Embedding Compression in Recommender Systems: A Survey

SHIWEI LI, Huazhong University of Science and Technology, China
HUIFENG GUO, Huawei Noah's Ark Lab, China
XING TANG, Tencent, China
RUIMING TANG and LU HOU, Huawei Noah's Ark Lab, China
RUIXUAN LI, Huazhong University of Science and Technology, China
RUI ZHANG, ruizhang.info, China

To alleviate the problem of information explosion, recommender systems are widely deployed to provide personalized information filtering services. Usually, embedding tables are employed in recommender systems to transform high-dimensional sparse one-hot vectors into dense real-valued embeddings. However, the embedding tables are huge and account for most of the parameters in industrial-scale recommender systems. In order to reduce memory costs and improve efficiency, various approaches are proposed to compress the embedding tables. In this survey, we provide a comprehensive review of embedding compression approaches in recommender systems. We first introduce deep learning recommendation models and the basic concept of embedding compression in recommender systems. Subsequently, we systematically organize existing approaches into three categories: low precision, mixed dimension, and weight sharing. Lastly, we summarize the survey with some general suggestions and provide future prospects for this field.

CCS Concepts: • **Information systems** → **Recommender systems;**

Additional Key Words and Phrases: recommender systems; embedding tables; model compression; survey

ACM Reference format:
Shiwei Li, Huifeng Guo, Xing Tang, Ruiming Tang, Lu Hou, Ruixuan Li, and Rui Zhang. 2024. Embedding Compression in Recommender Systems: A Survey. *ACM Comput. Surv.* 56, 5, Article 130 (January 2024), 21 pages.
https://doi.org/10.1145/3637841

This work is supported in part by the National Natural Science Foundation of China (grant nos. 62376103, 62302184, and 62206102) and the Science and Technology Support Program of Hubei Province (grant no. 2022BAA046).
S. Li and H. Guo contributed equally to this research.
This work was done when X. Tang worked at Huawei Noah's Ark Lab.
Authors' addresses: S. Li and R. Li (Corresponding author), Huazhong University of Science and Technology, Wuhan, China, 430074; e-mails: {lishiwei, rxli@hust}.edu.cn; H. Guo, R. Tang, and L. Hou, Huawei Noah's Ark Lab, Shenzhen, China, 518129; e-mails: {huifeng.guo, tangruiming, houlu3}@huawei.com; X. Tang, Tencent, Shenzhen, China, 518054; e-mail: xing.tang@hotmail.com; R. Zhang (Corresponding author), ruizhang.info, China; e-mail: rayteam@yeah.net.

0360-0300/2024/01-ART130 $15.00
https://doi.org/10.1145/3637841

1 INTRODUCTION

To alleviate the problem of information explosion, recommender systems [55, 61] are extensively deployed to provide personalized information filtering services, including online shopping [80], advertising systems [46], and so on. Meanwhile, deep learning techniques have shown impressive capabilities in capturing user preferences for candidate items. Thereupon, both the industry and research communities have proposed a variety of **deep learning recommendation models (DLRMs)** to enhance the performance of recommender systems, such as Wide & Deep [6] in Google Play, DIN [80] in Alibaba, and DeepFM [18] in Huawei.

1.1 Deep Learning Recommendation Models

Recommender systems are utilized for a diverse range of tasks, such as candidate item matching [30], **click-through rate (CTR)** prediction [51, 52], and **conversion rate (CVR)** prediction [44]. For each of these tasks, the employed DLRMs have undergone meticulous design processes to ensure optimal performance. However, without loss of generality, most DLRMs follow the *embedding table* and *neural network* paradigm [18, 45, 57, 58] despite the fact that the specific design of the *neural network* component may vary across different model architectures.

As illustrated in Figure 1, the embedding table is responsible for converting input rows into dense embedding vectors. It is worth noting that the input rows of DLRMs typically consist of categorical features, which are encoded as high-dimensional one-hot vectors. Each category feature will be referred to simply as *feature* and all features under the same category form a feature field. Generally, each feature is associated with a unique embedding stored in the embedding table $\mathbf{E} \in \mathbb{R}^{n \times d}$, where n denotes the total number of features and d denotes the embedding dimension.

On the other hand, the neural network is primarily engaged in interacting, processing, and analyzing feature embeddings, along with making predictions. Recent studies [18, 29, 36, 40, 65, 70] have consistently focused on optimizing the feature extraction capabilities of the neural networks. For example, [18, 58] utilize product operators to model the feature interactions between different feature fields. The authors of [36, 40] employ convolutions on embeddings to capture feature interactions of arbitrary order. The authors of [65] introduce an additional attention network to assign varying importance to different feature interactions. Additionally, [29, 70] automatically search for suitable interaction functions using AutoML techniques [21]. In this article, we do not delve into the detailed design of the neural network component. Instead, we recommend referring to the works [61, 74, 75] for a comprehensive understanding of the neural networks used in DLRMs.

Despite incorporating various intricate designs, the neural network usually entails relatively shallow layers and a limited number of model parameters. In contrast, the embedding table occupies the vast majority of model parameters. Especially in industrial-scale recommender systems, in which there are billions or even trillions of categorical features, the embedding table may take hundreds of GB or even TB to hold [17]. For example, the size of embedding tables in Baidu's advertising systems reaches 10 TB [66]. As the scale of recommender systems perpetually expands, the continuous growth in the number of features will bring greater storage overhead.

1.2 Embedding Compression in Recommender Systems

In addition to increasing storage overhead, larger embedding tables will also result in higher latency during table lookup,[1] which will reduce the efficiency of model training and inference. Therefore, to deploy the DLRMs with large embedding tables in real production environments efficiently and economically, it is necessary to compress their embedding tables.

[1]The process of retrieving an embedding from the embedding table based on the input feature or index.

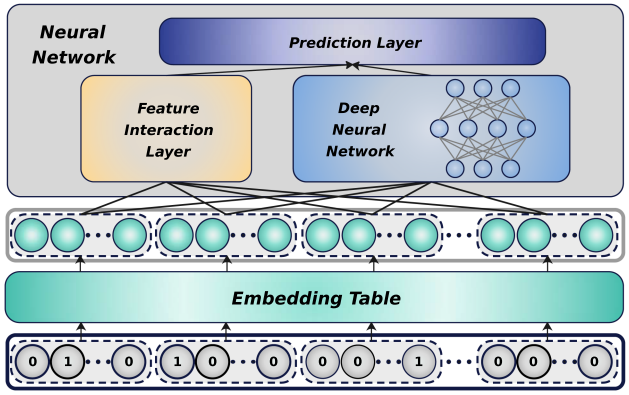

Fig. 1. The *embedding table* and *neural network* paradigm of deep learning recommendation models (DLRMs). Note that the *neural network* component may vary in different model architectures. Here, we only present the *neural network* with the classic dual tower architecture as an example.

However, embedding compression in DLRMs differs significantly from model compression in other fields, such as **Computer Vision (CV)** [8] and **Natural Language Processing (NLP)** [19]. These differences primarily manifest in three aspects: model architectures, properties of input data, and model size. First, vision models and language models are usually very deep neural networks stacked by fully connected layers, convolutional layers, or transformers. Consequently, compression methods designed for these models focus on compressing the aforementioned modules rather than embedding tables. In contrast, DLRMs are typically shallow models, with the majority of parameters concentrated in the embedding tables. Second, the input data of vision models and language models are usually images and texts, inherently containing abundant visual and semantic information that can be leveraged for model compression. For example, the authors of [4] use the semantic information as *a priori* knowledge to compress the word embeddings, while the authors of [20] exploit the similarity between feature maps derived from image inputs to compress convolution kernels. However, in recommender systems, there is generally limited visual or semantic information available. Fortunately, DLRMs possess unique properties in the input data that can facilitate embedding compression. Specifically, categorical features are organized in feature fields and often follow a highly skewed long-tail distribution, with varying numbers of features in different fields. We can compress embedding tables based on feature frequency and field size. Third, embedding tables of DLRMs are usually hundreds or even thousands of times larger than vision models or language models [53], which presents a more challenging and necessary task for compression.

Recently, embedding compression has gained increasing attention in recommender systems, leading to the development and application of various embedding compression techniques for DLRMs. However, there is currently no comprehensive survey summarizing the methods employed for embedding compression. Therefore, the primary objective of this article is to review and summarize representative research in this field. The embedding table can be regarded as a matrix with three dimensions: the precision of weights, the dimension of embeddings, and the number of embeddings. To this end, we summary the embedding compression methods into three categories according to the dimensions they compress, as illustrated in Figure 2. First, **low-precision** methods reduce the memory of each weight by decreasing its bit width. According to the size of bit width and its corresponding advantages, we further divide the low-precision methods into binarization and quantization. Second, **mixed-dimension** methods reduce the memory of specific embeddings by decreasing their dimensions and using mixed-dimension embeddings. According

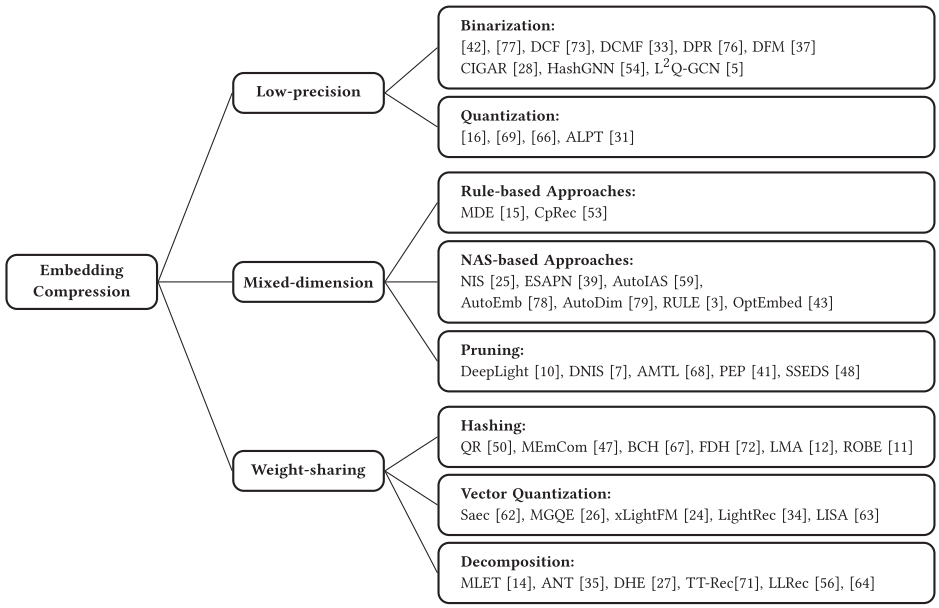

Fig. 2. Summary of representative studies on embedding compression in recommender systems.

to the techniques of determining the embedding dimension for different features, we categorize the mixed-dimension methods into rule-based approaches, NAS-based approaches, and pruning. Third, **weight-sharing** methods reduce the actual parameters of the embedding table by sharing weights among different embeddings. Considering that the number of features is given by the dataset, a solution to reduce the number of embeddings is to reuse embeddings among features. Furthermore, we generalize the sharing to the weight level and define the weight-sharing methods as generating embeddings with shared weights. According to the way that embeddings are generated, we categorize the mixed-dimension methods into hashing, vector quantization, and decomposition. We will introduce the three primary categories in Sections 2, 3, and 4, respectively.

