022. Single-shot embedding dimension search in recommender system. In *Proceedings of the 45th International ACM SIGIR conference on research and development in Information Retrieval.* 513–522.
[76] Yunke Qu, Tong Chen, Xiangyu Zhao, Lizhen Cui, Kai Zheng, and Hongzhi Yin. 2023. Continuous input embedding size search for recommender systems. In *Proceedings of the 46th International ACM SIGIR Conference on Research and Development in Information Retrieval.* 708–717.
[77] Yunke Qu, Liang Qu, Tong Chen, Xiangyu Zhao, Quoc Viet Hung Nguyen, and Hongzhi Yin. 2024. Scalable Dynamic Embedding Size Search for Streaming Recommendation. In *Proceedings of the 33rd ACM International Conference on Information and Knowledge Management.* 1941–1950.
[78] Pratanu Roy, Arijit Khan, and Gustavo Alonso. 2016. Augmented Sketch: Faster and More Accurate Stream Processing. In *Proceedings of the International Conference on Management of Data (SIGMOD).*
[79] Pengyang Shao, Le Wu, Lei Chen, Kun Zhang, and Meng Wang. 2022. FairCF: fairness-aware collaborative filtering. *Science China Information Sciences* 65, 12 (2022).
[80] Benwei Shi, Zhuoyue Zhao, Yanqing Peng, Feifei Li, and Jeff M. Phillips. 2021. At-the-time and Back-in-time Persistent Sketches. In *Proceedings of the International Conference on Management of Data (SIGMOD).*
[81] Hao-Jun Michael Shi, Dheevatsa Mudigere, Maxim Naumov, and Jiyan Yang. 2020. Compositional Embeddings Using Complementary Partitions for Memory-Efficient Recommendation Systems. In *Proceedings of the 26th ACM SIGKDD Conference on Knowledge Discovery & Data Mining (KDD).*
[82] Harald Steck. 2011. Item popularity and recommendation accuracy. In *Proceedings of the fifth ACM conference on Recommender systems.* 125–132.
[83] Jianing Sun, Zhaoyue Cheng, Saba Zuberi, Felipe Pérez, and Maksims Volkovs. 2021. Hgcf: Hyperbolic graph convolution networks for collaborative filtering. In *Proceedings of the Web Conference 2021.* 593–601.
[84] Daniel Ting. 2018. Data Sketches for Disaggregated Subset Sum and Frequent Item Estimation. In *Proceedings of the International Conference on Management of Data (SIGMOD).*
[85] Corinna Underwood. 2019. Use cases of recommendation systems in business–current applications and methods. *Emerj* (2019). https://emerj.com/ai-sector-overviews/use-cases-recommendation-systems/
[86] Jizhe Wang, Pipei Huang, Huan Zhao, Zhibo Zhang, Binqiang Zhao, and Dik Lun Lee. 2018. Billion-scale commodity embedding for e-commerce recommendation in alibaba. In *Proceedings of the 24th ACM SIGKDD international conference on knowledge discovery & data mining.* 839–848.
[87] Qinyong Wang, Hongzhi Yin, Zhiting Hu, Defu Lian, Hao Wang, and Zi Huang. 2018. Neural memory streaming recommender networks with adversarial training. In *Proceedings of the 24th ACM SIGKDD International Conference on Knowledge Discovery & Data Mining.* 2467–2475.
[88] Ruoxi Wang, Bin Fu, Gang Fu, and Mingliang Wang. 2017. Deep & Cross Network for Ad Click Predictions. In *Proceedings of the ADKDD'17.*
[89] Steve Wang and Will Cukierski. 2014. Avazu Click-Through Rate Prediction. https://kaggle.com/competitions/avazu-ctr-prediction.
[90] Weiqing Wang, Hongzhi Yin, Zi Huang, Qinyong Wang, Xingzhong Du, and Quoc Viet Hung Nguyen. 2018. Streaming ranking based recommender systems. In *The 41st International ACM SIGIR Conference on Research & Development in Information Retrieval.* 525–534.
[91] Xiang Wang, Xiangnan He, and Tat-Seng Chua. 2020. Learning and reasoning on graph for recommendation. In *Proceedings of the 13th international conference on web search and data mining.* 890–893.
[92] Yining Wang, Liwei Wang, Yuanzhi Li, Di He, Wei Chen, and Tie-Yan Liu. 2013. A theoretical analysis of ndcg ranking measures. In *Proceedings of the 26th annual conference on learning theory.*

[93] 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. 2022. Merlin HugeCTR: GPU-accelerated Recommender System Training and Inference. In *Proceedings of the 16th ACM Conference on Recommender Systems (RecSys).*

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

[95] Jingxuan Wen, Huafeng Liu, Liping Jing, and Jian Yu. 2024. Learning-based counterfactual explanations for recommendation. *Science China Information Sciences* 67, 8 (2024), 182102.

