AutoSrh: An Embedding Dimensionality Search Framework for Tabular Data Prediction

Shuming Kong, Weiyu Cheng, Yanyan Shen, *Member, IEEE,* and Linpeng Huang, *Member, IEEE*

Abstract—Prediction over tabular data is often a crucial task in many real-life applications. Recent advances in deep learning give rise to various deep models for tabular data prediction. A common and essential step in these models is to vectorize raw input features in tabular data into dense embeddings. Choosing a suitable dimension for each feature is challenging yet necessary to improve model's performance and reduce memory cost of model parameters. Existing solutions to embedding dimensionality search always choose dimensions from a restricted candidate set. This restriction improves the search efficiency but would produce suboptimal embedding dimensions that hurt model's predictive performance. In this paper, we develop AutoSrh, a flexible embedding dimensionality search framework that can select varying dimensions for different features through differentiable optimization. The key idea of AutoSrh is to relax the search space to be continuous and optimize the selection of embedding dimensions via gradient descent. After optimization, AutoSrh performs embedding pruning to derive the mixed embedding dimensions and retrains the model to further improve the performance. Extensive experiments on five real-world tabular datasets demonstrate that AutoSrh can achieve better predictive performance than the existing approaches with 1.1∼1.6x lower training time cost and reserve model's predictive performance while reducing 50∼95% embedding parameters.

Index Terms—Tabular data prediction, embedding dimensionality search, representation learning

1 Introduction

Tabular data is the most commonly used data type in real-world industries [1], [2]. It captures a huge wealth of information that is essential for making data-driven predictive analytics in various critical data science applications, such as rating prediction in recommender system [1], [2], [3], click-through rate (CTR) prediction in online advertising [4], [5], medical treatment [6] and fraud detection [7], etc. In general, as shown in Fig. 1, tabular data is constructed as a logic table, where each row represents a sample and each column represents a distinct feature field. Predictive analytics over tabular data is to estimate a function that maps the rows of features in the determinant feature columns to predict the corresponding values in a target column.

Due to the tremendous success of deep learning in images, audio and text data [8], [9], [10], [11], [12], there have been numerous research interests in extending this success to the tabular domain [4], [13], [14], [15], [16], [17], [18]. Neural networks have the potential to be end-to-end learners, alleviating the labor of manual feature encoding or complex feature engineering [4], [15]. However, distinct from raw features that are naturally found in images and audios, each tabular instance contains heterogeneous features that represent a mixture of dense numerical and sparse categorical values, and these values can be correlated or independent with no inherent positional information. To build predictive models with these heterogeneous features, a common practice of existing deep learning-based approaches is to first map input features into real-valued dense vectors via a feature embedding layer, and then feed them into a well-designed inference module to obtain the prediction results. Designing a suitable embedding layer plays a tremendously impactful role for building general deep models in tabular domain since it is the basis of a deep model on tabular data and directly affects model's predictive performance [19]. However, the majority of previous researches [4], [17], [20], [21], [22] have focused on designing different network architectures for the inference module while the embedding layer remains not well-studied, which misses the opportunity to further improve the model performance on tabular data [19].

The traditional feature embedding layer adopts a uniform and fixed embedding dimension for all the input features, and treats the dimension as a hyperparameter that needs to be tuned via grid search on a validation set. Unfortunately, when dealing with real-world tabular data which has a huge number of features and heavily skewed feature frequencies (e.g., there are millions of distinct features in MovieLens [23] dataset, and 10% input features appear as many times as the remaining 90% [24]), the fixed feature embedding dimension scheme will encounter two crucial issues. (1) Simply assigning the same dimension for all the features may lose the information of highly predictive features while giving too large representation capacity to less predictive features [25], [26]. In fact, it is desirable to assign lower embedding dimensions to less predictive and infrequent features to avoid the over-fitting issue [26], [27]. Likewise, more predictive and popular features deserve higher embedding dimensions that encourage a larger model capacity to fit them. (2) The embedding dimension determines the number of model parameters required for encoding each feature. Given a large number of input features (e.g., user IDs, item IDs) in real-world tabular data, using uniform and fixed embedding dimensions for all input features would inevitably lead to a gigantic embedding table that will consume hundreds of gigabytes of

• *The authors are with the Department of Computer Science and Engineering, Shanghai Jiao Tong University, Shanghai 200240, China. E-mail: {leinuo123, weiyu_cheng, shenyy, lphuang}@sjtu.edu.cn*

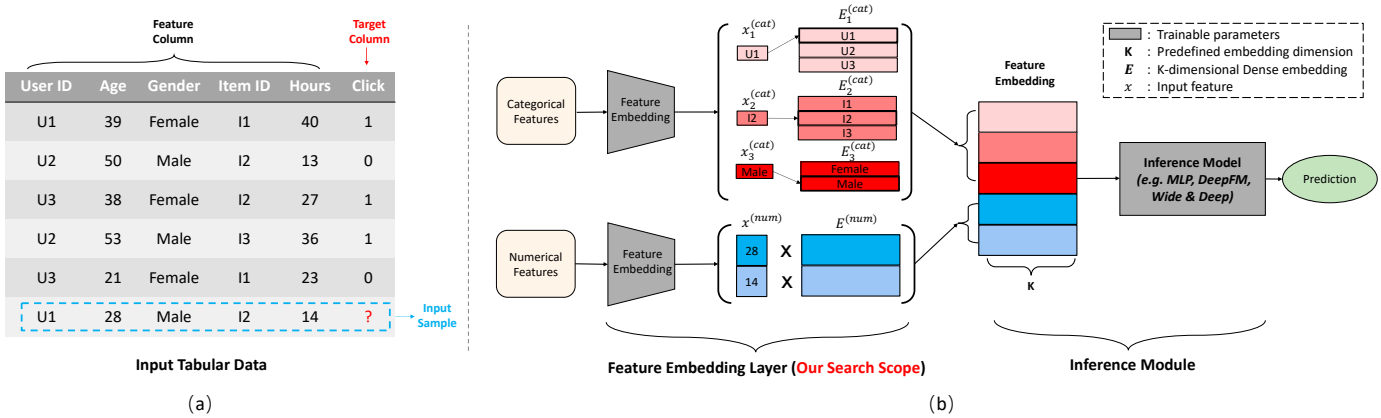


Fig. 1. (a) An example of tabular data with three categorical and two numerical feature columns. (b) The typical deep learning-based model architecture for tabular data prediction. Each individual feature in the same categorical feature field would be converted to a unique feature embedding and the numerical features share an identical field embedding.

memory space [20], [28]. According to the above reasons, it is of significant importance to replace a uniform embedding dimension with varying dimensions to improve the model's predictive performance and reduce the memory usage cost of model parameters.

In this paper, we study the problem of searching for mixed feature embedding dimensions automatically, which is defined as the *Embedding Dimensionality Search (EDS)* problem. There exist two kinds of approaches to dealing with EDS: *reinforcement learning-based* and *AutoML-based*. The reinforcement learning-based approaches [25], [29] divide a base dimension equally into several blocks, and then apply reinforcement learning to assign different features with varying dimensions by generating decision sequences on the selection of dimension blocks. The AutoML-based approaches [24], [26], [27] proposed to automatically assign embeddings with flexible dimensions relying on empirical function [24] or neural architecture search [26], [27]. These methods, however, restrict each feature embedding dimension to be chosen from a small set of candidate dimensions that is predefined [25], [29] or controlled by hyperparameters [24], [26]. Although such restrictions reduce search space to make the search time affordable in practice, the searched embedding dimensions based on a small candidate set would result in suboptimal performance [30]. Note that it is non-trivial to decide a good candidate set to achieve high predictive performance as well as search efficiency.

To address the limitations of existing works, we propose an automatic embedding dimensionality search framework named AutoSrh. Different from the existing works that search embedding dimensions over a predefined set of candidate dimensions, AutoSrh employs an adaptive dimensionality searching process to relax the search space to be continuous and optimizes the selection of embedding dimensions by gradient descent. More specifically, we introduce a *soft selection layer* between the feature embedding layer and inference module to directly control the significance of each dimension of a feature embedding, which could be optimized according to the model's validation performance. Next, we employ a fine-grained embedding pruning procedure to derive the discrete mixed embedding dimension scheme for different input features and finally retrain the model to further improve the predictive performance. Our AutoSrh is agnostic to the architectures of the inference module and thus can be seamlessly applied to various existing deep learning-based models for tabular data prediction. We conduct extensive experiments to demonstrate that our proposed framework is more effective and training efficient than the existing approaches [24], [26], [27], [30], [31].

The major contributions of this paper are summarized as follows:

- We propose AutoSrh, an end-to-end embedding dimensionality search framework for tabular data prediction, which has a flexible search space and is able to generate mixed feature embedding dimensions in a differentiable manner with gradient descent.
- In AutoSrh, we design an adaptive dimensionality search process to relax the search space to be continuous and optimize the selection of embedding dimensions for different features by gradient descent. We also propose a fine-grained pruning procedure to prune less informative embedding dimensions, followed by the model retraining to further improve the predictive performance.
- We conduct experiments with various model architectures on five real-world tabular datasets. The results demonstrate AutoSrh achieves better predictive performance than the existing baselines with 1.1∼1.6x lower training time cost and it also maintains strong predictive performance with 50∼95% embedding parameters pruned.

The remainder of this paper is organized as follows. We introduce preliminaries in Section 2, which is necessary for understanding deep learning models for tabular data prediction. We formalize the problem in Section 3 and elaborate our proposed AutoSrh framework in Section 4. Section 5 provides the experimental results on multiple benchmark datasets. We discuss related work in Section 6 and conclude this paper in Section 7.

2 Preliminaries

2.1 Tabular Data

Tabular data refers to the standardized data storage and analytics format in most industrial applications [4], [14],

[32], [33]. Specifically, tabular data is denoted as a logic table $\mathcal{T}$ with m determinant feature columns $\mathcal{F} = \{\mathcal{F}_i\}_{i=1}^{m}$ and one label column $\mathcal{Y}$. Each row denotes a sample which is represented as $(\boldsymbol{x}, y) = (x_1, x_2, ..., x_m, y)$, where $y \in \mathcal{Y}$ is the predictive value in the label column, $x_i \in \mathcal{F}_i$ is the feature value in the i-th feature column. Each categorical feature column consists of a collection of discrete feature values and each numerical column contains scalar feature values. Table 1 summarizes the key notations used throughout the paper.

2.2 Deep Models in the Tabular Domain

Existing works [1], [4], [16], [33], [34], [35] have been increasingly interested in designing deep learning-based models for tabular data prediction. In general, these models consist of two parts: one feature embedding layer, followed by the inference module.