Note that embeddings are fed into the neural network as representations of categorical features and form the foundations of DLRMs. Therefore, when compressing embeddings, it may affect many aspects of model performance, including model accuracy, inference efficiency, training efficiency, and training memory usage. We will discuss the pros and cons of different methods regarding these metrics at the end of each section. In Section 5, the survey is summarized, providing general suggestions for different scenarios and discussing future prospects for this field.

2 LOW PRECISION

As we all know, embedding weights are typically stored in the format of FP32,[2] which occupies 32 bits. To reduce the storage of each weight, low-precision approaches are developed to represent a weight with fewer bits. In particular, according to the bit width of weights, low-precision approaches can be further divided into **binarization** and **quantization**.

2.1 Binarization

Binarization is to compress a full-precision weight into a binary code that only occupies 1 bit. It is widely used in the embedding-based similarity search of recommender systems [28, 54] since the

[2]Short for *single-precision floating-point* format.

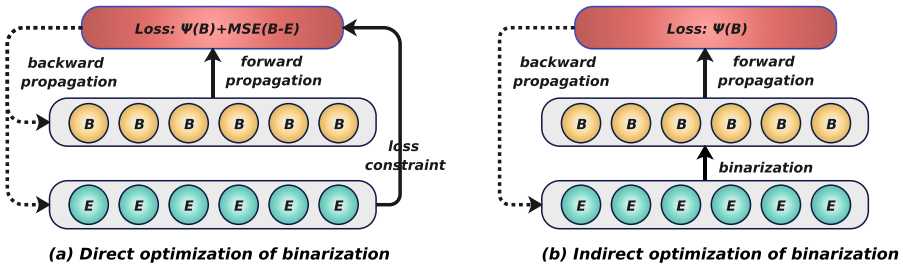

Fig. 3. End-to-end optimization paradigms of learning binary embeddings. B (yellow) and E (green) represent the binary and full-precision weights, respectively. *Loss* (red) is the training loss, where $\Psi(\cdot)$ is the objective function and MSE$(\cdot)$ is the mean square error function.

binary embeddings have two distinct advantages compared with the full-precision ones: (1) less memory or disk cost for storing embeddings; and (2) higher inference efficiency as the similarity (i.e., inner product) between binary embeddings can be calculated more efficiently through the Hamming distance, which has been proved in [73].

The work of [42, 77] pioneered obtaining binary embeddings in a two-stage (i.e, post-training) manner. Specifically, they first learn a full-precision embedding table while ignoring the binary constraints and then perform binarization (e.g., $sign(x)$) on the full-precision embeddings to get binary embeddings. However, the binarization procedure is not in the training process and, thus, cannot be optimized by minimizing the training objective, which will bring large irreparable errors and fail to meet an acceptable accuracy. To reduce accuracy degradation, subsequent works have focused on end-to-end approaches to learn the binary embeddings during training.

As shown in Figure 3, recent works typically learn binary embeddings following two optimization paradigms: direct optimization and indirect optimization. As Figure 3(a) shows, in the direct optimization of binarization, the binary embeddings are maintained as part of the model parameters and will be optimized directly by the training loss. For example, to improve the efficiency of **Collaborative Filtering (CF)**, DCF [73] learns a binary embedding table $\mathbf{B} \in \{\pm 1\}^{n \times d}$. To maximize the information encoded in each binary embedding, DCF further adds a balance-uncorrelation constraint to $\mathbf{B}$ (i.e., $\mathbf{B}^T\mathbf{1} = \mathbf{0}, \mathbf{B}^T\mathbf{B} = n\mathbf{I}$), where $\mathbf{I}$ is an identity matrix. However, it is NP-hard to optimize the binary embeddings with such constraint. To resolve this problem, DCF also maintains a full-precision embedding table $\mathbf{E} \in \mathbb{R}^{n \times d}$ with the same balance-uncorrelation constraint. The constraint of $\mathbf{B}$ is then replaced by adding the **mean square error (MSE)** of $(\mathbf{B}-\mathbf{E})$ to the objective function. During training, DCF will update $\mathbf{B}$ and $\mathbf{E}$ alternatively through different optimization algorithms. Specifically, $\mathbf{B}$ is updated by **Discrete Coordinate Descent (DCD)** and $\mathbf{E}$ is updated with the aid of **Singular Value Decomposition (SVD)**. This optimization paradigm has been widely used to learn binary embeddings in recommender systems such as DPR [76], DCMF [33], and DFM [37]. DPR changes the objective function of DCF (i.e., rating prediction) to personalized items ranking. DCMF and DFM extend this binarization paradigm to Content-aware Collaborative Filtering [32] and **Factorization Machine (FM)** [49], respectively.

As shown in Figure 3(b), another paradigm is indirect optimization, in which the binary embeddings $\mathbf{B}$ are generated from full-precision embeddings $\mathbf{E}$ on the fly and will be optimized indirectly by optimizing $\mathbf{E}$. However, it is infeasible to optimize $\mathbf{E}$ by the standard gradient descent as the gradients of the binary operations (e.g., $sign(x)$) are constantly zero. To solve this problem, CIGAR [28] replaces $sign(x)$ with the scaled tanh function $tanh(\alpha x)$ as $\lim_{\alpha \to \infty} tanh(\alpha x) = sign(x)$ and $tanh(\alpha x)$ has a better differential property. In the early stages of training, a smaller value of α is utilized to yield superior representations and, as the training progresses, its value

gradually increases to approximate $sign()$. Another way to solve the non-propagable gradient is with a **straight-through-estimator (STE)** [9], which treats some operations as identity maps during backpropagation. HashGNN [54] employs the STE variant of $sign()$, thus updating $\mathbf{E}$ with the gradients of $\mathbf{B}$. However, the huge gap between $\mathbf{B}$ and $\mathbf{E}$ will cause an imprecise update for $\mathbf{E}$. To solve this issue, HashGNN further develops a dropout-based binarization. Specifically, $\hat{\mathbf{E}} = (1 - \mathbf{P}) \odot \mathbf{E} + \mathbf{P} \odot \mathbf{B}$ will be fed into the following networks, where $\mathbf{P} \in \{0, 1\}^{n \times d}$ and $\odot$ is the element-wise product. Each element in $\mathbf{P}$ is a Bernoulli random value with probability p. During backpropagation, only the embeddings that are binarized will be updated through STE; the rest will be updated by standard gradient descent. To ensure convergence, HashGNN adopts a small value for p in the initial training phase, gradually increasing it as the training progresses. Similarly, L^{2}Q-GCN [5] uses the STE to optimize the full-precision embeddings while introducing a positive scaling factor $s = mean(|\boldsymbol{e}|)$ for each binary embedding to enhance its presentation capability, where $\boldsymbol{e}$ is the full-precision embedding. The comparison of the above three methods is summarized in Algorithm 1.

ALGORITHM 1: Comparison between CIGAR [28], HashGNN [54], and L^{2}Q-GCN [5].

Input: a full-precision embedding $\boldsymbol{e}$.
Output: the output embedding $\hat{\boldsymbol{e}}$. // $\hat{\boldsymbol{e}}$ will be fed into following networks.
Func CIGAR($\boldsymbol{e}$):
  $\hat{\boldsymbol{e}} = tanh(\alpha \cdot \boldsymbol{e})$ // α will increase as training progresses.
Func HashGNN($\boldsymbol{e}$):
  $\boldsymbol{b} = sign\_ste(\boldsymbol{e})$ // $sign\_ste()$ is the STE variant of $sign()$.
  $\boldsymbol{p} := \{0, 1\}^d$ // sample from Bernoulli distribution with probability p.
  $\hat{\boldsymbol{e}} = (1 - \boldsymbol{p}) \odot \boldsymbol{e} + \boldsymbol{p} \odot \boldsymbol{b}$ // p will increase as training progresses.
Func L^{2}Q-GCN($\boldsymbol{e}$):
  $\boldsymbol{b} = sign\_ste(\boldsymbol{e})$
  $\hat{\boldsymbol{e}} = mean(|\boldsymbol{e}|) \cdot \boldsymbol{b}$

2.2 Quantization

Although binarization has better efficiency and less memory cost at the inference stage, it may lead to a significant drop of accuracy, which is not acceptable in several scenarios. As Cheng et al. [6] claim, even a 0.1% decrease of the prediction accuracy may result in a large decline in revenue. To trade off the memory cost and the prediction accuracy, quantization is used to represent each weight with a multi-bit integer.

Quantization is the mapping of a 32-bit full-precision weight to an element in the set of quantized values $\mathbb{S} = \{q_0, q_1, \ldots, q_k\}$, where $k = 2^s - 1$ and s is the bit width. The most commonly used quantization function is uniform quantization, where the quantized values are uniformly distributed. Specifically, the step size $\Delta = q_i - q_{i-1}$ remains the same for any $i \in [1, k]$. Let w be a value clipped into the range $[q_0, q_k]$; we can quantize it into an integer as $\hat{w} = rd((w - q_0)/\Delta)$, where $rd(x)$ rounds x to an adjacent integer. The integer $\hat{w}$ will be de-quantized into a floating-point value $(\hat{w} \times \Delta + q_0)$ when used. Existing work on embedding quantization either performs post-training quantization or trains a quantized embedding table from scratch.

Guan et al. [16] study **post-training quantization (PTQ)** on the embedding tables and propose a uniform and a non-uniform quantization algorithm. Specifically, in the uniform quantization, they maintain a quantization range for each embedding and find the best quantization range by a greedy search algorithm. In the non-uniform quantization, they divide similar embeddings into groups and apply k-means clustering on the weights to produce a codebook for each group. The

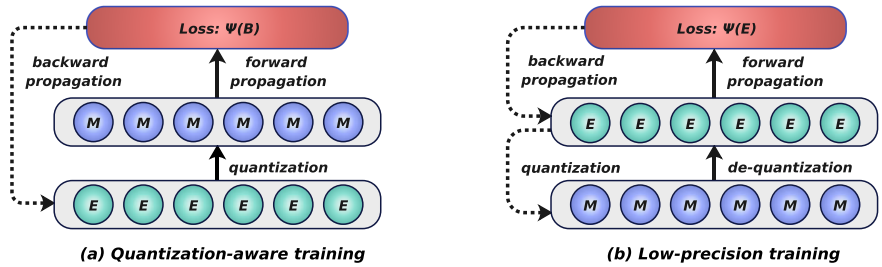

Fig. 4. Training frameworks of quantization. *Loss* (red) is the training loss. *M* (purple) and *E* (green) represent the integer and full-precision weights, respectively.

weights in each group will be mapped to the index of a value in the corresponding codebook. These two algorithms improve the accuracy of PTQ; however, they still suffer from accuracy degradation.