[96] Xin Xia, Hongzhi Yin, Junliang Yu, Qinyong Wang, Guandong Xu, and Quoc Viet Hung Nguyen. 2022. On-device next-item recommendation with self-supervised knowledge distillation. In *Proceedings of the 45th International ACM SIGIR Conference on Research and Development in Information Retrieval.* 546–555.

[97] Minhui Xie, Kai Ren, Youyou Lu, Guangxu Yang, Qingxing Xu, Bihai Wu, Jiazhen Lin, Hongbo Ao, Wanhong Xu, and Jiwu Shu. 2020. 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).*

[98] Xing Xie, Jianxun Lian, Zheng Liu, Xiting Wang, Fangzhao Wu, Hongwei Wang, and Zhongxia Chen. 2018. Personalized recommendation systems: Five hot research topics you must know. *Microsoft Research Lab-Asia* (2018). https://www.microsoft.com/en-us/research/lab/microsoft-research-asia/articles/personalized-recommendation-systems/

[99] Ronghui Xu, Meng Chen, Yongshun Gong, Yang Liu, Xiaohui Yu, and Liqiang Nie. 2023. TME: Tree-guided multi-task embedding learning towards semantic venue annotation. *ACM Transactions on Information Systems* 41, 4 (2023), 1–24.

[100] Zhiqiang Xu, Dong Li, Weijie Zhao, Xing Shen, Tianbo Huang, Xiaoyun Li, and Ping Li. 2021. Agile and Accurate CTR Prediction Model Training for Massive-Scale Online Advertising Systems. In *Proceedings of the International Conference on Management of Data (SIGMOD).*

[101] Bencheng Yan, Pengjie Wang, Jinquan Liu, Wei Lin, Kuang-Chih Lee, Jian Xu, and Bo Zheng. 2021. Binary Code based Hash Embedding for Web-scale Applications. In *Proceedings of the 30th ACM International Conference on Information & Knowledge Management (CIKM).*

[102] Tong Yang, Junzhi Gong, Haowei Zhang, Lei Zou, Lei Shi, and Xiaoming Li. 2018. HeavyGuardian: Separate and Guard Hot Items in Data Streams. In *Proceedings of the 24th ACM SIGKDD Conference on Knowledge Discovery & Data Mining (KDD).*

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

[104] Chunxing Yin, Bilge Acun, Carole-Jean Wu, and Xing Liu. 2021. TT-Rec: Tensor Train Compression for Deep Learning Recommendation Models. In *Proceedings of Machine Learning and Systems (MLSys).*

[105] Chunxing Yin, Da Zheng, Israt Nisa, Christos Faloutsos, George Karypis, and Richard Vuduc. 2022. Nimble gnn embedding with tensor-train decomposition. In *Proceedings of the 28th ACM SIGKDD Conference on Knowledge Discovery and Data Mining.* 2327–2335.

[106] Rex Ying, Ruining He, Kaifeng Chen, Pong Eksombatchai, William L Hamilton, and Jure Leskovec. 2018. Graph convolutional neural networks for web-scale recommender systems. In *Proceedings of the 24th ACM SIGKDD international conference on knowledge discovery & data mining.* 974–983.

[107] Zhiyang Yuan, Wenguang Zheng, Peilin Yang, Qingbo Hao, and Yingyuan Xiao. 2023. Evolving Interest with Feature Co-action Network for CTR Prediction. *Data Science and Engineering* 8, 4 (2023), 344–356.

[108] Liu Yufang, Wang Shaoqing, Li Keke, Li Xueting, and Sun Fuzhen. 2024. Channel-Enhanced Contrastive Cross-Domain Sequential Recommendation. *Data Science and Engineering* (2024), 1–16.

[109] Weixin Zeng, Xiang Zhao, Jiuyang Tang, Xuemin Lin, and Paul Groth. 2021. Reinforcement learning–based collective entity alignment with adaptive features. *ACM Transactions on Information Systems (TOIS)* 39, 3 (2021), 1–31.

[110] 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. 2020. Model Size Reduction Using Frequency Based Double Hashing for Recommender Systems. In *Proceedings of the 14th ACM Conference on Recommender Systems (RecSys).*

[111] Fuzheng Zhang, Nicholas Jing Yuan, Defu Lian, Xing Xie, and Wei-Ying Ma. 2016. Collaborative knowledge base embedding for recommender systems. In *Proceedings of the 22nd ACM SIGKDD international conference on knowledge discovery and data mining.* 353–362.

[112] Hailin Zhang, Zirui Liu, and Boxuan Chen. 2023. Source codes related to CAFE+. https://github.com/HugoZHL/CAFE.

[113] Hailin Zhang, Zirui Liu, and Boxuan Chen. 2023. Supplementary file of CAFE+. https://github.com/HugoZHL/CAFE/blob/master/supplementary_file.pdf.