Feature Embedding Layer. The feature embedding layer transforms all the features in an input instance into dense embedding vectors. Without loss of generality, the input instance $\boldsymbol{x}$ could be converted into a high-dimensional feature vector for C categorical fields and N numerical fields:

$$\mathbf{x} = [\underbrace{\mathbf{x}_1, \mathbf{x}_2, \cdots, \mathbf{x}_C}_{\text{one-hot vectors}}, \underbrace{x_1, x_2, \cdots, x_N}_{\text{scalars}}], \tag{1}$$

where $\mathbf{x}_i$ is the one-hot vector of the feature value in the i-th categorical field and x_i is the scalar value of the i-th numerical field.

For the feature value $\mathbf{x}_i$ in the i-th categorical field, the feature embedding can be obtained by:

$$\mathbf{e}_i = \mathbf{E}_i^{\top} \mathbf{x}_i, \tag{2}$$

where $\mathbf{E}_i \in \mathbb{R}^{C_i \times K}$ is the embedding matrix for the i-th field, C_i is the number of distinct features in the i-th categorical field and K is embedding dimension. To encode numerical features, the commonly used method is field embedding where all the numerical features in the same field share a uniform field embedding and the feature embedding is computed by:

$$\mathbf{e}_i = \mathbf{e}_i^{num} \cdot x_i, \tag{3}$$

where $\mathbf{e}_i^{num} \in \mathbb{R}^K$ is the field embedding vector for the i-th numerical feature field. By putting all the embedding matrices and vectors together, we have:

$$\mathbf{E} = \{\mathbf{E}_1, \mathbf{E}_2, \cdots, \mathbf{E}_C, \mathbf{e}_1^{num}, \mathbf{e}_2^{num}, \cdots, \mathbf{e}_N^{num}\}, \tag{4}$$

where $\mathbf{E} \in \mathbb{R}^{M \times K}$ represents the model's embedding matrix for all the features and M denotes the total number of feature embeddings. Note that all the numerical features in the same field share a field embedding, thus the total number of distinct feature embeddings equals to the number of categorical features plus the number of numerical feature fields, i.e., $M = \sum_{i=1}^{C} C_i + N$. Given an input sample $\mathbf{x}$, the output of the feature embedding layer is denoted as $\mathbf{X} = \mathbf{E}^{\top}\mathbf{x} = [\mathbf{e}_1, \mathbf{e}_2, ..., \mathbf{e}_m]$, which represents the collection of dense embedding vectors for the input features.

Inference Module. The inference module, which can be implemented by different architectures, essentially composes a parameterized function $\mathcal{G}$ that predicts the value in the target column based on the embedded feature vector $\mathbf{X}$ for the input instance. That is,

$$\hat{y} = \mathcal{G}(\boldsymbol{\theta}, \mathbf{X}), \tag{5}$$

where $\hat{y}$ is the model's prediction, and $\boldsymbol{\theta}$ denotes the set of parameters in the inference module. Prior works have developed various architectures for $\mathcal{G}$, based on the simple inner product function [18], and well-designed deep neural networks [15], [17], [36], [37], [38]. Most of the proposed architectures for the inference module require all the feature embeddings to be in a uniform dimension K. However, recent works [26], [27], [30] have shown that simply assigning uniform embedding dimension for all the input features in many real-world tabular datasets suffers from both effectiveness and memory cost issues.

TABLE 1
A list of commonly used notations.

Notation	Explanation
$\mathcal{T}$	Entire tabular dataset
$\mathbf{d}$	Dimension index vector
$\mathbf{e}$, $\mathbf{E}$	Embedding vector, embedding matrix
K	Base dimension of feature embeddings
M	The number of distinct feature embeddings
$\mathbf{d}_i$, $\mathbf{e}_i$	$\mathbf{d}$, $\mathbf{e}$ of i-th feature in $\mathbf{E}$
D, $\mathcal{D}$	Mixed dimension scheme, the search space of D
$\boldsymbol{\theta}$	The set of parameters in the inference module
Θ	The set of trainable model parameters
η_i	Frequency of feature embedding $\mathbf{e}_i$
L	The number of feature blocks
$\tilde{D}$	Mixed dimension scheme after feature blocking
$\tilde{\mathbf{D}}$	Dimension indicator matrix (the binary form of $\tilde{D}$)
$\boldsymbol{\alpha}$	Soft selection layer, $\boldsymbol{\alpha} \in \mathbb{R}^{L \times K}$
$\boldsymbol{\alpha}_{l*}$	The l-th row in $\boldsymbol{\alpha}$, corresponding to l-th block
$\tilde{\mathbf{e}}_i$	Output embedding of i-th feature in $\mathbf{E}$
$\mathbf{X}$	The collection of output feature embeddings
ξ	Learning rate for one-step update of Θ
$\tilde{\mathbf{E}}$, $\mathbf{E}'$	Output embedding matrix, $\tilde{\mathbf{E}}$ after pruning

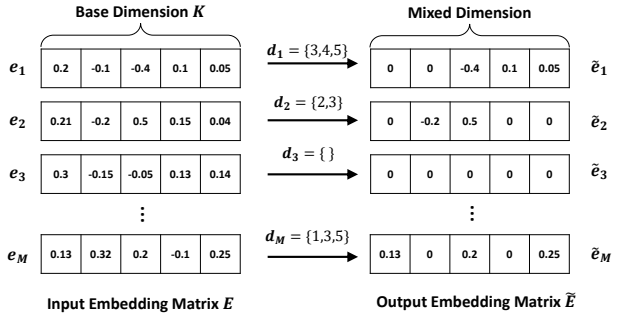


Fig. 2. A demonstration of key notations.

3 Problem Statement

The embedding dimensionality search (EDS) problem (also called embedding size search) has become an increasingly important issue for tabular data prediction, which aims to use embeddings with mixed dimensions to represent different input features towards better predictive performance and a smaller memory footprint caused by storing feature embeddings. It has been pointed out that the existing EDS approaches which restrict each feature embedding dimension to be chosen from a small predefined candidate set

Algorithm 1: The AutoSrh Framework

Input: Training dataset $\mathcal{T}_{train}$, validation dataset $\mathcal{T}_{val}$, learning rate ξ, batch size n
Output: The well-learned soft selection layer $\boldsymbol{\alpha}$, mixed dimension scheme D, model parameters Θ^*

1 $(\boldsymbol{\alpha}, \Theta) \leftarrow$ Derive the continuous mixed dimension scheme by Adaptive Dimensionality Search with $(\mathcal{T}_{train}, \mathcal{T}_{val}, \xi, n)$; // Section 4.2
2 $D \leftarrow$ Derive the discrete mixed dimension scheme by Embedding Pruning with $(\boldsymbol{\alpha}, \Theta)$; // Section 4.3
3 $\Theta^* \leftarrow$ Derive the model parameters by Model Retraining with $(\mathcal{T}_{train}, \Theta, D)$; // Section 4.4

would yield sub-optimal performance [30]. In this paper, we propose to search arbitrary embedding dimensions for different input features in a broader range. Specifically, we define the search space of our mixed embedding dimensions as follows.

Search Space. Suppose we have a *base dimension* K for each feature embedding $\mathbf{e}_i$. To facilitate an arbitrary embedding dimension for $\mathbf{e}_i$ in $[0, K]$, we maintain a *dimension index vector* $\mathbf{d}_i$ which records ordered locations of the feature's involving dimensions from the set $\{1, \cdots, K\}$. Thereafter, as shown in Fig. 2, we could use the index vector $\mathbf{d}_i$ to convert the input feature embedding $\mathbf{e}_i$ into a sparse output embedding vector $\tilde{\mathbf{e}}_i$ by pruning the dimensions which are not in the index vector.

The size of $\mathbf{d}_i$ varies among different feature embeddings to enforce a mixed dimension scheme. Formally, given the input embedding matrix $\mathbf{E} \in \mathbb{R}^{M \times K}$, we define the *mixed dimension scheme* $D = \{\mathbf{d}_1, \cdots, \mathbf{d}_M\}$ to be the collection of dimension index vectors for all the feature embeddings in $\mathbf{E}$. We use $\mathcal{D}$ to denote the *search space* of the mixed dimension scheme D, which includes 2^{MK} possible choices. Given D and $\mathbf{E}$, we can derive the sparse output embedding matrix $\tilde{\mathbf{E}}$ which can be fed into the inference module.

Problem Formulation. The goal of our framework is to find the $D \in \mathcal{D}$ which contains the optimal dimension index vectors for different feature embeddings that can lead to the best inference performance. Formally, letting $\Theta = \{\boldsymbol{\theta}, \mathbf{E}\}$ be the set of trainable model parameters, the model prediction is changed to:

$$\hat{y} = \mathcal{G}(\Theta, D, \mathbf{X}). \tag{6}$$

Our studied problem is to find both model parameters Θ and mixed dimension scheme D such that the training loss is minimized, which can be defined as:

$$\min_{D \in \mathcal{D}, \Theta} \mathcal{L}(\Theta, D, \mathcal{T}_{train}), \tag{7}$$

where $\mathcal{T}_{train} = \{\mathbf{x}, y\}$ represents the training tabular set and $\mathcal{L}$ is the loss function. In this paper, we focus on two most widely-used loss functions in tabular data prediction, i.e., Logloss for classification tasks and Mean Square Error (MSE) for regression tasks. Distinct from the previous EDS approaches [26], [27] which use hard selection to choose feature embedding dimensions from a small candidate dimension set, our approach searches arbitrary embedding dimensions for different features, i.e., in the range of [1, K].

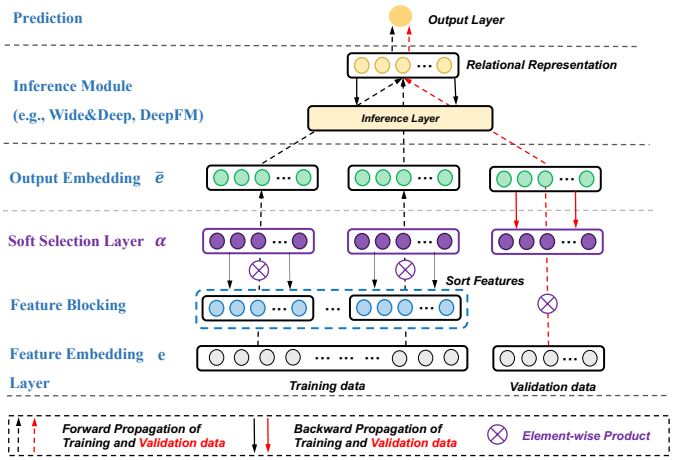


Fig. 3. An illustration of adaptive dimensionality search in AutoSrh.

However, to search an optimal mixed dimension scheme D directly is infeasible due to its discreteness and extremely large search space. To address the problem, we propose AutoSrh to relax the mixed dimension scheme D to be continuous and search the optimal D automatically via gradient descent.

4 Methodology

In this section, we present the AutoSrh framework, which can automatically search the optimal mixed dimension scheme for feature embeddings towards better predictive performance and lower memory cost. In what follows, we first provide the overview of AutoSrh and then elaborate on its details.

4.1 Overview

As described in Algorithm 1, AutoSrh consists of three major stages: *Adaptive Dimensionality Search*, *Embedding Pruning* and *Model Retraining*.