To further reduce accuracy degradation, recent works [66, 69] learn quantized weights from scratch. Unlike the well-known **quantization-aware training (QAT)** [2, 13], [66, 69] use another quantization training framework to exploit the sparsity of the input data, which we term low-precision training (LPT). As Figure 4(a) shows, QAT quantizes the full-precision weights in the forward pass and updates the full-precision weights with the gradients estimated by the STE. As Figure 4(b) shows, in LPT, the weights are stored in the format of integers at training, thereby compressing the training memory. The model takes the de-quantized weights as input and will quantize the weights back into integers after backpropagation. Since the input one-hot vectors of DLRMs are highly sparse, only an extremely small part of the embeddings will be de-quantized into floating-point values, whose memory is negligible. Xu et al. [66] use 16-bit LPT on the embedding table without sacrificing accuracy. To enhance the compression capability of LPT, Yang et al. [69] propose a mixed-precision scheme in which most embeddings are stored in the format of integers and only the most recently or frequently used embeddings are stored in a full-precision cache. With a small cache, they achieve lossless compression with 8-bit or even 4-bit quantization. Li et al. [31] propose an adaptive low-precision training scheme to learn the quantization step size for better model accuracy.

2.3 Discussion

Low precision is a simple yet effective way for embedding compression. At the inference stage, binarization can reduce the memory usage by 32× and accelerate the inference through Hamming distance. However, the binary embeddings usually cause severe accuracy degradation and need to be trained with the guidance of full-precision embeddings, which requires more memory usage and computing resources at training. In contrast, quantization has a limited compression capability but can achieve a comparable accuracy to the full-precision embeddings. Recent quantization approaches for embedding tables can also compress the memory usage at the training stage and improve the training efficiency by reducing the communication traffic.

3 MIXED DIMENSION

Embedding tables usually assign a uniform dimension to all the embeddings in a heuristic way, which turns out to be suboptimal in both prediction accuracy and memory usage [25]. As confirmed in [78], a low-dimensional embedding is good at handling less frequent features where a high-dimensional embedding cannot be well trained. Therefore, to boost the model performance, it is important to assign an appropriate dimension to each feature and use mixed-dimension embeddings.

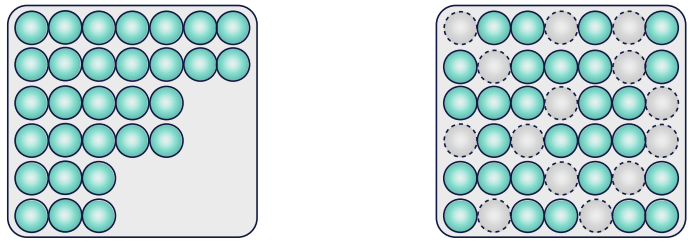

Fig. 5. Structures of mixed-dimension embeddings.

Existing methods can obtain mixed-dimension embeddings in a structured or an unstructured manner. As shown in Figure 5, structured approaches divide the embeddings into groups, each of which has a unique dimension, whereas unstructured approaches learn a sparse embedding table where the embeddings have various dimensions. However, these mixed-dimension embeddings are not compatible with the operations (e.g., inner product), which require embeddings of the same length. Therefore, the mixed-dimension embeddings need to be transformed into a uniform dimension before feeding into the following networks. Such transformation is usually achieved by linear projection or simply zero padding. Apart from the difference in the embedding structures, existing methods also differ greatly in generating mixed-dimension embeddings. In this section, we will introduce three kinds of mixed-dimension approaches: **rule-based** approaches, **NAS-based** approaches, and **pruning**.

3.1 Rule-Based Approaches

It is a common understanding that the features with higher frequencies are more informative and the fields with more features occupy more memory. Thus, the embedding dimension can be set with a heuristic rule based on the feature frequency and field size. To deploy the item embeddings into resource-constraint devices, CpRec [53] divides the items into several groups by frequency and assigns a predefined dimension to each group according to the frequencies of owned features. Similarly, MDE [15] assigns each feature field with a unique dimension according to the number of features included in this field. Specifically, let $\boldsymbol{n} \in \mathbb{R}^m$ denote the number of features in all m feature fields and $\boldsymbol{p} = 1/\boldsymbol{n}$; then, the embedding dimension of the i-th field would be $\bar{d}\boldsymbol{p_i}^{\alpha}/||\boldsymbol{p}||_{\infty}^{\alpha}$, where $\bar{d}$ is the base dimension and $\alpha \in [0, 1]$ denotes the temperature. These rule-based approaches are simple yet effective in reducing the memory usage and alleviating the overfitting problems. However, they suffer from suboptimal performance as the heuristic rules cannot be optimized by the ultimate goal of minimizing the training objective.

3.2 NAS-Based Approaches

NAS was originally proposed to search for the optimal neural network architectures [82]. Recently, it has also been adopted in searching embedding dimensions for different features. Unlike the rule-based approaches in which the dimension is set based on *a priori*, it is now learned. Generally, there are three components in the NAS-based approaches: (1) search space — relaxing the large optimization space of embedding dimensions with heuristic assumptions; (2) controller —usually a neural network or learnable parameters, selecting candidate dimension from the search space in a hard or soft manner; and (3) updating algorithm —updating the controller with **reinforcement learning (RL)** algorithms or **differential architecture search (DARTS)** [38] techniques and so on.

We first introduce the approaches of NIS [25], ESAPN [39], and AutoIAS [59], which adopt a policy network as the controller and update the controller with RL algorithms. NIS [25] and ESAPN [39] are designed to learn embedding dimensions of users and items. In NIS, the authors relax the original search space $n \times d$ to $G \times B$ ($G < n$ and $B < d$) by dividing the features into G groups and cutting the embedding dimension of each group into B blocks. Then, they use the controller to select several blocks and generate the final embeddings. In ESAPN, the authors predefine the search space as a set of candidate embedding dimensions $\mathbb{D} = \{d_1, d_2, \ldots, d_k\}$, where $d_1 < d_2 < \ldots < d_k$. Inspired by the fact that the frequencies of features are monotonically increasing in the data stream, they decide to increase or keep the current embedding dimension instead of selecting one in $\mathbb{D}$. The decision is made by the controller based on the feature frequency and the current embedding dimension. In contrast to NIS and ESAPN, AutoIAS searches not only the embedding dimensions of feature fields but also the following neural network architectures. The authors design a search space for each model component (e.g., the search space of embedding dimensions is similar as $\mathbb{D}$ in ESAPN). To boost the training efficiency, they maintain a supernet at training and use the controller to generate sub-architectures by inheriting parameters from the supernet.

The RL-based approaches described above perform a hard selection by selecting only one embedding dimension for each feature or field at a time. Instead, inspired by DARTS, AutoEmb [78] and AutoDim [79] make a soft selection by weighted summing over the embeddings of the candidate dimensions in $\mathbb{D} = \{d_1, d_2, \ldots, d_k\}$. Let $\boldsymbol{\omega} \in [0, 1]^k$ denote the vector composed of the weighting coefficients. AutoEmb searches for the embedding dimensions of individual features, whereas AutoDim searches for the embedding dimensions of the feature fields. In AutoEmb, the controller is a neural network and generates $\boldsymbol{\omega}$ based on the feature frequency, whereas AutoDim directly assigns each field with a learnable vector $\boldsymbol{\omega}$, and it further approximates the hard selection by performing Gumbel-Softmax [22] on $\boldsymbol{\omega}$. At training, the controller in AutoEmb and the learnable vectors in AutoDim are optimized through DARTS techniques. After training, the corresponding dimension of the largest weight in $\boldsymbol{\omega}$ is selected and the model will be retrained for better accuracy.

Considering that the training process of the controller is quite time-consuming, recent works [3, 43] search for the optimal embedding dimensions after training the models without using any controller. They first sample some structures from the search space and then explore the entire search space by using evolutionary search strategies on the sampled structures. Specifically, RULE [3] cuts the embedding table into $G \times B$ blocks similar to NIS and adds a diversity regularization to the blocks in the same group for maximizing expressiveness. After training, RULE selects the most suitable embedding blocks under a memory budget (i.e., the maximum number of blocks). OptEmbed [43] trains a supernet while removing non-informative embeddings. After training, it then assigns each field with a binary mask $\boldsymbol{m} \in \{0, 1\}^d$ to obtain mixed-dimension embeddings, where d is the original dimension. The block selections in RULE and the masks in OptEmbed are determined and evolved by the search strategies.

3.3 Pruning

Instead of shortening the length of embeddings, pruning can obtain a sparse embedding table and thus get mixed-dimension embeddings. For instance, DeepLight [10] prunes the embedding table in a certain proportion. During training, it prunes and retrains the embedding table alternatively so that the mistakenly pruned weights can grow back. In addition, DeepLight will increase the pruning proportion gradually as training proceeds.

Another way to prune the embeddings is to train the embeddings with learnable masks. Specifically, an embedding $\boldsymbol{e}$ is pruned as $\hat{\boldsymbol{e}} = \boldsymbol{m} \odot \boldsymbol{e}$ for the forward pass, where $\boldsymbol{m}$ is the mask and $\odot$ is the element-wise product. DNIS [7] divides features into groups by frequency and assigns each group with a learnable mask $\boldsymbol{m} \in [0, 1]^d$. AMTL [68] develops a network to generate a binary

mask $\boldsymbol{m} \in \{0,1\}^d$ for each feature based on its frequency. Similarly, PEP [41] generates a binary mask $\boldsymbol{m} \in \{0,1\}^d$ for each feature as $\boldsymbol{m} = \mathbb{I}(|\boldsymbol{e}| > g(s))$, where $\mathbb{I}(\cdot)$ is the indicator function and $g(s)$ (e.g., $sigmoid(s)$) serves as a learnable threshold. In particular, in PEP, the embeddings should minus $g(s)$ before being pruned by the masks (i.e. $\hat{\boldsymbol{e}} = \boldsymbol{m} \odot (\boldsymbol{e} - g(s))$). At training, the network in AMTL and the learnable threshold in PEP are optimized together with the model parameters by gradient descent, whereas the learnable masks in DNIS are optimized by DARTS. After training, AMTL and PEP preserve $\hat{\boldsymbol{e}}$ as the final embeddings, whereas DNIS need pruning $\hat{\boldsymbol{e}}$ with a threshold as the redundant weights in $\hat{\boldsymbol{e}}$ are not exact zero. The differences between the above methods in generating masks are summarized in Algorithm 2.