[114] Hailin Zhang, Zirui Liu, Boxuan Chen, Yikai Zhao, Tong Zhao, Tong Yang, and Bin Cui. 2024. CAFE: Towards Compact, Adaptive, and Fast Embedding for Large-scale Recommendation Models. *In Proceedings of the 2024 ACM International Conference on Management of Data (SIGMOD)* (2024).

[115] Hailin Zhang, Penghao Zhao, Xupeng Miao, Yingxia Shao, Zirui Liu, Tong Yang, and Bin Cui. 2023. Experimental Analysis of Large-scale Learnable Vector Storage Compression. *Proceedings of the VLDB Endowment* 17, 4 (2023), 808–822.

[116] Jia-Dong Zhang and Chi-Yin Chow. 2015. 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)*.

[117] Xiao Zhang, Meng Liu, Jianhua Yin, Zhaochun Ren, and Liqiang Nie. 2021. Question tagging via graph-guided ranking. *ACM Transactions on Information Systems (TOIS)* 40, 1 (2021), 1–23.

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

[119] Yuan Zhang, Dong Wang, and Yan Zhang. 2019. Neural IR meets graph embedding: A ranking model for product search. In *The World Wide Web Conference*. 2390–2400.

[120] Zheng Zhang, Zhihui Lai, Zi Huang, Wai Keung Wong, Guo-Sen Xie, Li Liu, and Ling Shao. 2019. Scalable supervised asymmetric hashing with semantic and latent factor embedding. *IEEE Transactions on Image Processing* 28, 10 (2019), 4803–4818.

[121] Zhigao Zhang, Fanfei Song, Bin Wang, and Chuansheng Dong. 2024. Extract implicit semantic friends and their influences from bipartite network for social recommendation. *Data Science and Engineering* 9, 3 (2024), 278–293.

[122] Weijie Zhao, Deping Xie, Ronglai Jia, Yulei Qian, Ruiquan Ding, Mingming Sun, and Ping Li. 2020. Distributed Hierarchical GPU Parameter Server for Massive Scale Deep Learning Ads Systems. In *Proceedings of Machine Learning and Systems (MLSys)*.

[123] Weijie Zhao, Jingyuan Zhang, Deping Xie, Yulei Qian, Ronglai Jia, and Ping Li. 2019. AIBox: CTR Prediction Model Training on a Single Node. In *Proceedings of the 28th ACM International Conference on Information & Knowledge Management (CIKM)*.

[124] Wayne Xin Zhao, Yupeng Hou, Junhua Chen, Jonathan JH Zhu, Eddy Jing Yin, Hanting Su, and Ji-Rong Wen. 2020. Learning semantic representations from directed social links to tag microblog users at scale. *ACM Transactions on Information Systems (TOIS)* 38, 2 (2020), 1–30.

[125] Xiangyu Zhao, Haochen Liu, Hui Liu, Jiliang Tang, Weiwei Guo, Jun Shi, Sida Wang, Huiji Gao, and Bo Long. 2021. AutoDim: Field-aware Embedding Dimension Searchin Recommender Systems. In *Proceedings of the Web Conference (WWW)*.

[126] Xiangyu Zhao, Haochen Liu, Hui Liu, Jiliang Tang, Weiwei Guo, Jun Shi, Sida Wang, Huiji Gao, and Bo Long. 2021. Autodim: Field-aware embedding dimension searchin recommender systems. In *Proceedings of the Web Conference 2021*. 3015–3022.

[127] Xiangyu Zhaok, Haochen Liu, Wenqi Fan, Hui Liu, Jiliang Tang, Chong Wang, Ming Chen, Xudong Zheng, Xiaobing Liu, and Xiwang Yang. 2021. Autoemb: Automated embedding dimensionality search in streaming recommendations. In *2021 IEEE International Conference on Data Mining (ICDM)*. IEEE, 896–905.

[128] Guorui Zhou, Xiaoqiang Zhu, Chengru Song, Ying Fan, Han Zhu, Xiao Ma, Yanghui Yan, Junqi Jin, Han Li, and Kun Gai. 2018. Deep Interest Network for Click-Through Rate Prediction. In *Proceedings of the 24th ACM SIGKDD Conference on Knowledge Discovery & Data Mining (KDD)*.

[129] Sheng Zhou, Xin Wang, Martin Ester, Bolang Li, Chen Ye, Zhen Zhang, Can Wang, and Jiajun Bu. 2021. Direction-aware user recommendation based on asymmetric network embedding. *ACM Transactions on Information Systems (TOIS)* 40, 2 (2021), 1–23.

[130] Yu Zhou, Chen Chen, Yongchao Wang, Tingting Han, and Taolue Chen. 2023. Context-aware API recommendation using tensor factorization. *Science China Information Sciences* 66, 2 (2023), 122101.

[131] Jieming Zhu, Jinyang Liu, Shuai Yang, Qi Zhang, and Xiuqiang He. 2021. Open Benchmarking for Click-Through Rate Prediction. In *Proceedings of the 30th ACM International Conference on Information & Knowledge Management (CIKM)*.