Adaptive Dimensionality Search is to relax the discrete mixed dimension scheme D to be continuous and trainable using a *soft selection layer* $\boldsymbol{\alpha}$. Then, we optimize model parameters Θ and $\boldsymbol{\alpha}$ with the training data and validation data, respectively.

Embedding Pruning is to derive the discrete mixed dimension scheme D after the dimensionality search. We use the learned soft selection layer $\boldsymbol{\alpha}$ and model parameters Θ to compute the feature embeddings $\tilde{\mathbf{e}}_i$. Then we prune the unimportant embedding dimensions in each $\tilde{\mathbf{e}}_i$ according to the memory requirement.

Model Retraining is to retrain the model to further improve the predictive performance. We fix the learned soft selection layer $\boldsymbol{\alpha}$ and update the model parameters Θ in the embedding layer and the inference module.

4.2 Adaptive Dimensionality Search

Learning mixed dimension scheme D directly is infeasible due to its discreteness and extremely large search space. To address the problem, we propose Adaptive Dimensionality Search as illustrated in Fig. 3. Specifically, we apply feature blocking technique to significantly reduce the search

space of D and then perform continuous relaxation to relax the discrete mixed dimension scheme D to be continuous. Thereafter, we employ bi-level optimization to simultaneously learn the continuous dimensions and model parameters. With the learned mixed dimension scheme D, we acquire the significance of each dimension of feature embedding and thus can adaptively assign different dimensions for individual input features in $[1, K]$ without predefining a candidate dimension set. The detailed design of Adaptive Dimensionality Search is introduced as follows.

4.2.1 Feature Blocking

Feature blocking has been a novel ingredient used in the existing EDS methods [24], [25] to facilitate the reduction of search space. The intuition behind is that features with similar frequencies could be grouped into a block sharing the same embedding dimension. Following the existing works, we sort all the feature embeddings in $\mathbf{E}$ in the descending order of frequency (i.e., the number of their feature occurrences in the tabular dataset). Let η_i denote the frequency of feature embedding $\mathbf{e}_i$. We can obtain a sorted feature embedding matrix $\mathbf{E} = [\mathbf{e}'_1, \mathbf{e}'_2, ..., \mathbf{e}'_M]$ such that $\eta_i \geq \eta_j$ for any $i < j$. We then divide $\mathbf{E}$ equally into L blocks, where the feature embeddings in a block share the same dimension index vector $\tilde{d}_i$. We denote by $\tilde{D} = [\tilde{\mathbf{d}}_1, \tilde{\mathbf{d}}_2, ..., \tilde{\mathbf{d}}_L]$ the mixed dimension scheme after feature blocking. Then the length of the mixed dimension scheme $|\tilde{D}|$ becomes L, and the search space size is reduced from 2^{MK} to 2^{LK} accordingly. In practice, M can be up to tens of thousands in real-world tabular datasets due to the existence of categorical feature fields with large domain sizes, e.g., User ID. We set a small value for L to reduce the search space of D significantly, i.e., $L << M$. In our experiments, we set the maximum value of L to be 10 and the empirical results show that the improvement in model's performance diminishes with increasing value of L.

4.2.2 Continuous Relaxation

After feature blocking, we convert our problem of learning the mixed dimension scheme from D to $\tilde{D}$, which significantly reduces the search space. In order to optimize $\tilde{D}$, we first transform $\tilde{D}$ into a binary *dimension indicator matrix* $\tilde{\mathbf{D}} \in \mathbb{R}^{L \times K}$, where each element in $\tilde{\mathbf{D}}$ is either 1 or 0 indicating the existence of the corresponding embedding dimension according to $\tilde{D}$. We then introduce a *soft selection layer* to relax the search space of $\tilde{\mathbf{D}}$ to be continuous and trainable. The soft selection layer is essentially a numerical matrix $\boldsymbol{\alpha} \in \mathbb{R}^{L \times K}$, where each element in $\boldsymbol{\alpha}$ satisfies: $0 \leq \boldsymbol{\alpha}_{l,k} \leq 1$. That is, each binary choice $\tilde{\mathbf{D}}_{l,k}$ (the existence of the k-th embedding dimension in the l-th feature block) in $\tilde{\mathbf{D}}$ is relaxed to be a continuous variable $\boldsymbol{\alpha}_{l,k}$ within the range of $[0, 1]$. We insert the soft selection layer between the feature embedding layer and inference layers, as illustrated in Fig. 3. Given $\boldsymbol{\alpha}$ and the embedding matrix $\mathbf{E}$, the output embedding $\tilde{\mathbf{e}}_i$ in the l-th block produced by the bottom two layers can be computed as follows:

$$\tilde{\mathbf{e}}_i = \mathbf{e}'_i \odot \boldsymbol{\alpha}_{l*}, \tag{8}$$

where $\boldsymbol{\alpha}_{l*} \in \mathbb{R}^K$ is the l-th row in $\boldsymbol{\alpha}$, and $\odot$ is the element-wise product. By applying Equation (8) to each row in $\mathbf{E}$, we can obtain the output feature embedding matrix $\tilde{\mathbf{E}}$. Given an instance $(\boldsymbol{x}, y) = (x_1, x_2, ..., x_m, y)$ in the tabular dataset we use $\tilde{\mathbf{E}}$ to output its embedding vector as $\tilde{\mathbf{X}} = \tilde{\mathbf{E}}^\top \mathbf{x} = [\tilde{\mathbf{e}}_1, \tilde{\mathbf{e}}_2, ..., \tilde{\mathbf{e}}_m]$. Finally, we feed the output feature embedding into the inference module to produce the prediction result:

$$\hat{y} = \mathcal{G}(\boldsymbol{\theta}, \tilde{\mathbf{X}}), \tag{9}$$

where θ and $\mathcal{G}$ denotes the parameters and the inference module of the model.

4.2.3 Bi-level Optimization

After continuous relaxation, we relax the mixed dimension scheme $\tilde{D}$ (after feature blocking) via the soft selection layer $\boldsymbol{\alpha}$, thus our problem stated in Equation (7) can be transformed into:

$$\min_{\boldsymbol{\alpha}, \Theta} \mathcal{L}(\Theta, \boldsymbol{\alpha}, \mathcal{T}_{train}), \tag{10}$$

where $\mathcal{T}_{train}$ is the training tabular dataset and $\Theta = \{\boldsymbol{\theta}, \mathbf{E}\}$ represents model parameters in both the embedding layer and inference layers. However, we could not simultaneously find the optimal parameters Θ and $\boldsymbol{\alpha}$ by optimizing Equation (10) with the training dataset since the training model is non-convex and the parameters Θ and $\boldsymbol{\alpha}$ are highly dependent on each other. Otherwise, it would cause the model over-fitting issue on the training dataset [39], [40]. Inspired by the Hyperparameters Optimization (HPO) techniques [41], we treat the parameters of the soft selection layer $\boldsymbol{\alpha}$ as the hyperparameters and perform bi-level optimization as follows:

$$\begin{aligned} \min_{\boldsymbol{\alpha}} \quad & \mathcal{L}(\Theta^*, \boldsymbol{\alpha}, \mathcal{T}_{val}) \\ \text{s.t.} \quad & \Theta^* = \operatorname*{argmin}_{\Theta} \mathcal{L}(\Theta, \boldsymbol{\alpha}, \mathcal{T}_{train}) \wedge \boldsymbol{\alpha}_{k,j} \in [0, 1], \end{aligned} \tag{11}$$

where $\mathcal{T}_{train}$ is used to optimize the parameters Θ and a small unbiased validation set $\mathcal{T}_{val}$ is used to optimize the hyperparameters $\boldsymbol{\alpha}$. Equation (11) essentially defines a bi-level optimization problem [42], which has been studied and empirically verified to obtain the better model generalization ability in gradient-based hyperparameter optimization approaches [39], [40], [43]. Basically, $\boldsymbol{\alpha}$ and Θ are respectively treated as the upper-level and lower-level variables to be optimized in an interleaving way. Since the scale parameter Θ is much bigger than $\boldsymbol{\alpha}$, training the model until convergence to get the optimal Θ^* in each iteration is computationally expensive. To address this challenge, we follow the general network training trick that employs stochastic gradient descent (SGD) to optimize the parameter Θ. Specifically, in each iteration of training, a mini-batch of training samples $\{(x_i, y_i), 1 \leq i \leq n\}$ is sampled, where n is the mini-batch size. Then we approximate Θ^* by adapting Θ using a single training step and the updating equation of the model parameters Θ on the mini-batch training samples can be formulated as

$$\Theta^* \approx \Theta - \xi \nabla_\Theta \mathcal{L}_{train}(\Theta, \boldsymbol{\alpha}), \tag{12}$$

where ξ is the learning rate for one-step update of model parameters Θ. After receiving the feedback of the model parameters updating formulation from Equation (12), the

Algorithm 2: Adaptive Dimensionality Search

Input: Training dataset $\mathcal{T}_{train}$, validation dataset $\mathcal{T}_{val}$, learning rate ξ, batch size n
Output: The soft selection layer $\boldsymbol{\alpha}$, embedding matrix $\mathbf{E}$, inference module parameters $\boldsymbol{\theta}$

Sort features in $\tilde{\mathcal{F}}$ and divide them into L blocks;
Initialize the soft selection layer $\boldsymbol{\alpha}$ to be an all-one matrix, and randomly initialize Θ; // $\Theta = \{\boldsymbol{\theta}, \mathbf{E}\}$
while *not converged* **do**
  $\{(\boldsymbol{x}, y)\} \leftarrow$ SampleMiniBatch($\mathcal{T}_{train}$, n);
  $\{(\boldsymbol{x}_{val}, y_{val})\} \leftarrow$ SampleMiniBatch($\mathcal{T}_{val}$, n);
  Update trainable parameters Θ by descending $\nabla_{\Theta}\mathcal{L}_{train}(\Theta, \boldsymbol{\alpha})$;
  Calculate the gradients of $\boldsymbol{\alpha}$ using Equation (13): $-\xi\nabla^2_{\boldsymbol{\alpha},\Theta}\mathcal{L}_{train}(\Theta, \boldsymbol{\alpha}) \cdot \nabla_{\Theta'}\mathcal{L}_{val}(\Theta', \boldsymbol{\alpha}) + \nabla_{\boldsymbol{\alpha}}\mathcal{L}_{val}(\Theta', \boldsymbol{\alpha})$;
  Perform Equation (14) to normalize the gradients in $\boldsymbol{\alpha}$;
  Update $\boldsymbol{\alpha}$ by descending the gradients, and then clip its values into the range of $[0, 1]$;
end while

parameter $\boldsymbol{\alpha}$ of the soft selection layer would be readily updated on a mini-batch of validation data with the following gradient:

$$\begin{aligned}&\nabla_{\boldsymbol{\alpha}}\mathcal{L}_{val}(\Theta - \xi\nabla_{\Theta}\mathcal{L}_{train}(\Theta, \boldsymbol{\alpha}), \boldsymbol{\alpha}) \\ =&\nabla_{\boldsymbol{\alpha}}\mathcal{L}_{val}(\Theta', \boldsymbol{\alpha}) - \xi\nabla^2_{\boldsymbol{\alpha},\Theta}\mathcal{L}_{train}(\Theta, \boldsymbol{\alpha}) \cdot \nabla_{\Theta'}\mathcal{L}_{val}(\Theta', \boldsymbol{\alpha}),\end{aligned} \tag{13}$$

where $\Theta' = \Theta - \xi\nabla_{\Theta}\mathcal{L}_{train}(\Theta, \boldsymbol{\alpha})$ denotes the model parameters after one-step update. Equation (13) can be solved efficiently using the existing deep learning libraries that allow automatic differentiation, such as Pytorch [44] or TensorFlow [45]. The second-order derivative term in Equation (13) can be omitted to further improve computational efficiency considering ξ to be near zero, which is called the *first-order approximation*. Algorithm 2 summarizes the bi-level optimization procedure for solving Equation (11) via model training.