ALGORITHM 2: Comparison between DNIS [7], AMTL [68], and PEP [41]

Input: the full-precision embedding $\boldsymbol{e}$ and feature frequency f.
Output: the pruned embedding $\hat{\boldsymbol{e}}$. `// ê will be fed into following networks.`
Func `DNIS(`$\boldsymbol{e}$`)`**:**
  $\boldsymbol{m} := [0,1]^d$ `// m is a learnable mask.`
  $\hat{\boldsymbol{e}} = \boldsymbol{m} \odot \boldsymbol{e}$ `// m is shared by features with similar frequency.`
Func `AMTL(`$\boldsymbol{e}$`)`**:**
  $\boldsymbol{m} := amtl(f)$ `// m is generated by a network` $amtl()$ `with the frequency` f.
  $\hat{\boldsymbol{e}} = \boldsymbol{m} \odot \boldsymbol{e}$
Func `PEP(`$\boldsymbol{e}$`)`**:**
  $\boldsymbol{m} = \mathbb{I}(|\boldsymbol{e}| > sigmoid(s))$ `//` $sigmoid(s)$ `serves as a learnable threshold.`
  $\hat{\boldsymbol{e}} = \boldsymbol{m} \odot (\boldsymbol{e} - sigmoid(s))$

To get rid of the extra training process of optimizing the masks, SSEDS [48] develops a single-shot pruning algorithm to prune on the pretrained models. For a pretrained model, SSEDS will prune the columns of the embedding matrix for each field and produce structured embeddings. After training, SSEDS assigns each feature field with a mask and forms a mask matrix $\mathbf{M} = \{1\}^{\hat{n} \times d}$, where $\hat{n}$ is the number of fields and d is the original embedding dimension. Instead of learning $\mathbf{M}$ in the training process, SSEDS uses $g_{ij} = \partial f(\mathbf{M}, \mathbf{E}) / \partial \mathbf{M}_{ij}$ to identify the importance of the j-th dimension in the i-th field, where g_{ij} is the gradient of the loss function with respect to $\mathbf{M}_{ij}$. Specifically, a larger magnitude of $|g_{ij}|$ means that the corresponding dimension has a greater impact on the loss function. Note that all $|g_{ij}|$ can be computed efficiently via only one forward-backward pass. Given a memory budget, SSEDS calculates a saliency score for each dimension as $s_{ij} = |g_{ij}| / \sum_{i=0}^{\hat{n}} \sum_{j=0}^{d} |g_{ij}|$ and prunes the dimensions with the lowest saliency scores.

3.4 Discussion

Mixed-dimension approaches can alleviate the overfitting problems and obtain better accuracy but usually have worse efficiency at the training and inference stages. At inference, the structured approaches usually suffer from extra computing cost due to the linear transformation and the unstructured approaches store the sparse embedding table using *sparse matrix storage*, which will cost extra effort to access. At training, NAS-based approaches require an extremely long time for searching and pruning usually needs to retrain the pruned models for better accuracy, which doubles the training time. In contrast, rule-based approaches have little influence on efficiency and can save memory also at the training stage. However, they cannot achieve the optimal accuracy.

4 WEIGHT-SHARING

Low-precision approaches reduce the number of bits in a weight and mixed-dimension approaches reduce the number of weights in an embedding. Unlike them, weight-sharing approaches share weights among the embedding table, thereby reducing the actual number of parameters within it.

Existing weight-sharing approaches usually generate embeddings with several shared vectors. In this section, we relax the definition of weight-sharing and formulate a weight-sharing paradigm based on existing approaches. Specifically, the embedding generation is formulated as $\boldsymbol{e} = \bigcup / \sum_{i=1}^{s} \boldsymbol{I}^i \times \mathbf{T}^i$, where $\bigcup / \sum$ denotes concatenation or summation, $\mathbf{T}^i$ is a matrix composed of shared vectors and $\boldsymbol{I}^i$ is an index vector for selecting shared vectors in $\mathbf{T}^i$. For ease of expression, we refer to the shared vectors as meta-embeddings and the matrices of shared vectors as meta-tables. According to the principle of constructing the index vectors, we introduce three kinds of weight-sharing methods: **hashing**, **vector quantization**, and **decomposition**.

4.1 Hashing

Hashing methods generate the index vectors by processing the original feature id with hash functions. For instance, the naïve hashing method [60] compresses the embedding table with a simple hash function (e.g., the reminder function). Specifically, given the original size of the embedding table $n \times d$, each feature has an embedding $\boldsymbol{e} = \boldsymbol{I} \times \mathbf{T}$, where $\mathbf{T} \in \mathbb{R}^{m \times d} (m < n)$ and $\boldsymbol{I} = \text{one-hot}(\texttt{id} \,\%m) \in \{0, 1\}^m$. Note that $\boldsymbol{I} \times \mathbf{T}$ is actually achieved by table look-up when $\boldsymbol{I}$ is a one-hot vector. However, [60] naïvely maps multiple features to the same embedding. The collisions between features will result in loss of information and drop of accuracy.

ALGORITHM 3: Comparison between QR [50], MEmCom [47], BCH [67], and FDH [72]

Input: the feature id ($\texttt{id} \leq n$).
Output: the generated embedding $\hat{\boldsymbol{e}}$. // $\hat{e}$ will be fed into following networks.
Func QR(id):
  $\boldsymbol{I}^1 = \text{one-hot}(\texttt{id} \,\%m)$, $\boldsymbol{I}^2 = \text{one-hot}(\texttt{id} \mid m)$ // m is a predefined parameter.
  $\hat{\boldsymbol{e}} = \boldsymbol{I}^1 \times \mathbf{T}^1 + \boldsymbol{I}^2 \times \mathbf{T}^2$ // $\mathbf{T}^1 \in \mathbb{R}^{m \times d}$ and $\mathbf{T}^2 \in \mathbb{R}^{(n|m) \times d}$.
Func MEmCom(id):
  $\boldsymbol{I}^1 = \text{one-hot}(\texttt{id} \,\%m)$, $\boldsymbol{I}^2 = \text{one-hot}(\texttt{id})$
  $\hat{\boldsymbol{e}} = (\boldsymbol{I}^1 \times \mathbf{T}^1) \cdot (\boldsymbol{I}^2 \times \mathbf{T}^2)$ // $\mathbf{T}^1 \in \mathbb{R}^{m \times d}$ and $\mathbf{T}^2 \in \mathbb{R}^{n \times 1}$.
Func BCH(id):
  divide the bits within id into s sub-ids $\{\texttt{id}_1, \ldots, \texttt{id}_s\}$.
  $\hat{\boldsymbol{e}} = \sum_{i=1}^{s} \text{one-hot}(\texttt{id}_i) \times \mathbf{T}$ // $\mathbf{T}$ is a shared meta-table.
Func FDH(id):
  if *feature* id *is frequent* **then**
    $\hat{\boldsymbol{e}} := \mathbb{R}^d$ // frequent features have unique embeddings.
  else
    $\hat{\boldsymbol{e}} = \text{QR}(\texttt{id})$

To reduce the collisions, existing hashing methods use multiple hash functions to process the feature id. They usually maintain multiple meta-tables $(\mathbf{T}^1, \ldots, \mathbf{T}^s)$ and generate multiple index vectors as $\boldsymbol{I}^i = \text{one-hot}(\text{hash}^i(\texttt{id}))$, where $i \in [1, s]$ and $\{\text{hash}^i\}_{i=1}^{s}$ is a group of hash functions. For example, QR [50] maintains two meta-tables and uses the quotient function and the reminder function to generate two index vectors. Similarly, MEmCom [47] also maintains two meta-tables ($\mathbf{T}^1 \in \mathbb{R}^{m \times d}, \mathbf{T}^2 \in \mathbb{R}^{n \times 1}$) and generates two index vectors as $\boldsymbol{I}^1 = \text{one-hot}(\texttt{id} \,\%m)$, $\boldsymbol{I}^2 = \text{one-hot}(\texttt{id})$. To better distinguish the features, MEmCom multiplies two meta-embeddings as the final embedding. Further, Yan et al. [67] use **Binary Code–Based Hash (BCH)** functions to process the feature id at bit level. It divides the 64 bits of a feature id into s groups and restructures them into s sub-ids $(\texttt{id}_1, \ldots, \texttt{id}_s)$. Each sub-id corresponds to an index vector (i.e., $\boldsymbol{I}^i = \text{one-hot}(\texttt{id}_i), i \in [1, s]$) and obtains a meta-embedding. Additionally, to enhance the compression capability, BCH keeps one single meta-table and shares it among all $\mathbf{T}^i, i \in [1, s]$. Although the hashing methods

described above are efficient in memory reduction, they still suffer from accuracy degradation and extra computing cost of the hash functions. To alleviate these problems, Zhang et al. [72] developed a **Frequency-Based Double Hashing (FDH)** method, which only uses hashing on the features with low frequencies. In this way, fewer features need to be processed by the hash function. With a little extra storage for the most frequent features, FDH not only improves the prediction accuracy but also the inference efficiency. The difference between the above methods in generating embeddings is reflected in Algorithm 3.

Instead of generating embeddings with meta-embeddings, LMA [12] and ROBE [11] use hash functions to map each weight in the embedding table into a shared memory M. For a weight $w_{i,j}$ in the embedding table, they take both i and j as the input of hash functions. LMA utilizes locality sensitive hashing (LSH) to map the weights of each embedding to M randomly. ROBE organizes M as a circular array and divides the flattened embedding table (i.e., concatenates all rows) into blocks of size Z. The head of each block is mapped to M randomly and the following weights in the block will be mapped to the position next to the head.

4.2 Vector Quantization

Hashing methods typically get the index vector by processing the feature id with hash functions, which fail to capture the similarity between features themselves [26]. To capture the similarity, **vector quantization (VQ)** constructs the index vectors through **approximated nearest neighbor search (ANNS)**. Specifically, for a feature with an original embedding of $\boldsymbol{e}$, VQ gets its index vector as $\boldsymbol{I} = \text{one-hot}(\arg\max_k \text{sim}(\boldsymbol{e}, \mathbf{T}_k)) \in \{0,1\}^m$, where $\mathbf{T}_k$ is the k-th meta-embedding in the meta-table $\mathbf{T} \in \mathbb{R}^{m \times d}$ and $\text{sim}(\cdot)$ is a similarity function (e.g., Euclidean distance). In other words, VQ takes the original embedding as input and quantizes it into its most similar meta-embedding. Note that the meta-table and the meta-embedding are commonly referred to as *codebook* and *codeword* in recent literature on VQ. Here, we use *meta-table* and *meta-embedding* for consistency.