4.2.4 Gradient Normalization

During the optimization of $\boldsymbol{\alpha}$ by the gradient in Equation (13), we propose a gradient normalization technique to normalize the row-wise gradients of $\boldsymbol{\alpha}$ over each training batch:

$$\mathbf{g}_{norm}(\boldsymbol{\alpha}_{l*}) = \frac{\mathbf{g}(\boldsymbol{\alpha}_{l*})}{\Sigma_{k=1}^{K}|\mathbf{g}(\boldsymbol{\alpha}_{l,k})|/K + \epsilon_{\mathbf{g}}}, \quad k \in [1, K], \tag{14}$$

where $\mathbf{g}$ and $\mathbf{g}_{norm}$ denote the gradients before and after normalization respectively, and $\epsilon_{\mathbf{g}}$ is a small value (e.g., 1e-7) to avoid numerical overflow. Here we use row-wise gradient normalization to deal with the high variance of the gradients of $\boldsymbol{\alpha}$ during backpropogation. More specifically, $g(\boldsymbol{\alpha}_{l*})$ of a high-frequency feature block can be several orders of magnitude larger than that of a low-frequency feature block due to their difference on the number of related data samples. By normalizing the gradients for each block, we can apply the same learning rate to different rows of $\boldsymbol{\alpha}$ during optimization. Otherwise, a single learning rate shared by different feature blocks will fail to optimize most rows of $\boldsymbol{\alpha}$.

Algorithm 3: Dimensionality Pruning

Input: Θ, $\boldsymbol{\alpha}$ obtained by Algorithm 2
Output: Mixed dimension $\tilde{D}$

Calculate the output embedding matrix $\tilde{\mathbf{E}}$ using $\boldsymbol{\alpha}$ and $\mathbf{E}$ according to Equation (8);
Prune $\tilde{\mathbf{E}}$ into a sparse matrix $\mathbf{E}'$ following Equation (15);
Derive the mixed dimension scheme $\tilde{D}$ with $\mathbf{E}'$;

4.3 Fine-grained Embedding Pruning

After differentiable dimensionality search, we have the learned parameters for $\boldsymbol{\theta}$, $\mathbf{E}$ and $\boldsymbol{\alpha}$. A straightforward way to derive the discrete mixed dimension scheme $\tilde{D}$ is to prune insignificant embedding dimensions in the soft selection layer $\boldsymbol{\alpha}$. Here we employ a fine-grained pruning procedure. Specifically, for feature f_i in the l-th block, we can compute its output embedding $\tilde{\mathbf{e}}_i$ with $\mathbf{e}_i$ and $\boldsymbol{\alpha}_{l*}$ following Equation (8). We collect the output embeddings $\{\tilde{\mathbf{e}}_i\}$ for all the features in $\mathcal{F}$ and form an output embedding matrix $\tilde{\mathbf{E}} \in \mathbb{R}^{M \times K}$. We then prune non-informative embedding dimensions in $\tilde{\mathbf{E}}$ as follows:

$$\mathbf{E}'_{i,j} = \begin{cases} 0, & \text{if } |\tilde{\mathbf{E}}_{i,j}| < \epsilon \\ \tilde{\mathbf{E}}_{i,j}, & \text{otherwise} \end{cases}, \tag{15}$$

where ϵ is a threshold that can be manually tuned according to the requirement on pruned embedding parameter size. In practice, we define CR (Compression Rate) as the ratio between the number of embedding parameters (i.e., MK) and the number of remaining embedding parameters after pruning. The value of CR is set according to the actual requirement on memory usage and practitioners could adjust ϵ to prune embedding parameters until the given CR is reached. For example, when we set CR=20 (only 0.05% embedding parameters remain), we could set ϵ to a value which is larger than 95% of the embedding values in the output embedding matrix $\tilde{\mathbf{E}}$. This allows us to prune 95% embedding parameters to achieve the compression ratio.

The pruned output embedding matrix $\mathbf{E}'$ is sparse and used to derive the discrete mixed dimension scheme $\tilde{D}$ by recording the indices of non-zero elements in the output embeddings. With fine-grained pruning, the derived embedding dimensions can be different even for features in the same feature block, resulting in a more flexible mixed dimension scheme. Recall that real-world tabular data generally contains a large number of distinct features and the numerous features result in a gigantic embedding table that incurs high memory cost during model serving [30]. Note that the embedding parameters can dominate the total model parameters [24], [26], [27]. Our approach can prune insignificant embedding dimensions and save memory usage with the sparse matrix storage technique [46]. Algorithm 3 summarizes the embedding pruning procedure for deriving the discrete mixed dimension scheme $\tilde{D}$.

TABLE 2
Statistics of the datasets.

	Dataset	#Samples #Train	#Test	#Fields	#Features	Task	AutoSrh Hyperparameters
CTR Prediction	Avazu	32,343,174	4,042,897	22	1,544,488	Classification	$K = 64$, $L = 6$, $n = 4096$
	Criteo	36,672,494	4,584,062	39	2,086,936	Classification	$K = 64$, $L = 6$, $n = 4096$
	KDD12	119,711,284	14,963,911	13	8,429,196	Classification	$K = 16$, $L = 6$, $n = 4096$
Rating Prediction	MovieLens	16,000,210	2,000,026	2	165,771	Regression	$K = 64$, $L = 10$, $n = 4096$
	Amazon	2,578,193	322,274	2	187,966	Regression	$K = 64$, $L = 10$, $n = 4096$

Algorithm 4: Model Retraining

Input: Training dataset $\mathcal{T}_{train}$, batch size n, model parameters $\boldsymbol{\alpha}$, $\Theta = \{\boldsymbol{\theta}, \mathbf{E}'\}$
Output: Model parameters Θ^*

```
Sort features in F̃ and divide them into L blocks;
Fix parameters of soft selection layer α;
while not converged do
    {(x, y)} ← SampleMiniBatch(T_train, n);
    Update trainable parameters Θ by descending
      ∇_Θ L_train(Θ, α);
end while
Derive the Θ* with D̃;
```

4.4 Model Retraining

After the fine-grained embedding pruning, we derive the dimension for each feature embedding. In this section, we introduce model retraining to update the mode parameters Θ with the learned mixed dimension scheme $\tilde{D}$ to further improve the model performance. The model retraining algorithm in AutoSrh is summarized in Algorithm 4 and all the inputs to the retraining process are obtained by Algorithm 2 and 3. Specifically, we use the training loss to update the model parameters $\Theta = \{\theta, E\}$ through back-propagation until convergence and fix the parameters $\boldsymbol{\alpha}$ in the soft selection layer during the retraining process. The intuition behind is that the model parameters Θ would be affected by the suboptimal embedding dimensions during the dimensionality search stage, and retraining the model with the learned embedding dimensions is useful to alleviate such a negative impact. Finally, we obtain model parameters Θ^* with the discrete mixed dimension scheme $\tilde{D}$. We also empirically show that model retraining can effectively improve the predictive performance in Section 5.3.

5 Experiments

In this section, we conduct experiments to answer the following research questions:

- **RQ1:** How does our proposed AutoSrh perform compared with the existing dimensionality search algorithms in different prediction tasks?
- **RQ2:** What is the training efficiency of our AutoSrh and the baselines?
- **RQ3:** How do the hyperparameters of AutoSrh affect its performance?
- **RQ4:** How does model retraining affect the model performance?
- **RQ5:** What do we learn from the search results of AutoSrh?

5.1 Experimental Settings

5.1.1 Datasets

We evaluate AutoSrh on two important real-world applications of tabular data prediction with five large-scale tabular datasets, including click-through rate prediction (Avazu[1], Criteo[2], KDD12[3] for classification task) and rating prediction (Movielens-20M[4], Amazon[5] for regression task). For each dataset, we randomly split the dataset by 8:1:1 to obtain the training, validation and test sets. The statistics of the five datasets are summarized in Table 2 together with the hyperparameters settings for AutoSrh.

• **Avazu.** It is a public click-through rate (CTR) benchmark dataset provided by the mobile advertising platform Avazu, which contains users' mobile behaviors and aims to predict whether a displayed mobile ad is clicked by a user or not. The Avazu dataset contains 22 feature fields spanning from user/device features to ad attributes and 40,428,967 samples with 1,544,488 numerical/categorical input features.

• **Criteo.** It is a popular industry benchmark dataset for CTR prediction, which contains millions of users' clicking records on displayed ads. The classification task is to predict whether a given ad under the page context is clicked by a user or not. The Criteo dataset comprises 13 numerical feature fields and 26 categorical feature fields. We used the version containing 45,840,617 samples with 2,086,936 distinct numerical/categorical input features.

• **KDD12.** It is a large-scale tabular dataset released by KDDCup 2012, which aims to predict whether the user will click on a given ad with the session logs of each user derived from the Tencent proprietary search engine. The KDD12 dataset has 139,639,105 samples of 13 feature fields with 6,109,086 distinct numerical/categorical input features.

• **Movielens-20M.** It is a real-world movie recommendation dataset containing more than 20 million user ratings ranging from 1 to 5 on movies. The regression task is to predict movie ratings given the feature field user ID and movie ID. There are 20,000,263 samples in total with 165,771 unique input features.

1. https://www.kaggle.com/c/avazu-ctr-prediction
2. https://www.kaggle.com/c/criteo-display-ad-challenge/
3. https://www.kaggle.com/c/kddcup2012-track2
4. https://grouplens.org/datasets/movielens/20m/
5. https://jmcauley.ucsd.edu/data/amazon/

• **Amazon.** It is a real-world Amazon-Electronics product dataset containing reviews spanning May 1996 - July 2014. The regression task is to predict the product ratings given the feature field user ID and item ID. There are 7,704,596 samples in total with 1,338,298 unique input features.

5.1.2 Evaluation Metrics

We adopt AUC (Area Under the ROC Curve) and Logloss for classification task, and use MSE (Mean Squared Error) for regression task. For both MSE, AUC and Logloss, an improvement at 0.001 level is considered to be significant on the adopted benchmark datasets [4], [15]. In addition to predictive performance, following the previous embedding dimensionality search approach [4], we also report the "Params" metric (i.e., the total number of embedding parameters selected by the algorithm) and the training time of each method. Note that we omit the number of model parameters in soft selection layer and inference module, which only occupy an insignificant proportion ($0.5\% \sim 1\%$) of the total model parameters.

5.1.3 Comparison Methods

We compare our AutoSrh with the following seven approaches.

- **UE** (short for Uniform Embedding). The uniform-embedding is commonly accepted in existing embedding-based models, of which all features have a uniform embedding dimension.
- **Grid Search**. This is the traditional approach to searching for a uniform embedding dimension. In our experiments, we search dimensions in $\{4, 8, 12, 16\}$ for KDD12 dataset. For remaining datasets, we search over 16 different dimensions, ranging from 4 to 64 with a stride of 4.
- **Random Search**. Random search has been recognized as a strong baseline for hyperparameter search problems [31]. For fairness, we use the same feature blocking to divide features into different blocks according to their frequencies and assign features in each block with same dimension. We randomly allocate dimensions to feature blocks in each experiment time and report the best performance.
- **MDE** (Mixed Dimension Embedding [24]). This method performs feature blocking and applies a heuristic scheme where the number of dimensions per feature block is proportional to some fractional power of its frequency. We use the same hyperparameter settings as suggested in the original paper.
- **AutoEmb** (Automatically Embedding Search [26]). This is a neural architecture search-based method which enables several embedding dimensions for features according to their popularities. We follow the original paper and select the embedding dimensions $\{2, 8, 16\}$ for KDD12 dataset and $\{8, 32, 64\}$ for the remaining datasets.
- **AutoDim** (AutoML-based framework [27]). This is the state-of-the-art EDS approach which allocates embedding dimensions to different feature fields in an automated and data-driven manner. We select the candidate embedding dimensions $\{2, 4, 8, 12, 16\}$ for KDD12 dataset and $\{8, 16, 32, 48, 64\}$ for the remaining datasets.
- **PEP** (Plug-in Embedding Pruning [30]). This is the state-of-the-art network pruning method which proposes the learnable thresholds to directly prune the unnecessary embedding parameters in one shot. We set the thresholds to be feature-dimension wise and use the same experimental settings in the original paper.

Note that GridSearch assigns uniform embedding dimension for all the input features. We allow GridSearch to evaluate more candidate dimensions to figure out the optimal uniform embedding dimension for all input features. AutoEmb and AutoDim are learning-based embedding dimensionality search approaches which perform hard selection to assign different embedding dimensions to features based on a predefined candidate set, and we follow their original papers to prepare the candidate dimension sets.

5.1.4 Implementation Details

Since all the compared EDS algorithms are model-agnostic, we apply them on several well-known model architectures in tabular domain to compare their performance. The configuration of each model is the same over different methods for a fair comparison. This experimental setting is also adopted in the previous works [26], [47]. Specifically, we adopt MF [48], MLP [17] and NeuMF [17] for rating prediction, and FM [18], Wide&Deep [38], DeepFM [15] for CTR prediction. For AutoSrh, we apply Adam optimizer with the learning rate of 0.001 for model parameters Θ and that of 0.01 for soft selection layer parameters $\boldsymbol{\alpha}$. The mini-batch size n is set to 4096 and the uniform base embedding dimension K is set to 16 for KDD12 dataset and 64 for remaining datasets with all the embedding-based models. We apply the same feature blocking for Random Search, MDE and AutoSrh. The default number of feature blocks L is set to 10 for rating prediction tasks and 6 for CTR prediction task, respectively. Besides, we exploit early-stopping for all the methods according to the change of validation loss during model training. For AutoSrh, we provide the results with different compression rates (CR), i.e., the ratio between the number of embedding parameters and the number of remaining embedding parameters after pruning. All the experiments were conducted on a Linux server equipped with Intel Xeon 2.10GHz CPUs and NVIDIA GeForce RTX 2080Ti GPUs using Pytorch [44]. Our code could be found in the https://github.com/Viperccc/AutoSrh.

5.2 Overall Performance (RQ1)

Performance Comparison on CTR Prediction. We first compare the overall performance of different embedding dimension search methods on CTR prediction tasks. Table 3 shows the results of AUC, Logloss, and the parameter sizes of feature embeddings in three large-scale tabular datasets. We have the following findings.

(1) Unsurprisingly, UE achieves the worst predictive performance with the largest size of model parameters for all the model architectures on both datasets. The results confirm that assigning the uniform embedding dimension is not only memory inefficient, but causes the over-fitting problem for those features that do not require too much

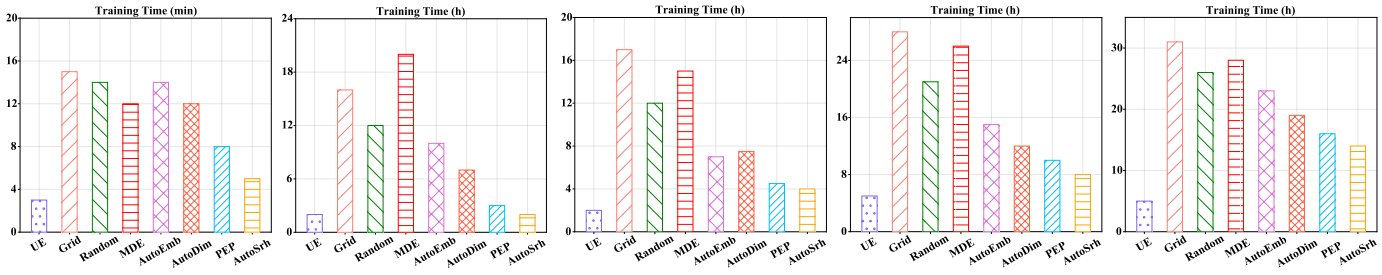
TABLE 3
Comparison between AutoSrh and baselines on the CTR prediction (classification) task. The average AUC, Logloss and the parameter size of the derived feature embeddings per method over 5 repetitions are reported. **Bold** values are the best in each column, while the second best values are underlined. For AutoSrh, we show its results with and w/o different compression rates (CR), i.e., the ratio between embedding parameter size and remaining embedding parameter size after pruning. (M=Million)

Dataset	Model	Metrics	Comparison Methods									
			UE	Grid Search	Random Search	MDE	AutoEmb	AutoDim	PEP	AutoSrh (unpruned)	AutoSrh (CR=20)	AutoSrh (CR=50)
Criteo	FM	AUC	0.7987	0.7987	0.7997	0.7986	0.7991	0.8001	0.7995	**0.8024**	0.8015	0.8015
		Logloss	0.4525	0.4525	0.4518	0.4530	0.4522	0.4513	0.4517	**0.4501**	0.4505	0.4505
		Params(M)	133.56	133.56	63.10	19.79	25.47	16.56	2.74	133.56	6.68	**2.67**
	Wide&Deep	AUC	0.8077	0.8079	0.8084	0.8076	0.8069	0.8089	0.8085	**0.8105**	0.8098	0.8095
		Logloss	0.4437	0.4435	0.4434	0.4439	0.4441	0.4431	0.4432	**0.4409**	0.4412	0.4414
		Params(M)	133.56	100.17	41.40	21.32	30.33	17.06	**2.52**	133.56	6.68	2.67
	DeepFM	AUC	0.8075	0.8080	0.8084	0.8077	0.8068	0.8091	0.8082	**0.8107**	0.8102	0.8102
		Logloss	0.4445	0.4435	0.4434	0.4438	0.4451	0.4424	0.4442	**0.4407**	0.4411	0.4413
		Params(M)	133.56	116.86	41.40	22.27	27.81	14.63	2.81	133.56	6.68	**2.67**
Avazu	FM	AUC	0.7693	0.7693	0.7725	0.7705	0.7679	0.7727	0.7724	**0.7752**	0.7749	0.7742
		Logloss	0.3864	0.3864	0.3845	0.3852	0.3850	0.3834	0.3843	0.3826	**0.3825**	0.3830
		Params(M)	98.84	98.84	24.79	17.54	13.74	14.59	**1.63**	98.84	4.94	1.97
	Wide&Deep	AUC	0.7705	0.7705	0.7739	0.7717	0.7725	0.7746	0.7741	**0.7777**	0.7771	0.7759
		Logloss	0.3855	0.3855	0.3837	0.3844	0.3839	0.3829	0.3831	**0.3815**	0.3817	0.3827
		Params(M)	98.84	98.84	27.81	18.62	15.77	12.71	**1.52**	98.84	4.94	1.97
	DeepFM	AUC	0.7721	0.7721	0.7745	0.7727	0.7739	0.7765	0.7754	**0.7781**	0.7772	0.7757
		Logloss	0.3852	0.3852	0.3842	0.3849	0.3845	0.3831	0.3836	0.3814	**0.3813**	0.3837
		Params(M)	98.84	98.84	30.51	29.49	21.74	15.91	2.17	98.84	4.94	**1.97**
KDD12	FM	AUC	0.7663	0.7667	0.7679	0.7671	0.7652	0.7682	0.7669	**0.7705**	0.7701	0.7687
		Logloss	0.1633	0.1632	0.1626	0.1632	0.1651	0.1621	0.1635	**0.1603**	0.1607	0.1626
		Params(M)	134.87	67.44	62.86	52.11	42.23	32.55	3.27	134.87	6.73	**2.71**
	Wide&Deep	AUC	0.7605	0.7605	0.7644	0.7632	0.7645	0.7647	0.7602	**0.7667**	0.7652	0.7641
		Logloss	0.1651	0.1651	0.1633	0.1639	0.1631	0.1632	0.1652	**0.1608**	0.1619	0.1627
		Params(M)	134.87	134.87	37.25	49.04	44.87	44.36	3.12	134.87	6.73	**2.71**
	DeepFM	AUC	0.7673	0.7673	0.7699	0.7686	0.7695	0.7697	0.7667	**0.7721**	0.7712	0.7699
		Logloss	0.1631	0.1631	0.1613	0.1626	0.1617	0.1623	0.1632	**0.1591**	0.1605	0.1621
		Params(M)	134.87	134.87	53.83	40.65	47.65	34.77	2.95	134.87	6.73	**2.71**