Saec [62] generates a meta-table $\mathbf{T}$ by clustering the most frequent embeddings of a pretrained model and then quantizes each original embedding into a meta-embedding in $\mathbf{T}$. However, assigning the same meta-embedding to different features (i.e., collisions) still results in a drop in accuracy, even though the features have some similarity. In addition, Saec cannot optimize the meta-table together with the original embeddings, which also results in suboptimal accuracy. To alleviate the collisions, subsequent works adopt **product quantization (PQ)** [23] and **additive quantization (AQ)** [1] to quantize an embedding into multiple meta-embeddings. To optimize the meta-table together with the original embeddings, researchers usually quantize the original embeddings into meta-embeddings during training and use the meta-embeddings as the input of the following network, where the original embeddings will be optimized through an STE [9].

PQ considers an embedding as a concatenation of several segments (i.e., $\boldsymbol{e} = \bigcup_{i=1}^{s} \boldsymbol{e}^i$). Each segment $\boldsymbol{e}^i$ corresponds to a meta-table $\mathbf{T}^i$. At training, an embedding $\boldsymbol{e}$ is quantized as $\bigcup_{i=1}^{s} \boldsymbol{I}^i \times \mathbf{T}^i$, where $\boldsymbol{I}^i = \text{one-hot}(\arg\max_k \text{sim}(\boldsymbol{e}^i, \mathbf{T}^i_k))$. In other words, PQ quantizes each segment into its most similar meta-embedding in the corresponding meta-table. After training, the original embeddings are discarded and only the meta-tables are preserved. Since the selection of a meta-embedding in a meta-table can be compactly encoded by $\log N$ bits, where N is the size of the meta-table, an embedding can now be stored by $s \log N$ bits with the help of the meta-tables. Further, MGQE [26] takes the feature frequency into consideration when using PQ. Specifically, it divides the embeddings of items into m groups in ascending order of frequency as $\mathbb{G} = \{\mathbf{E}_1, \mathbf{E}_2, \ldots, \mathbf{E}_m\}$ and defines $\mathbb{N} = \{n_1, n_2, \ldots, n_m\}$ where $n_1 < n_2 < \ldots < n_m$. The embeddings in the i-th group can only be quantized into the first n_i meta-embeddings in each meta-table.

Similarly, xLightFM [24] performs PQ in each feature field. Considering that the feature fields have various sizes, xLightFM searches for the optimal size (i.e., the number of meta-embeddings) of the meta-tables for each field. The search process is achieved by the DARTS algorithm, which is similar to the embedding dimension search in Section 3.2.

Similar to PQ, AQ considers an embedding as a summation of s vectors: $\boldsymbol{e} = \sum_{i=1}^{s} \boldsymbol{e}_i$. AQ generates its quantized embeddings by $\sum_{i=1}^{s} \boldsymbol{I}^i \times \mathbf{T}^i$, $\boldsymbol{I}^i = \text{one-hot}(\arg\max_k \text{sim}(\boldsymbol{e} - \sum_{v=1}^{i-1} \mathbf{T}^v_{k_v}, \mathbf{T}^i_k))$, where k_v is the index of the selected meta-embedding in the v-th meta-table. Specifically, the first meta-table takes the embedding $\boldsymbol{e}$ as input and outputs its nearest meta-embedding $\mathbf{T}^1_{k_1}$. The second meta-table then quantizes the residual part $(\boldsymbol{e} - \mathbf{T}^1_{k_1})$ into $\mathbf{T}^2_{k_2}$ and so on. The final output embedding $\hat{\boldsymbol{e}} = \sum_{i=1}^{s} \mathbf{T}^i_{k_i}$. LightRec [34] adopts AQ to compress the item embeddings and uses a pretrained model as a teacher to train the meta-tables effectively. LISA [63] utilizes AQ to compress the DLRMs where self-attention is performed for sequence processing. Note that there is a mass of inner product between embeddings in self-attention which suffers from extremely expensive computing costs. To alleviate this problem, LISA pre-calculates the inner product between meta-embeddings in the same meta-table and stores the results in a small table after training. Then, the inner product of embeddings in self-attention can be calculated by summing the inner product of meta-embeddings, which can accelerate the inference significantly.

4.3 Decomposition

Hashing and vector quantization use one-hot index vectors to perform a hard selection (i.e., selecting only one meta-embedding) in the meta-table and alleviate the collisions between features by maintaining multiple meta-tables. In contrast, decomposition approaches make a soft selection by summing over all the meta-embeddings in a meta-table $\mathbf{T} \in \mathbb{R}^{m \times d}$ with a real-valued index vector $\boldsymbol{I} \in \mathbb{R}^m$. Due to the wide representation space of the real-valued index vectors, one meta-table is sufficient to resolve the collisions between features. Each feature will have a unique index vector stored in the index matrix $\mathbf{I}_M \in \mathbb{R}^{n \times m}$ when formulating the decomposition as $\mathbf{E} = \mathbf{I}_M \times \mathbf{T}$.

MLET [14] factorizes the embedding table $\mathbf{E} \in \mathbb{R}^{n \times d}$ in terms of $\mathbf{I}_M \in \mathbb{R}^{n \times m}$ and $\mathbf{T} \in R^{m \times d}$. In contrast to low-rank decomposition, where $m < d$, MLET decomposes the embedding table into larger matrices (i.e., $m > d$) at training to ensure a larger optimization space. After training, MLET generates the embedding table as $\mathbf{E} = \mathbf{I}_M \times \mathbf{T}$ for memory reduction and fast retrieval. ANT [35] adopts a better initialization for $\mathbf{T}$ and imposes a sparse constraint on $\mathbf{I}_M$. Specifically, ANT initializes the meta-table $\mathbf{T}$ by clustering the embeddings of a pretrained model. In addition, to reduce redundancy, ANT uses an ℓ_1 penalty on $\mathbf{I}_M$ and constrains its domain to be non-negative. Instead of learning and storing the index vectors at training, DHE [27] develops a hash encoder $\mathcal{H} : \mathbb{N} \to \mathbb{R}^m$ to map each feature id into an index vector $\boldsymbol{I} \in \mathbb{R}^m$ on the fly. Specifically, $\mathcal{H}(x) = [h^1(x), h^2(x), \ldots, h^m(x)]$, where $\{h^i\}_{i=1}^m$ is a group of hash functions. With the hash encoder, DHE can eliminate the storage and optimization of $\mathbf{I}_M$. Moreover, considering that the index vectors are deterministic and cannot be optimized, DHE further decomposes the meta-table T into a multi-layer neural network to enhance its expressive ability.

In contrast to the above naïve decomposition, the authors of [56, 64, 71] use **tensor train decomposition (TTD)** to decompose the embedding tables. As shown in Figure 6, the embedding table $\mathbf{E} \in \mathbb{R}^{n \times d}$ will first be reshaped into $\mathcal{E} \in \mathbb{R}^{(n_1 d_1) \times (n_2 d_2) \times \ldots \times (n_s d_s)}$, where $n = \prod_{i=1}^{s} n_i$ and $d = \prod_{i=1}^{s} d_i$. Then, $\mathcal{E}$ will be decomposed as $\mathcal{E} = \mathcal{G}_1 \times \mathcal{G}_2 \times \ldots \times \mathcal{G}_s$, where $\mathcal{G}_i \in \mathbb{R}^{r_{i-1} \times n_i d_i \times r_i}$. $\{\mathcal{G}_i\}_{i=1}^s$ are called TT-cores and $\{r_i\}_{i=0}^s$ are called TT-ranks, in particular, $r_0 = r_s = 1$. TT-Rec [71] is the first to use TTD on the embedding tables of DLRMs. It implements optimized kernels of TTD for embedding tables. LLRec [56] uses TTD on the embedding tables while maintaining the prediction accuracy by knowledge distillation. To enhance the compression capability of TTD, the authors of

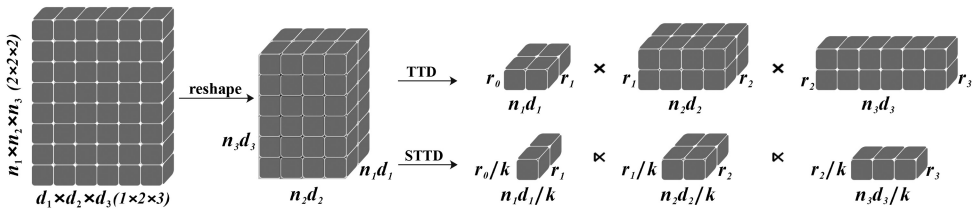

Fig. 6. Example of TTD and STTD, where $r_3 = 1$, $r_1 = r_2 = k = 2$, and r_0 is 1 in TTD and is 2 in STTD.

[64] further develop **semi-tensor product–based tensor train decomposition (STTD)**. Semi-tensor product is a generalization of matrix product. Specifically, given $\boldsymbol{a} \in \mathbb{R}^{1\times np}$ and $\boldsymbol{b} \in \mathbb{R}^{p}$, $\boldsymbol{a}$ can be cut into p blocks $\{\boldsymbol{a}^i \in \mathbb{R}^{1\times n}\}_{i=1}^{p}$ and $\boldsymbol{a} \ltimes \boldsymbol{b} = \sum_{i=1}^{p} \boldsymbol{a}^i \times \boldsymbol{b}_i \in \mathbb{R}^{1\times n}$, where $\ltimes$ is the left semi-tensor product. For matrices $\mathbf{A} \in \mathbb{R}^{h\times np}$ and $\mathbf{B} \in \mathbb{R}^{p\times q}$, $\mathbf{A} \ltimes \mathbf{B} \in \mathbb{R}^{h\times nq}$ contains $h \times q$ blocks and each block is the semi-tensor product between a row of $\mathbf{A}$ and a column of $\mathbf{B}$. The authors of [64] replace the conventional matrix tensor product of TTD with the left semi-tensor product. As Figure 6 shows, in STTD, $\mathcal{E} = \hat{\mathcal{G}}_1 \ltimes \hat{\mathcal{G}}_2 \ltimes \ldots \ltimes \hat{\mathcal{G}}_s$, where $\hat{\mathcal{G}}_i \in \mathbb{R}^{\frac{r_{i-1}}{k}\times\frac{n_i d_i}{k}\times r_i}$,$r_0 = k$ and $r_s = 1$. In addition, the authors of [64] use self-supervised knowledge distillation to reduce accuracy loss from compression.