representation capacity, resulting in the poor predictive performance. While the candidate dimension sets used by AutoEmb and AutoDim is smaller than that used by GridSearch, both AutoEmb and AutoDim consistently perform better than GridSearch and have smaller parameter sizes. This result also demonstrates the significance of using mixed embedding dimension scheme towards better model performance and smaller parameter size. (2) AutoSrh achieves the best predictive performance over all the model architectures and datasets. The results confirm that searching embedding dimensions for different features in a flexible search space via gradient descent is more effective than selecting dimensions from a predefined candidate set. Furthermore, we use the soft selection layer to directly control the significance of each dimension in feature embedding. By optimizing the soft selection layer with the validation loss, we could assign lower/higher weights to the embedding dimensions in less/more predictive features. This penalizes the model for overfitting to less predictive features and encourages the model to learn more predictive features, leading to better model performance. Besides, by assigning lower weights to the embedding dimensions in less predictive features, AutoSrh can prune unimportant dimensions to reduce the parameter size without a significant drop in model performance. (3) AutoSrh consistently performs better than the state-of-the-art EDS approach AutoDim in AUC and Logloss, which validates that assigning different embedding sizes to individual feature values rather than the feature fields (embeddings in the same feature field share the same dimension) could achieve better predictive performance. The reason is that, in CTR task, the frequency of features in the same feature field may also vary greatly and they need specific embedding dimensions. Besides, AutoSrh outperforms pruning method PEP by a large margin. This is because PEP simultaneously optimizes the learnable threshold and model parameters with the same training batches, which may cause overfitting and decrease the generalization performance. (4) AutoSrh can yield a significant reduction in the embedding parameter size while maintaining competitive predictive performance. For example, AutoSrh with the CR of 20 (reducing 95% parameters in feature embeddings) consistently outperforms all the baselines in AUC and Logloss in all the cases and even AutoSrh with the CR of 50 (reducing 98% parameters in feature embeddings) could achieve the better performance in AUC and Logloss in 13 of 18 cases w.r.t different model and datasets. (5) Only PEP and AutoSrh with the CR of 50 could greatly reduce the number of parameters, which validates the effectiveness of embedding pruning. Besides, our proposed AutoSrh with the CR of 50 could consistently achieve better predictive performance than PEP. (6) Compared with the results on Criteo and Avazu, we observe that the predictive performance of AutoSrh on KDD12 dataset would decrease more rapidly as the CR increases. This is because the number of input features in the KDD12 dataset is much larger than those datasets. Pruning too many parameters would limit the

TABLE 4
Comparison between AutoSrh and baselines on the rating prediction (regression) task.

Dataset	Model	Metrics	Comparison Methods									
			UE	Grid Search	Random Search	MDE	AutoEmb	AutoDim	PEP	AutoSrh (unpruned)	AutoSrh (CR=2)	AutoSrh (CR=2.5)
MovieLens	MF	MSE	0.6224	0.6224	0.6153	0.6138	0.6144	0.6222	0.6254	**0.6089**	0.6126	0.6167
		Params(M)	10.61	10.61	9.78	10.14	8.78	10.61	**2.17**	10.61	5.30	4.24
	MLP	MSE	0.6400	0.6400	0.6361	0.6312	0.6345	0.6291	0.6445	**0.6255**	0.6303	0.6361
		Params(M)	10.61	10.61	6.67	10.25	9.22	9.84	**1.59**	10.61	5.30	4.24
	NeuMF	MSE	0.6251	0.6251	0.6238	0.6249	0.6226	0.6257	0.6288	**0.6146**	0.6169	0.6231
		Params(M)	10.61	10.61	5.31	9.77	8.64	10.05	**2.04**	10.61	5.30	4.24
Amazon	MF	MSE	1.1053	1.1053	1.1016	1.0976	1.0971	0.1031	1.1095	**1.0916**	1.0964	1.1077
		Params(M)	12.03	12.03	9.97	10.25	8.59	10.26	**4.37**	12.03	6.01	**4.82**
	MLP	MSE	1.1105	1.1105	1.1112	1.0986	1.0995	1.1112	1.1145	**1.0975**	1.1143	1.1187
		Params(M)	12.03	12.03	10.81	10.26	9.47	11.26	**4.52**	12.03	6.01	4.82
	NeuMF	MSE	1.1024	1.1024	1.1093	1.0945	1.0963	1.1027	1.1052	**1.0871**	1.0946	1.1075
		Params(M)	12.03	12.03	10.76	9.49	9.76	10.65	5.17	12.03	6.01	**4.82**



(a) Amazon (b) Movieslens (c) Criteo (d) Avazu (e) KDD12

Fig. 4. Training efficiency results. "GD" and "RD" are Grid Search and Random Search for short, respectively.

model capability to effectively encode all the input features, resulting in inferior performance.

Performance Comparison on Rating Prediction. We also evaluate the performance of AutoSrh and the baseline methods on rating prediction task using Movielens-20M and Amazon datasets. As shown in Table 4, AutoSrh achieves the best predictive performance over all the model architectures. The results confirm that AutoSrh is able to learn discriminative feature embeddings with significantly higher effectiveness than the existing search methods. Besides, we observe that the compression rates (CR) on rating prediction task cannot reach the CR of CTR task without sacrificing predictive performance. This is because the distribution of feature frequency on these two datasets is relatively even, leading to a lower number of redundant dimensions for low-frequency features. This also explains why PEP fails in this situation.

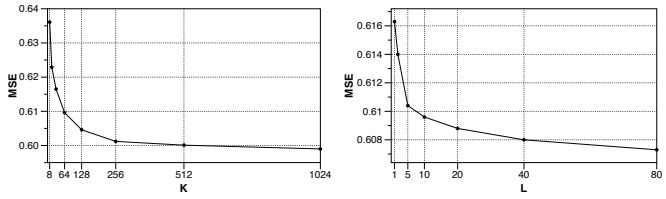
To sum up, compared with the baselines, AutoSrh reports the best performance in CTR prediction task,and it with 95∼98% embedding parameters pruned achieves comparable performance to other EDS approaches. AutoSrh with 50∼60% embedding parameter pruned also achieves lower MSE in rating prediction task. In a nutshell, these results prove the effectiveness of the AutoSrh framework.

5.3 Training Efficiency Analysis (RQ2)

In addition to predictive performance, training efficiency is also an essential metric for model deployment in real-world applications. We present the training time results of different searching algorithms on five datasets in Fig. 4. We use MF model for rating prediction and FM model for CTR prediction, while the results of using other deep learning models have similar trends. We can observe that: (1) Unsurprisingly, UE has the lower training time since the embedding dimension is set to a constant and there is no time cost of dimension search. However, its predictive performance and parameters size are the worst among all the baselines. (2) In addition to UE, AutoSrh and PEP have the lower training time cost than other baselines, which verifies the training efficiency of using the trainable parameters to search the mixed dimension scheme. Besides, AutoSrh could still achieve $1.1 \sim 1.6\times$ lower time cost of model training compared with PEP. This is because AutoSrh has a smaller search space than PEP by using the feature blocking. As a result, our proposed AutoSrh is more efficient than the baselines in terms of training time cost, which makes it easier to be deployed in real-world systems.

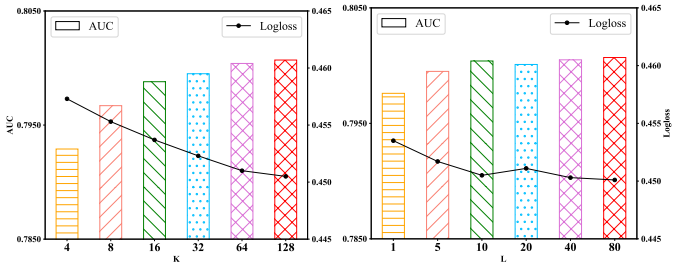
5.4 Hyperparameter Investigation (RQ3)

In order to investigate the effect of different choices of base embedding dimensions K and feature block numbers L on the model performance of AutoSrh, we evaluate these hyperparameters on Movielens and Criteo datasets and provide the sensitivity analysis, while the results on the other datasets have similar trends. Fig. 5(a) and Fig. 6(a) shows the performance change in Movielens and Criteo datasets with respect to the different settings of K and fixed $L = 10$. We can observe the similar pattern that increasing



(a) MSE *vs* K. (b) MSE *vs* L.

Fig. 5. Effect of hyperparameters on the rating prediction performance of AutoSrh (MF, Movielens-20M dataset).



(a) AUC, Logloss *vs* K. (b) AUC, Logloss *vs* L.

Fig. 6. Effect of hyperparameters on the CTR prediction performance of AutoSrh (FM, Criteo dataset).

K is beneficial to improving predictive performance. This is because a larger K allows a larger search space that could improve the representations of high-frequency features by giving more embedding dimensions. Besides, we observe a marginal decrease in performance gain. Specifically, the MSE reduction in rate prediction task is merely 0.001 when K changes from 512 to 1024 and the improvement of AUC in CTR prediction is only 0.0003 when K increases from 64 to 128. This implies that K may have exceeded the largest number of dimensions required by all the features, leading to minor improvements. Figure 5(b) and Figure 6(b) shows the effects of the number of feature blocks L w.r.t $K = 64$. We find that increasing L improves the prediction performance of AutoSrh, and the performance improvement decreases as L becomes larger. This is because dividing features into more blocks facilitates a more finer-grained control on the embedding dimensions of different features, leading to more flexible mixed dimension schemes. Since both K and L affect the computation complexity of AutoSrh, we suggest to choose reasonably large values for K and L to balance the computational efficiency and predictive performance based on the application requirements.

5.5 Effect of Model Retraining (RQ4)

We now investigate how the model retraining further improves the model performance. We report the AUC results on CTR prediction task with Wide&Deep model and MSE results on rating prediction task with NeuMF model in Table 5, while the results with other prediction models have similar trends. AutoSrh+FE (IF) fixes the model parameters in the feature embedding layer (inference module) during the model retraining and AutoSrh\R represents AutoSrh without model retraining. We can observe that: (1) AutoSrh consistently outperforms AutoSrh\R by a large margin. This is in line with our analysis in Section 4.4 that

TABLE 5
Effectiveness of model retraining in AutoSrh.

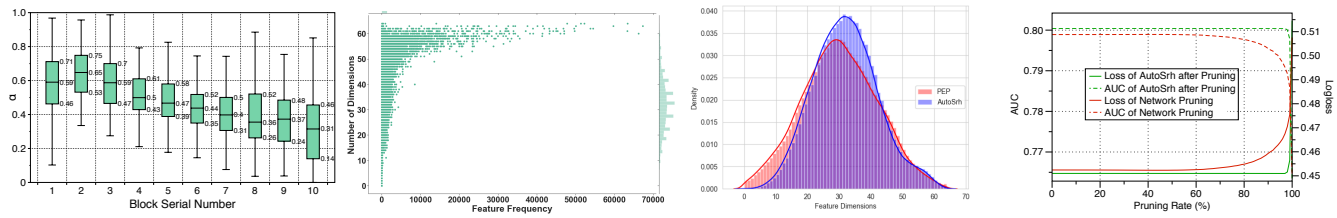
Method	CTR Prediction (AUC↑)			Rating Prediction (MSE↓)	
	Criteo	Avazu	KDD12	Movielens	Amazon
AutoSrh\R	0.8085	0.7726	0.7652	0.6096	1.0924
AutoSrh+FE	0.8089	0.7756	0.7652	0.6092	1.0919
AutoSrh+IF	0.8095	0.7764	0.7657	0.6095	1.0918
AutoSrh	0.8105	0.7776	0.7667	0.6089	1.0916

suboptimal embedding dimensions in the dimensionality search stage would interfere with the model performance and the retraining stage is useful to improve the predictive performance. (2) AutoSrh consistently performs better than AutoSrh+IF and AutoSrh+FE. This is because model parameters in both the feature embedding layer and inference layer are highly dependent on the soft selection layer and affected by suboptimal embedding dimensions. Therefore, it is crucial to fine-tune all the model parameters with learned soft selection layer.