4.4 Discussion

Weight sharing approaches usually remarkably reduce memory usage. However, they suffer from low efficiency at training due to extra computing cost for generating embeddings, especially the nearest neighbor search in VQ and the matrix multiplication in decomposition approaches. The extra computing cost will also slow down the inference speed except in VQ, where we can store the results of inner product between meta-embeddings to accelerate the inference. Nevertheless, VQ maintains the original embeddings during training, which requires extra memory usage. Moreover, these methods usually cannot improve prediction accuracy, especially hashing, which usually causes a severe drop in accuracy.

5 SUMMARY

Embedding tables usually constitute a large portion of model parameters in DLRMs, which need to be compressed for efficient and economical deployment. As recommender systems continue to grow in scale, embedding compression has attracted more and more attention. In this survey, we provide a comprehensive review of the embedding compression methods in recommender systems, accompanied by a systematic and rational organization of existing studies.

The embedding table can be conceptualized as a matrix with three dimensions: the precision of weights, the dimension of embeddings, and the number of embeddings. Consequently, we classify embedding compression methods into three primary categories according to the dimensions they compress: low precision, mixed dimension, and weight sharing. Low-precision methods reduce the memory of each weight by decreasing its bit width, including binarization and quantization. Mixed-dimension methods reduce the memory of specific embeddings by decreasing their dimensions, including rule-based approaches, NAS-based approaches, and pruning. Weight-sharing methods reduce the actual parameters of the embedding table by sharing weights among different embeddings, including hashing, VQ, and decomposition.

5.1 General Suggestions

In the above sections, we have discussed the pros and cons of different compression methods in detail. However, there are no golden criteria to measure which one is the best. How to choose a

proper compression method depends greatly on the application scenarios and requirements. Therefore, we offer some general suggestions for the common requirements on the key metrics discussed in Section 1.2: model accuracy, inference efficiency, training efficiency, and training memory usage.

- **Model accuracy.** In scenarios that demand high model accuracy, any accuracy degradation caused by compression is deemed unacceptable. In such cases, mixed-dimension methods are recommended as they have been reported to remove redundant parameters and avoid model overfitting. With an appropriate compression ratio, mixed-dimension methods can effectively compress embeddings while maintaining or even improving accuracy. Accuracy can also be preserved when compressing embeddings by quantization with a higher bit width. For instance, using a 16-bit representation has been proven to be sufficient for achieving accurate results. For scenarios that do not require high prediction accuracy, quantization with lower bit width or even binarization (1-bit) can be employed to achieve stronger compression.
- **Inference efficiency.** In scenarios such as online inference, model inference efficiency is of paramount importance. Generally speaking, most embedding compression methods will not have a great negative impact on inference speed. However, in several decomposition methods in which the embedding table is decomposed into multiple small matrices, the process of recovering embeddings may introduce significant inference latency and should be avoided in this context. To improve inference efficiency while compressing embeddings, VQ is suggested, as the feature interaction (e.g., inner product) of embeddings can be pre-calculated to accelerate the inference process. Binarization is also worth considering when there is no high requirement on model accuracy. The calculation of feature interactions between binary embeddings is faster compared with that between full-precision embeddings.
- **Training efficiency.** In scenarios in which the models are supposed to be updated in a timely manner, training efficiency becomes a critical factor. However, it is unfortunate that most embedding compression methods do not contribute to improving training efficiency. In fact, some of them may significantly reduce training efficiency, particularly NAS-based approaches, pruning, VQ, and decomposition. Specifically, NAS-based approaches involve complex calculations to search for optimal embedding dimensions, which can be computationally intensive and time-consuming. Pruning often necessitates retraining to achieve higher accuracy, resulting in additional training overhead. VQ also involves cumbersome calculations for nearest neighbor searches. Decomposition may require multiple matrix multiplications to recover and retrieve embeddings. Therefore, in scenarios that prioritize training efficiency and timely model updates, these methods are not recommended.
- **Training memory usage.** In scenarios in which computing devices have limited memory, it is desirable to compress the training memory usage of embeddings or, at the very least, avoid increasing it. In such cases, we suggest using rule-based approaches, hashing, or decomposition, as they can compress the embedding table before training. The low-precision training of quantization is also worth considering, as the embeddings are stored in the format of integers during training. NAS-based approaches and vector quantization are not recommended in this context. They often require storing a significant number of intermediate results to guide the training process, which will consume more memory.

5.2 Future Prospects

Embedding compression in recommender systems has witnessed rapid development and notable achievements, although there are still several challenging issues that require attention. We identify several potential directions for further research in this field.

— **Low precision.** The key problem faced by low-precision methods is the severe accuracy degradation at extremely lower bit widths. In view of the extensive and advanced research on quantization and binarization in the deep learning community, we can refer to related techniques to alleviate the accuracy loss when compressing embeddings, which is quite challenging and valuable.
— **Mixed dimension.** In recent advanced mixed-dimension methods, there is a need to enhance the training efficiency of NAS-based approaches and pruning. To address this, we recommend designing lighter NAS frameworks that can efficiently search for the optimal embedding dimension. Finding solutions to avoid retraining pruned models is also crucial for enhancing training efficiency. Furthermore, while numerous studies have demonstrated the significant impact of the embedding dimension on model accuracy, there is still a lack of theoretical understanding regarding how the embedding dimension precisely affects model accuracy. Having a solid theoretical basis would be invaluable in guiding the optimal selection of embedding dimensions, enabling more efficient and effective model training.
— **Weight-sharing.** To approach the limit of weight sharing methods, we believe that an intriguing direction to explore is the use of embedding generation networks. Considering the powerful representation capabilities of neural networks, we may learn a powerful neural network to generate embeddings instead of directly learning and maintaining the embeddings themselves.
— **Hybrid approaches.** Since the methods within the three primary categories compress the embeddings from different dimensions and enjoy different advantages, we expect future research to establish a unified method for compressing multiple dimensions or develop hybrid approaches combining these techniques. By integrating the strengths of different compression methods, it is possible to create more powerful and comprehensive compression algorithms.
— **Open benchmarks.** This review offers a thorough discussion of embedding compression methods. However, we did not undertake an experimental comparison across these methods. On one hand, distinct methods are applied to different tasks in recommender systems, each of which has unique accuracy metrics. For example, in click-through rate (CTR) prediction, the commonly used metric is the **Area Under the Curve (AUC)**; whereas for rating prediction, **Root Mean Square Error (RMSE)** and **Mean Absolute Error (MAE)** are typically employed. For Top-N recommendations, **Mean Average Precision (MAP)** and **Normalized Discounted Cumulative Gain (NDCG)** are commonly utilized as accuracy metrics. On the other hand, a majority of research relies on proprietary datasets without sharing open-source code, presenting obstacles to reproducibility and comparative analyses. Nonetheless, the implementation of these methods is not inherently complex. Given the focus on the embedding tables, a solution involves the definition of a new embedding module during implementation coupled with the rewriting of the lookup operation for the embedding vector. Therefore, it is necessary to establish a foundational benchmark to evaluate the effectiveness of distinct methods across a spectrum of tasks, such as BARS [81], a benchmark designed for recommendations. We posit that this would substantially expedite the application and advancement of this field.

REFERENCES

[1] Artem Babenko and Victor S. Lempitsky. 2014. Additive quantization for extreme vector compression. In *2014 IEEE Conference on Computer Vision and Pattern Recognition, CVPR 2014, Columbus, OH, USA, June 23–28, 2014*. IEEE Computer Society, Columbus, OH, USA, 931–938.

[2] Yash Bhalgat, Jinwon Lee, Markus Nagel, Tijmen Blankevoort, and Nojun Kwak. 2020. LSQ+: Improving low-bit quantization through learnable offsets and better initialization. In *2020 IEEE/CVF Conference on Computer Vision and Pattern Recognition, CVPR Workshops 2020, Seattle, WA, USA, June 14–19, 2020*. Computer Vision Foundation / IEEE, 2978–2985.
[3] Tong Chen, Hongzhi Yin, Yujia Zheng, Zi Huang, Yang Wang, and Meng Wang. 2021. Learning elastic embeddings for customizing on-device recommenders. In *KDD'21: The 27th ACM SIGKDD Conference on Knowledge Discovery and Data Mining, Virtual Event, Singapore, August 14–18, 2021*. ACM, 138–147.
[4] Yunchuan Chen, Lili Mou, Yan Xu, Ge Li, and Zhi Jin. 2016. Compressing neural language models by sparse word representations. In *Proceedings of the 54th Annual Meeting of the Association for Computational Linguistics, ACL 2016, August 7–12, 2016, Berlin, Germany, Volume 1: Long Papers*. The Association for Computer Linguistics.
[5] Yankai Chen, Yifei Zhang, Yingxue Zhang, Huifeng Guo, Jingjie Li, Ruiming Tang, Xiuqiang He, and Irwin King. 2021. Towards low-loss 1-bit quantization of user-item representations for top-K recommendation. *CoRR* abs/2112.01944 (2021).
[6] 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 2016, Boston, MA, USA, September 15, 2016*. ACM, 7–10.
[7] Weiyu Cheng, Yanyan Shen, and Linpeng Huang. 2020. Differentiable neural input search for recommender systems. *CoRR* abs/2006.04466 (2020).
[8] Tejalal Choudhary, Vipul Kumar Mishra, Anurag Goswami, and Jagannathan Sarangapani. 2020. A comprehensive survey on model compression and acceleration. *Artif. Intell. Rev.* 53, 7 (2020), 5113–5155.
[9] Matthieu Courbariaux, Yoshua Bengio, and Jean-Pierre David. 2015. BinaryConnect: Training deep neural networks with binary weights during propagations. In *Advances in Neural Information Processing Systems 28: Annual Conference on Neural Information Processing Systems 2015, December 7-12, 2015, Montreal, Quebec, Canada*. 3123–3131.
[10] 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 *WSDM'21, The 14th ACM International Conference on Web Search and Data Mining, Virtual Event, Israel, March 8–12, 2021*. ACM, 922–930.
[11] Aditya Desai, Li Chou, and Anshumali Shrivastava. 2022. Random offset block embedding (ROBE) for compressed embedding tables in deep learning recommendation systems. *Proceedings of Machine Learning and Systems* 4 (2022), 762–778.
[12] Aditya Desai, Yanzhou Pan, Kuangyuan Sun, et al. 2021. Semantically constrained memory allocation (SCMA) for embedding in efficient recommendation systems. *CoRR* abs/2103.06124 (2021).
[13] Steven K. Esser, Jeffrey L. McKinstry, Deepika Bablani, Rathinakumar Appuswamy, and Dharmendra S. Modha. 2020. Learned step size quantization. In *8th International Conference on Learning Representations, ICLR 2020, Addis Ababa, Ethiopia, April 26–30, 2020*. OpenReview.net.
[14] Benjamin Ghaemmaghami, Zihao Deng, Benjamin Y. Cho, et al. 2020. Training with multi-layer embeddings for model reduction. *CoRR* abs/2006.05623 (2020).
[15] 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 2021, Melbourne, Australia, July 12–20, 2021*. IEEE, 2786–2791.
[16] Hui Guan, Andrey Malevich, Jiyan Yang, Jongsoo Park, and Hector Yuen. 2019. Post-training 4-bit quantization on embedding tables. *CoRR* abs/1911.02079 (2019).
[17] Huifeng Guo, Wei Guo, Yong Gao, Ruiming Tang, Xiuqiang He, and Wenzhi Liu. 2021. ScaleFreeCTR: MixCache-based distributed training system for CTR models with huge embedding table. In *SIGIR'21: The 44th International ACM SIGIR Conference on Research and Development in Information Retrieval, Virtual Event, Canada, July 11–15, 2021*. ACM, 1269–1278.
[18] 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 2017, Melbourne, Australia, August 19–25, 2017*. ijcai.org, 1725–1731.
[19] Manish Gupta and Puneet Agrawal. 2022. Compression of deep learning models for text: A survey. *ACM Trans. Knowl. Discov. Data* 16, 4 (2022), 61:1–61:55.
[20] Kai Han, Yunhe Wang, Qi Tian, Jianyuan Guo, Chunjing Xu, and Chang Xu. 2020. GhostNet: More features from cheap operations. In *2020 IEEE/CVF Conference on Computer Vision and Pattern Recognition, CVPR 2020, Seattle, WA, USA, June 13–19, 2020*. 1577–1586.
[21] Xin He, Kaiyong Zhao, and Xiaowen Chu. 2021. AutoML: A survey of the state-of-the-art. *Knowledge-Based Systems* 212 (2021), 106622.
[22] Eric Jang, Shixiang Gu, and Ben Poole. 2017. Categorical reparameterization with Gumbel-Softmax. In *5th International Conference on Learning Representations, ICLR 2017, Toulon, France, April 24–26, 2017, Conference Track Proceedings*. OpenReview.net.