5.6 Analysis on AutoSrh Results (RQ5)

We first study the learned feature dimensions of AutoSrh through the learned soft selection layer $\boldsymbol{\alpha}$ and feature embedding dimensions after dimension pruning. Fig. 7(a) depicts the distributions of the trained parameters in $\boldsymbol{\alpha}$ for the 10 feature blocks on Movielens. Recall that the blocks are sorted in the descending order of feature frequency. We can see that the learned parameters in $\boldsymbol{\alpha}$ for the feature blocks with lower frequencies converge to smaller values, indicating that lower-frequency features tend to be represented by smaller numbers of embedding dimensions. Fig. 7(b) provides the correlation between embedding dimensions and feature frequency after dimension pruning. The results show that features with high frequencies end up with high embedding dimensions, whereas the dimensions are more likely to be pruned for low-frequency features. Nevertheless, there is no linear correlationship between the derived embedding dimension and the feature frequency. Note that the embedding dimensions for low-frequency features scatter over a long range of numbers. This is consistent with the inferior performance of MDE which directly determines the embedding dimensions of features according to their frequency. We have also shown the distribution of feature embedding dimensions obtained by AutoSrh and PEP in Fig. 7(c). The result demonstrates that both PEP and AutoSrh adaptively assign different embedding dimensions to different input features. Compared with PEP, our approach has fewer features with lower dimensions and more features with higher dimensions, which indicates it can adaptively find more predictive features.

We further compare AutoSrh with network pruning method [49]. For illustration purpose, we provide the results of the FM model on Criteo dataset. Fig. 7(d) shows the performance of two methods on different pruning rates (i.e., the ratio of pruned embedding weights). From the result, AutoSrh achieves better AUC and Logloss results than network pruning over all the pruning rates. This is because AutoSrh optimizes feature embeddings with the



(a) $\boldsymbol{\alpha}$ in different feature blocks. (b) Embedding dimension vs η_f. (c) Dimension distribution. (d) Performance over different pruning rates

Fig. 7. (a) The distribution of trained parameters of the soft selection layer α. Here we show the results of MF on Movielens, where L is set to 10. (b) The joint distribution of feature embedding dimensions and feature frequencies after dimension pruning. (c) The distribution of embedding dimensions of AutoSrh and PEP on MovieLens with 50% embedding parameters pruned. (d) Comparison of AutoSrh and network pruning performance over different pruning rates.

gradients from the validation set, which benefits the selection of predictive dimensions, instead of simply removing redundant weights in the embeddings.

6 Related Work

Automatically searching for mixed feature embedding dimensions has been an emerging research problem in recent years. Existing approaches can be generally categorized into two groups: *reinforcement learning-based* and *AutoML-based* methods. The reinforcement learning-based approaches [25], [29] first define a set of candidate dimensions, and then apply reinforcement learning methods to learn to select candidate dimensions for different features. Joglekar et al. [25] proposed NIS, which divided feature embeddings into multiple blocks according to the feature frequency and used a reinforcement learning-based algorithm to search dimension for each block. Liu et al. [29] proposed a novel embedding size adjustment policy network, which applied hard selection on different embedding sizes to adaptively search appropriate embedding dimensions for different features. The existing methods, however, restrict each feature embedding dimension to be chosen from a small set of candidate dimensions that is explicitly predefined. This inevitably leads to another question of how to decide the candidate dimensions, since a suboptimal candidate set would result in deficient search results [30].

The second category includes AutoML-based methods. Some recent approaches treated the embedding dimensionality search problem as one kind of Neural Architecture Search (NAS) [50] and assigned adaptive embedding dimensions to different input features in an automated manner. Zhao et al. [26] proposed AutoEmb that trains a controller to assign weights to various candidate embedding sizes according to the feature frequency and automatically assigns different embedding dimensions for different features by hard selection. However, the performance of AutoEmb is dependent on the predefined set of candidate dimensions. Zhao et al. [27] proposed AutoDim which utilizes an AutoML-based framework to learn the mixed embedding dimensions for feature fields. However, they ignore the fact that the feature frequency in the same feature field may be heavily skewed and hence assigning the same embedding dimension to all the features in a feature field would limit model's predictive performance. Besides, it is of limited utility when tabular data only involves a small number of feature fields. Note that we do not compare with traditional NAS approaches that achieve remarkable performance in various tasks such as image classification [51], semantic segmentation [52] and object detection [53]. The reason is that most of the researches in the area of NAS focus on searching for optimal network structures automatically and ignore the design of the input component since the input features in visual tasks are already given in the form of floating-point values, e.g., image pixels. In this paper, we focus on the design of the embedding layer in deep models on tabular data, which is developed to transform raw features (e.g., discrete user identifiers) into dense embeddings. Hence, our work can be considered as a kind of NAS on the input component (i.e., the embedding layer) and the proposed AutoSrh is able to automatically search embedding dimensions for heterogenous features in tabular data in a differentiable manner.

Our work is also related to network pruning, which improves the efficiency of over-parameterized deep neural networks by removing redundant neurons or connections without damaging model performance [54], [55], [56]. Some researches [55], [56] indicated that network pruning can also be considered as performing implicit architecture search. Liu et al. [30] proposed a Plug-in Embedding Pruning (PEP) method to reduce the size of the embedding matrix using the trainable threshold while alleviating performance degradation. Distinct from PEP, our AutoSrh aims to find the optimal mixed dimension scheme to minimize validation loss, which is similar to the task of hyperparameters optimization.

Our work is different from dimensionality reduction approaches [57] which transform the whole embedding table from high-dimensional space into low-dimensional space by retaining useful information of the original embeddings. While dimensionality reduction approaches could also reduce the size of embedding table, they focus on assigning all the input features with the same lower embedding dimension to achieve memory-performance tradeoff, whereas we allow features to have different embedding dimensions towards better predictive performance.

7 Conclusion

Existing deep learning-based models, while becoming the state-of-the-art driving force of tabular data prediction, have been recognized to suffer from limited effectiveness and

high memory cost when applying the uniform embedding dimensions for all input features. To fill this gap, we propose a novel end-to-end embedding dimensionality search framework named AutoSrh, which searches for mixed features embedding dimensions in a differentiable manner through gradient descent. The key idea of AutoSrh is the adaptive dimensionality search process which introduces a soft selection layer that controls the significance of each embedding dimension and optimizes the parameters according to model's validation performance. AutoSrh further employs a fine-grained pruning procedure to produce a flexible mixed embedding dimension scheme for different features and performs model retraining to improve the predictive performance. AutoSrh is model-agnostic, which can be applied to various architectures of deep learning-based models for tabular data prediction. We conduct experiments on two typical applications of tabular data prediction with five real-world tabular datasets. The results show that AutoSrh can achieve better predictive performance than the existing embedding dimensionality search methods, with fewer embedding parameters and lower training time cost.

ACKNOWLEDGMENTS

The authors would like to thank the anonymous reviewers for their insightful reviews. This work is supported by the National Key Research and Development Program of China (2022YFE0200500), Shanghai Municipal Science and Technology Major Project (2021SHZDZX0102), the Tencent Wechat Rhino-Bird Focused Research Program, and SJTU Global Strategic Partnership Fund (2021 SJTU-HKUST).

REFERENCES

[1] M. A. Khamis, H. Q. Ngo, X. Nguyen, D. Olteanu, and M. Schleich, "Learning models over relational data using sparse tensors and functional dependencies," *ACM Trans. Database Systems*, vol. 45, no. 2, pp. 7:1–7:66, 2020.

[2] J. Bobadilla, F. Ortega, A. Hernando, and A. Gutiérrez, "Recommender systems survey," *Knowledge-based systems*, vol. 46, pp. 109–132, 2013.

[3] J. Zhang, C. Gao, D. Jin, and Y. Li, "Group-buying recommendation for social e-commerce," in *ICDE*, 2021, pp. 1536–1547.

[4] S. Cai, K. Zheng, G. Chen, H. V. Jagadish, B. C. Ooi, and M. Zhang, "Arm-net: Adaptive relation modeling network for structured data," in *SIGMOD*, 2021, pp. 207–220.

[5] J. Zhao, P. Zhao, L. Zhao, Y. Liu, V. S. Sheng, and X. Zhou, "Variational self-attention network for sequential recommendation," in *ICDE*, 2021, pp. 1559–1570.

[6] I. Kononenko, "Machine learning for medical diagnosis: history, state of the art and perspective," *Artificial Intelligence in medicine*, vol. 23, no. 1, pp. 89–109, 2001.

[7] R. J. Bolton and D. J. Hand, "Statistical fraud detection: A review," *Statistical science*, vol. 17, no. 3, pp. 235–255, 2002.

[8] D. Amodei, S. Ananthanarayanan, R. Anubhai, J. Bai, E. Battenberg, C. Case, J. Casper, B. Catanzaro, Q. Cheng, G. Chen *et al.*, "Deep speech 2: End-to-end speech recognition in english and mandarin," in *ICML*, 2016, pp. 173–182.

[9] S. Lai, L. Xu, K. Liu, and J. Zhao, "Recurrent convolutional neural networks for text classification," in *Twenty-ninth AAAI conference on artificial intelligence*, 2015.

[10] K. He, X. Zhang, S. Ren, and J. Sun, "Deep residual learning for image recognition," in *CVPR*, 2016, pp. 770–778.

[11] J. Luo, S. Xiao, S. Jiang, H. Gao, and Y. Xiao, "ripple2vec: Node embedding with ripple distance of structures," *Data Sci. Eng.*, vol. 7, no. 2, pp. 156–174, 2022.

[12] B. Hosseini, R. Montagné, and B. Hammer, "Deep-aligned convolutional neural network for skeleton-based action recognition and segmentation," *Data Sci. Eng.*, vol. 5, no. 2, pp. 126–139, 2020.

[13] M. Mahdavi and Z. Abedjan, "Baran: Effective error correction via a unified context representation and transfer learning," *Proceedings of the VLDB Endowment*, vol. 13, no. 12, pp. 1948–1961, 2020.

[14] S. Rendle, "Scaling factorization machines to relational data," *Proc. VLDB Endow.*, vol. 6, no. 5, pp. 337–348, 2013.

[15] H. Guo, R. Tang, Y. Ye, Z. Li, and X. He, "Deepfm: A factorization-machine based neural network for CTR prediction," in *IJCAI*, 2017, pp. 1725–1731.

[16] S. Li, L. Chen, and A. Kumar, "Enabling and optimizing non-linear feature interactions in factorized linear algebra," in *SIGMOD*, 2019, pp. 1571–1588.