[23] Hervé Jégou, Matthijs Douze, and Cordelia Schmid. 2011. Product quantization for nearest neighbor search. *IEEE Trans. Pattern Anal. Mach. Intell.* 33, 1 (2011), 117–128.

[24] Gangwei Jiang, Hao Wang, Jin Chen, Haoyu Wang, Defu Lian, and Enhong Chen. 2021. xLightFM: Extremely memory-efficient factorization machine. In *SIGIR'21: The 44th International ACM SIGIR Conference on Research and Development in Information Retrieval, Virtual Event, Canada, July 11–15, 2021*. ACM, 337–346.

[25] 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 *KDD'20: The 26th ACM SIGKDD Conference on Knowledge Discovery and Data Mining, Virtual Event, CA, USA, August 23–27, 2020*. ACM, 2387–2397.

[26] 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 of The 2020 Web Conference 2020, Taipei, Taiwan, April 20–24, 2020*. ACM / IW3C2, 562–566.

[27] Wang-Cheng Kang, Derek Zhiyuan Cheng, Tiansheng Yao, Xinyang Yi, Ting Chen, Lichan Hong, and Ed H. Chi. 2021. Learning to embed categorical features without embedding tables for recommendation. In *KDD'21: The 27th ACM SIGKDD Conference on Knowledge Discovery and Data Mining, Virtual Event, Singapore, August 14–18, 2021*. ACM, 840–850.

[28] Wang-Cheng Kang and Julian John McAuley. 2019. Candidate generation with binary codes for large-scale top-N recommendation. In *Proceedings of the 28th ACM International Conference on Information and Knowledge Management, CIKM 2019, Beijing, China, November 3–7, 2019*. ACM, 1523–1532.

[29] Farhan Khawar, Xu Hang, Ruiming Tang, Bin Liu, Zhenguo Li, and Xiuqiang He. 2020. AutoFeature: Searching for feature interactions and their architectures for click-through rate prediction. In *CIKM'20: The 29th ACM International Conference on Information and Knowledge Management, Virtual Event, Ireland, October 19–23, 2020*. ACM, 625–634.

[30] Houyi Li, Zhihong Chen, Chenliang Li, Rong Xiao, Hongbo Deng, Peng Zhang, Yongchao Liu, and Haihong Tang. 2021. Path-based deep network for candidate item matching in recommenders. In *SIGIR'21: The 44th International ACM SIGIR Conference on Research and Development in Information Retrieval, Virtual Event, Canada, July 11–15, 2021*. ACM, 1493–1502.

[31] 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 *37th AAAI Conference on Artificial Intelligence, AAAI 2023, 35th Conference on Innovative Applications of Artificial Intelligence, IAAI 2023, 13th Symposium on Educational Advances in Artificial Intelligence, EAAI 2023, Washington, DC, USA, February 7–14, 2023*. AAAI Press, 4435–4443.

[32] Defu Lian, Yong Ge, Fuzheng Zhang, Nicholas Jing Yuan, Xing Xie, Tao Zhou, and Yong Rui. 2015. Content-aware collaborative filtering for location recommendation based on human mobility data. In *2015 IEEE International Conference on Data Mining, ICDM 2015, Atlantic City, NJ, USA, November 14–17, 2015*. IEEE Computer Society, 261–270.

[33] Defu Lian, Rui Liu, Yong Ge, Kai Zheng, Xing Xie, and Longbing Cao. 2017. Discrete content-aware matrix factorization. In *Proceedings of the 23rd ACM SIGKDD International Conference on Knowledge Discovery and Data Mining, Halifax, NS, Canada, August 13–17, 2017*. ACM, 325–334.

[34] Defu Lian, Haoyu Wang, Zheng Liu, Jianxun Lian, Enhong Chen, and Xing Xie. 2020. LightRec: A memory and search-efficient recommender system. In *WWW'20: The Web Conference 2020, Taipei, Taiwan, April 20-24, 2020*. ACM / IW3C2, 695–705.

[35] Paul Pu Liang, Manzil Zaheer, Yuan Wang, and Amr Ahmed. 2021. Anchor & transform: Learning sparse embeddings for large vocabularies. In *9th International Conference on Learning Representations, ICLR 2021, Virtual Event, Austria, May 3-7, 2021*. OpenReview.net.

[36] Bin Liu, Ruiming Tang, Yingzhi Chen, Jinkai Yu, Huifeng Guo, and Yuzhou Zhang. 2019. Feature generation by convolutional neural network for click-through rate prediction. In *The World Wide Web Conference, WWW 2019, San Francisco, CA, USA, May 13–17, 2019*. ACM, 1119–1129.

[37] Han Liu, Xiangnan He, Fuli Feng, Liqiang Nie, Rui Liu, and Hanwang Zhang. 2018. Discrete factorization machines for fast feature-based recommendation. In *Proceedings of the 27th International Joint Conference on Artificial Intelligence, IJCAI 2018, July 13–19, 2018, Stockholm, Sweden*. ijcai.org, 3449–3455.

[38] Hanxiao Liu, Karen Simonyan, and Yiming Yang. 2019. DARTS: Differentiable architecture search. In *7th International Conference on Learning Representations, ICLR 2019, New Orleans, LA, USA, May 6–9, 2019*. OpenReview.net.

[39] Haochen Liu, Xiangyu Zhao, Chong Wang, Xiaobing Liu, and Jiliang Tang. 2020. Automated embedding size search in deep recommender systems. In *Proceedings of the 43rd International ACM SIGIR Conference on Research and Development in Information Retrieval, SIGIR 2020, Virtual Event, China, July 25–30, 2020*. ACM, 2307–2316.

[40] Qiang Liu, Feng Yu, Shu Wu, and Liang Wang. 2015. A convolutional click prediction model. In *Proceedings of the 24th ACM International Conference on Information and Knowledge Management, CIKM 2015, Melbourne, VIC, Australia, October 19–23, 2015*. ACM, 1743–1746.

[41] Siyi Liu, Chen Gao, Yihong Chen, Depeng Jin, and Yong Li. 2021. Learnable embedding sizes for recommender systems. In *9th International Conference on Learning Representations, ICLR 2021, Virtual Event, Austria, May 3–7, 2021*. OpenReview.net.

[42] Xianglong Liu, Junfeng He, Cheng Deng, and Bo Lang. 2014. Collaborative hashing. In *2014 IEEE Conference on Computer Vision and Pattern Recognition, CVPR 2014, Columbus, OH, USA, June 23-28, 2014*. IEEE Computer Society, 2147–2154.

[43] 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, Atlanta, GA, USA, October 17–21, 2022*. ACM, 1399–1409.

[44] Xiao Ma, Liqin Zhao, Guan Huang, Zhi Wang, Zelin Hu, Xiaoqiang Zhu, and Kun Gai. 2018. Entire space multi-task model: An effective approach for estimating post-click conversion rate. In *The 41st International ACM SIGIR Conference on Research & Development in Information Retrieval, SIGIR 2018, Ann Arbor, MI, USA, July 08–12, 2018*, Kevyn Collins-Thompson, Qiaozhu Mei, Brian D. Davison, Yiqun Liu, and Emine Yilmaz (Eds.). ACM, 1137–1140.

[45] Kelong Mao, Jieming Zhu, Jinpeng Wang, Quanyu Dai, Zhenhua Dong, Xi Xiao, and Xiuqiang He. 2021. SimpleX: A Simple and strong baseline for collaborative filtering. In *CIKM'21: The 30th ACM International Conference on Information and Knowledge Management, Virtual Event, Queensland, Australia, November 1–5, 2021*. ACM, 1243–1252.

[46] H. Brendan McMahan, Gary Holt, David Sculley, Michael Young, Dietmar Ebner, Julian Grady, Lan Nie, Todd Phillips, Eugene Davydov, Daniel Golovin, Sharat Chikkerur, Dan Liu, Martin Wattenberg, Arnar Mar Hrafnkelsson, Tom Boulos, and Jeremy Kubica. 2013. Ad click prediction: A view from the trenches. In *The 19th ACM SIGKDD International Conference on Knowledge Discovery and Data Mining, KDD 2013, Chicago, IL, USA, August 11–14, 2013*. ACM, 1222–1230.

[47] 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 2022, MLSys 2022, Santa Clara, CA, USA, August 29–September 1, 2022*. mlsys.org.

[48] Liang Qu, Yonghong Ye, Ningzhi Tang, Lixin Zhang, Yuhui Shi, and Hongzhi Yin. 2022. Single-shot embedding dimension search in recommender system. In *SIGIR'22: The 45th International ACM SIGIR Conference on Research and Development in Information Retrieval, Madrid, Spain, July 11–15, 2022*. ACM, 513–522.

[49] Steffen Rendle. 2010. Factorization machines. In *ICDM 2010, The 10th IEEE International Conference on Data Mining, Sydney, Australia, 14–17 December 2010*. IEEE Computer Society, 995–1000.

[50] Hao-Jun Michael Shi, Dheevatsa Mudigere, Maxim Naumov, and Jiyan Yang. 2020. Compositional embeddings using complementary partitions for memory-efficient recommendation systems. In *KDD'20: The 26th ACM SIGKDD Conference on Knowledge Discovery and Data Mining, Virtual Event, CA, USA, August 23–27, 2020*. ACM, 165–175.

[51] Yixin Su, Rui Zhang, Sarah M. Erfani, and Zhenghua Xu. 2021. Detecting beneficial feature interactions for recommender systems. In *35th AAAI Conference on Artificial Intelligence, AAAI 2021, 33rd Conference on Innovative Applications of Artificial Intelligence, IAAI 2021, The 11th Symposium on Educational Advances in Artificial Intelligence, EAAI 2021, Virtual Event, February 2–9, 2021*. AAAI Press, 4357–4365.

[52] Yixin Su, Yunxiang Zhao, Sarah M. Erfani, Junhao Gan, and Rui Zhang. 2022. Detecting arbitrary order beneficial feature interactions for recommender systems. In *KDD'22: The 28th ACM SIGKDD Conference on Knowledge Discovery and Data Mining, Washington, DC, USA, August 14–18, 2022*. ACM, 1676–1686.

[53] Yang Sun, Fajie Yuan, Min Yang, Guoao Wei, Zhou Zhao, and Duo Liu. 2020. A generic network compression framework for sequential recommender systems. In *Proceedings of the 43rd International ACM SIGIR Conference on Research and Development in Information Retrieval, SIGIR 2020, Virtual Event, China, July 25–30, 2020*. ACM, 1299–1308.

[54] Qiaoyu Tan, Ninghao Liu, Xing Zhao, Hongxia Yang, Jingren Zhou, and Xia Hu. 2020. Learning to hash with graph neural networks for recommender systems. In *WWW'20: The Web Conference 2020, Taipei, Taiwan, April 20–24, 2020*. ACM / IW3C2, 1988–1998.

[55] Jinpeng Wang, Ziyun Zeng, Yunxiao Wang, Yuting Wang, Xingyu Lu, Tianxiang Li, Jun Yuan, Rui Zhang, Hai-Tao Zheng, and Shu-Tao Xia. 2023. MISSRec: Pre-training and transferring multi-modal interest-aware sequence representation for recommendation. In *Proceedings of the 31st ACM International Conference on Multimedia, MM 2023, Ottawa, ON, Canada, 29 October 2023–3 November 2023*. ACM, 6548–6557.

[56] Qinyong Wang, Hongzhi Yin, Tong Chen, Zi Huang, Hao Wang, Yanchang Zhao, and Nguyen Quoc Viet Hung. 2020. Next point-of-interest recommendation on resource-constrained mobile devices. In *WWW'20: The Web Conference 2020, Taipei, Taiwan, April 20–24, 2020*. ACM / IW3C2, 906–916.

[57] Ruoxi Wang, Bin Fu, Gang Fu, and Mingliang Wang. 2017. Deep & cross network for ad click predictions. In *Proceedings of the ADKDD'17, Halifax, NS, Canada, August 13–17, 2017*. ACM, 12:1–12:7.

[58] Ruoxi Wang, Rakesh Shivanna, Derek Zhiyuan Cheng, Sagar Jain, Dong Lin, Lichan Hong, and Ed H. Chi. 2021. DCN V2: Improved deep & cross network and practical lessons for web-scale learning to rank systems. In *WWW'21: The Web Conference 2021, Virtual Event / Ljubljana, Slovenia, April 19–23, 2021*. ACM / IW3C2, 1785–1797.

[59] Zhikun Wei, Xin Wang, and Wenwu Zhu. 2021. AutoIAS: Automatic integrated architecture searcher for click-trough rate prediction. In *CIKM'21: The 30th ACM International Conference on Information and Knowledge Management, Virtual Event, Queensland, Australia, November 1–5, 2021*. ACM, 2101–2110.

[60] 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 Annual International Conference on Machine Learning, ICML 2009, Montreal, Quebec, Canada, June 14-18, 2009 (ACM International Conference Proceeding Series, Vol. 382)*. ACM, 1113–1120.

[61] Le Wu, Xiangnan He, Xiang Wang, Kun Zhang, and Meng Wang. 2023. A survey on accuracy-oriented neural recommendation: From collaborative filtering to information-rich recommendation. *IEEE Trans. Knowl. Data Eng.* 35, 5 (2023), 4425–4445.

[62] Xiaorui Wu, Hong Xu, Honglin Zhang, Huaming Chen, and Jian Wang. 2020. Saec: Similarity-aware embedding compression in recommendation systems. In *APSys'20: 11th ACM SIGOPS Asia-Pacific Workshop on Systems, Tsukuba, Japan, August 24–25, 2020*. ACM, 82–89.

[63] Yongji Wu, Defu Lian, Neil Zhenqiang Gong, Lu Yin, Mingyang Yin, Jingren Zhou, and Hongxia Yang. 2021. Linear-time self attention with codeword histogram for efficient recommendation. In *WWW'21: The Web Conference 2021, Virtual Event / Ljubljana, Slovenia, April 19–23, 2021*. ACM / IW3C2, 1262–1273.

[64] 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 *SIGIR'22: The 45th International ACM SIGIR Conference on Research and Development in Information Retrieval, Madrid, Spain, July 11 - 15, 2022*. ACM, 546–555.

[65] Jun Xiao, Hao Ye, Xiangnan He, Hanwang Zhang, Fei Wu, and Tat-Seng Chua. 2017. Attentional factorization machines: Learning the weight of feature interactions via attention networks. In *Proceedings of the 26th International Joint Conference on Artificial Intelligence, IJCAI 2017, Melbourne, Australia, August 19–25, 2017*. ijcai.org, 3119–3125.

[66] 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 *SIGMOD'21: International Conference on Management of Data, Virtual Event, China, June 20–25, 2021*. ACM, 2404–2409.

[67] 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 *CIKM'21: The 30th ACM International Conference on Information and Knowledge Management, Virtual Event, Queensland, Australia, November 1–5, 2021*. ACM, 3563–3567.

[68] Bencheng Yan, Pengjie Wang, Kai Zhang, Wei Lin, Kuang-Chih Lee, Jian Xu, and Bo Zheng. 2021. Learning effective and efficient embedding via an adaptively-masked twins-based layer. In *CIKM'21: The 30th ACM International Conference on Information and Knowledge Management, Virtual Event, Queensland, Australia, November 1–5, 2021*. ACM, 3568–3572.

[69] Jie Amy Yang, Jianyu Huang, Jongsoo Park, Ping Tak Peter Tang, and Andrew Tulloch. 2020. Mixed-precision embedding using a cache. *CoRR* abs/2010.11305 (2020).

[70] Quanming Yao, Xiangning Chen, James T. Kwok, Yong Li, and Cho-Jui Hsieh. 2020. Efficient neural interaction function search for collaborative filtering. In *WWW'20: The Web Conference 2020, Taipei, Taiwan, April 20–24, 2020*. ACM / IW3C2, 1660–1670.

[71] 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 2021, MLSys 2021, Virtual Event, April 5–9, 2021*. mlsys.org.

[72] 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 *RecSys 2020: 14th ACM Conference on Recommender Systems, Virtual Event, Brazil, September 22–26, 2020*. ACM, 521–526.

[73] Hanwang Zhang, Fumin Shen, Wei Liu, Xiangnan He, Huanbo Luan, and Tat-Seng Chua. 2016. Discrete collaborative filtering. In *Proceedings of the 39th International ACM SIGIR Conference on Research and Development in Information Retrieval, SIGIR 2016, Pisa, Italy, July 17–21, 2016*. ACM, 325–334.

[74] Shuai Zhang, Lina Yao, Aixin Sun, and Yi Tay. 2019. Deep learning based recommender system: A survey and new perspectives. *ACM Comput. Surv.* 52, 1 (2019), 5:1–5:38.

[75] Weinan Zhang, Jiarui Qin, Wei Guo, Ruiming Tang, and Xiuqiang He. 2021. Deep learning for click-through rate estimation. In *Proceedings of the 30th International Joint Conference on Artificial Intelligence, IJCAI 2021, Virtual Event / Montreal, Canada, 19–27 August 2021*. ijcai.org, 4695–4703.

[76] Yan Zhang, Defu Lian, and Guowu Yang. 2017. Discrete personalized ranking for fast collaborative filtering from implicit feedback. In *Proceedings of the 31st AAAI Conference on Artificial Intelligence, February 4–9, 2017, San Francisco, California, USA*. AAAI Press, 1669–1675.

[77] Zhiwei Zhang, Qifan Wang, Lingyun Ruan, and Luo Si. 2014. Preference preserving hashing for efficient recommendation. In *The 37th International ACM SIGIR Conference on Research and Development in Information Retrieval, SIGIR'14, Gold Coast, QLD, Australia, July 06–11, 2014*. ACM, 183–192.

[78] Xiangyu Zhao, 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. (2021), 896–905.

[79] Xiangyu Zhao, Haochen Liu, Hui Liu, Jiliang Tang, Weiwei Guo, Jun Shi, Sida Wang, Huiji Gao, and Bo Long. 2020. Memory-efficient embedding for recommendations. *CoRR* abs/2006.14827 (2020).

[80] 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 International Conference on Knowledge Discovery & Data Mining, KDD 2018, London, UK, August 19–23, 2018*. ACM, 1059–1068.

[81] Jieming Zhu, Quanyu Dai, Liangcai Su, Rong Ma, Jinyang Liu, Guohao Cai, Xi Xiao, and Rui Zhang. 2022. BARS: Towards open benchmarking for recommender systems. In *SIGIR'22: The 45th International ACM SIGIR Conference on Research and Development in Information Retrieval, Madrid, Spain, July 11–15, 2022*. ACM, 2912–2923.

[82] Barret Zoph and Quoc V. Le. 2017. Neural architecture search with reinforcement learning. In *5th International Conference on Learning Representations, ICLR 2017, Toulon, France, April 24–26, 2017, Conference Track Proceedings*. OpenReview.net.

Received 4 October 2022; revised 9 July 2023; accepted 1 December 2023