[17] X. He, L. Liao, H. Zhang, L. Nie, X. Hu, and T.-S. Chua, "Neural collaborative filtering," in *Proceedings of the 26th international conference on world wide web*, 2017, pp. 173–182.

[18] S. Rendle, "Factorization machines," in *IEEE International conference on data mining*, 2010, pp. 995–1000.

[19] H. Guo, B. Chen, R. Tang, W. Zhang, Z. Li, and X. He, "An embedding learning framework for numerical features in ctr prediction," in *KDD*, 2021, pp. 2910–2918.

[20] J. Park, M. Naumov, P. Basu, S. Deng, A. Kalaiah, D. Khudia, J. Law, P. Malani, A. Malevich, S. Nadathur *et al.*, "Deep learning inference in facebook data centers: Characterization, performance optimizations and hardware implications," *arXiv:1811.09886*, 2018.

[21] W. Cheng, Y. Shen, and L. Huang, "Adaptive factorization network: Learning adaptive-order feature interactions," in *Proceedings of the AAAI Conference on Artificial Intelligence*, vol. 34, no. 04, 2020, pp. 3609–3616.

[22] X. Zhou, Y. Shen, L. Huang, T. Zang, and Y. Zhu, "Multi-level attention networks for multi-step citywide passenger demands prediction," *IEEE Transactions on Knowledge and Data Engineering*, vol. 33, no. 5, pp. 2096–2108, 2021.

[23] F. M. Harper and J. A. Konstan, "The movielens datasets: History and context," *ACM Trans. Interact. Intell. Syst.*, vol. 5, no. 4, 2015.

[24] A. Ginart, M. Naumov, D. Mudigere, J. Yang, and J. Zou, "Mixed dimension embeddings with application to memory-efficient recommendation systems," *arXiv:1909.11810*, 2019.

[25] M. R. Joglekar, C. Li, M. Chen, T. Xu, X. Wang, J. K. Adams, P. Khaitan, J. Liu, and Q. V. Le, "Neural input search for large scale recommendation models," in *KDD*, 2020, pp. 2387–2397.

[26] X. Zhao, C. Wang, M. Chen, X. Zheng, X. Liu, and J. Tang, "Autoemb: Automated embedding dimensionality search in streaming recommendations," *arXiv:2002.11252*, 2020.

[27] X. Zhao, H. Liu, H. Liu, J. Tang, W. Guo, J. Shi, S. Wang, H. Gao, and B. Long, "Autodim: Field-aware embedding dimension searchin recommender systems," in *Proceedings of the Web Conference*, 2021, pp. 3015–3022.

[28] P. Covington, J. Adams, and E. Sargin, "Deep neural networks for youtube recommendations," in *Proceedings of the 10th ACM conference on recommender systems*, 2016, pp. 191–198.

[29] H. Liu, X. Zhao, C. Wang, X. Liu, and J. Tang, "Automated embedding size search in deep recommender systems," in *SIGIR*, 2020, pp. 2307–2316.

[30] S. Liu, C. Gao, Y. Chen, D. Jin, and Y. Li, "Learnable embedding sizes for recommender systems," in *ICLR*, 2021.

[31] H. Liu, K. Simonyan, and Y. Yang, "DARTS: differentiable architecture search," in *ICLR*, 2019.

[32] X. Guo, Y. Quan, H. Zhao, Q. Yao, Y. Li, and W. Tu, "Tabgnn: Multiplex graph neural network for tabular data prediction," *arXiv:2108.09127*, 2021.

[33] J. Qin, W. Zhang, R. Su, Z. Liu, W. Liu, R. Tang, X. He, and Y. Yu, "Retrieval & interaction machine for tabular data prediction," in *KDD*, 2021, pp. 1379–1389.

[34] R. Bordawekar and O. Shmueli, "Using word embedding to enable semantic queries in relational databases," in *Proceedings of the 1st Workshop on Data Management for End-to-End Machine Learning, DEEM@SIGMOD*, 2017, pp. 5:1–5:4.

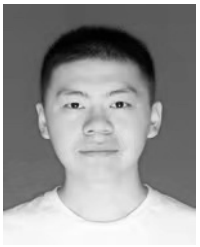
[35] W. Wang, M. Zhang, G. Chen, H. V. Jagadish, B. C. Ooi, and K. Tan, "Database meets deep learning: Challenges and opportunities," *CoRR*, vol. abs/1906.08986, 2019.

[36] W. Cheng, Y. Shen, Y. Zhu, and L. Huang, "DELF: A dual-embedding based deep latent factor model for recommendation," in *IJCAI*, 2018, pp. 3329–3335.

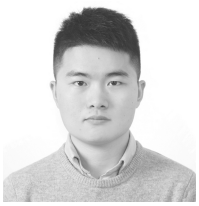
[37] J. Lian, X. Zhou, F. Zhang, Z. Chen, X. Xie, and G. Sun, "xdeepfm: Combining explicit and implicit feature interactions for recommender systems," in *KDD*, 2018, pp. 1754–1763.

[38] H. Cheng, L. Koc, J. Harmsen, T. Shaked, T. Chandra, H. Aradhye, G. Anderson, G. Corrado, W. Chai, M. Ispir, R. Anil, Z. Haque,

L. Hong, V. Jain, X. Liu, and H. Shah, "Wide & deep learning for recommender systems," in *Proceedings of the 1st Workshop on Deep Learning for Recommender Systems*, 2016, pp. 7–10.
[39] Y. Chen, B. Chen, X. He, C. Gao, Y. Li, J. Lou, and Y. Wang, "λopt: Learn to regularize recommender models in finer levels," in *KDD*, 2019, pp. 978–986.
[40] L. Franceschi, P. Frasconi, S. Salzo, R. Grazzi, and M. Pontil, "Bilevel programming for hyperparameter optimization and meta-learning," in *ICML*, 2018, pp. 1563–1572.
[41] J. Bergstra, R. Bardenet, Y. Bengio, and B. Kégl, "Algorithms for hyper-parameter optimization," *NeurIPS*, vol. 24, 2011.
[42] B. Colson, P. Marcotte, and G. Savard, "An overview of bilevel optimization," *Annals OR*, vol. 153, no. 1, pp. 235–256, 2007.
[43] F. Pedregosa, "Hyperparameter optimization with approximate gradient," in *ICML*, 2016, pp. 737–746.
[44] A. Paszke, S. Gross, F. Massa, A. Lerer, J. Bradbury, G. Chanan, T. Killeen, Z. Lin, N. Gimelshein, L. Antiga, A. Desmaison, A. Köpf, E. Yang, Z. DeVito, M. Raison, A. Tejani, S. Chilamkurthy, B. Steiner, L. Fang, J. Bai, and S. Chintala, "Pytorch: An imperative style, high-performance deep learning library," in *NeurIPS*, 2019, pp. 8024–8035.
[45] M. Abadi, P. Barham, J. Chen, Z. Chen, A. Davis, J. Dean, M. Devin, S. Ghemawat, G. Irving, M. Isard *et al.*, "Tensorflow: A system for large-scale machine learning," in *OSDI*, 2016, pp. 265–283.
[46] P. Virtanen, R. Gommers, T. E. Oliphant, M. Haberland, T. Reddy, D. Cournapeau, E. Burovski, P. Peterson, W. Weckesser, J. Bright *et al.*, "Scipy 1.0: fundamental algorithms for scientific computing in python," *Nature methods*, vol. 17, no. 3, pp. 261–272, 2020.
[47] T. Liu, J. Fan, Y. Luo, N. Tang, G. Li, and X. Du, "Adaptive data augmentation for supervised learning over missing data," *Proceedings of the VLDB Endowment*, vol. 14, no. 7, pp. 1202–1214, 2021.
[48] Y. Koren, R. Bell, and C. Volinsky, "Matrix factorization techniques for recommender systems," *Computer*, vol. 42, no. 8, pp. 30–37, 2009.
[49] S. Han, J. Pool, J. Tran, and W. Dally, "Learning both weights and connections for efficient neural network," in *NeurIPS*, 2015, pp. 1135–1143.
[50] T. Elsken, J. H. Metzen, and F. Hutter, "Neural architecture search: A survey," *The Journal of Machine Learning Research*, vol. 20, no. 1, pp. 1997–2017, 2019.
[51] E. Real, A. Aggarwal, Y. Huang, and Q. V. Le, "Regularized evolution for image classifier architecture search," in *Proceedings of the aaai conference on artificial intelligence*, vol. 33, no. 01, 2019, pp. 4780–4789.
[52] L.-C. Chen, M. Collins, Y. Zhu, G. Papandreou, B. Zoph, F. Schroff, H. Adam, and J. Shlens, "Searching for efficient multi-scale architectures for dense image prediction," *Advances in neural information processing systems*, vol. 31, 2018.
[53] B. Zoph, V. Vasudevan, J. Shlens, and Q. V. Le, "Learning transferable architectures for scalable image recognition," in *Proceedings of the IEEE conference on computer vision and pattern recognition*, 2018, pp. 8697–8710.
[54] Y. Cheng, D. Wang, P. Zhou, and T. Zhang, "A survey of model compression and acceleration for deep neural networks," *arXiv:1710.09282*, 2017.
[55] Z. Liu, M. Sun, T. Zhou, G. Huang, and T. Darrell, "Rethinking the value of network pruning," in *ICLR*, 2019.
[56] J. Frankle and M. Carbin, "The lottery ticket hypothesis: Finding sparse, trainable neural networks," in *ICLR*, 2019.
[57] Y. Zeng, Y. Tong, and L. Chen, "Hst+: An efficient index for embedding arbitrary metric spaces," in *2021 IEEE 37th International Conference on Data Engineering (ICDE)*, 2021, pp. 648–659.



Shuming Kong received the BS degree from Xi'an Jiao Tong University, Xi'an, China, in 2019. He is currently working toward the PhD degree with the Department of Computer Science and Engineering, Shanghai Jiao Tong University, Shanghai, China. His research interests include machine learning, representation learning and data mining.



Weiyu Cheng received the BE degree and the PhD degree in computer science from the Shanghai Jiao Tong University, Shanghai, China in 2016 and 2021, respectively. He is currently a machine learning researcher at Meituan. His research interests include deep learning and recommender systems.

Yanyan Shen received the BSc degree from Peking University, Haidian, China, in 2010, and the PhD degree in computer science from the National University of Singapore, Kent Ridge, Singapore in 2015. She is currently an associate professor at Shanghai Jiao Tong University, Shanghai, China. Her research interests include machine learning, large-scale data management, and data integration. She is a member of the IEEE.

Linpeng Huang received the MS and PhD degrees in computer science from Shanghai Jiao Tong University, Shanghai, China in 1989 and 1992, respectively. He is a professor of computer science with the Department of Computer Science and Engineering, Shanghai Jiao Tong University. His research interests include the area of distributed systems and service oriented computing. He is a member of the IEEE.