iHAS: Instance-wise Hierarchical Architecture Search for Deep Learning Recommendation Models

Yakun Yu
yakun2@ualberta.ca
University of Alberta
Edmonton, Alberta, Canada

Shi-ang Qi
shiang@ualberta.ca
University of Alberta
Edmonton, Alberta, Canada

Jiuding Yang
jiuding@ualberta.ca
University of Alberta
Edmonton, Alberta, Canada

Liyao Jiang
liyao1@ualberta.ca
University of Alberta
Edmonton, Alberta, Canada

Di Niu
dniu@ualberta.ca
University of Alberta
Edmonton, Alberta, Canada

ABSTRACT

Current recommender systems employ large-sized embedding tables with uniform dimensions for all features, leading to overfitting, high computational cost, and suboptimal generalizing performance. Many techniques aim to solve this issue by feature selection or embedding dimension search. However, these techniques typically select a fixed subset of features or embedding dimensions for all instances and feed all instances into one recommender model without considering heterogeneity between items or users. This paper proposes a novel instance-wise Hierarchical Architecture Search framework, iHAS, which automates neural architecture search at the instance level. Specifically, iHAS incorporates three stages: searching, clustering, and retraining. The searching stage identifies optimal instance-wise embedding dimensions across different field features via carefully designed Bernoulli gates with stochastic selection and regularizers. After obtaining these dimensions, the clustering stage divides samples into distinct groups via a deterministic selection approach of Bernoulli gates. The retraining stage then constructs different recommender models, each one designed with optimal dimensions for the corresponding group. We conduct extensive experiments to evaluate the proposed iHAS on two public benchmark datasets from a real-world recommender system. The experimental results demonstrate the effectiveness of iHAS and its outstanding transferability to widely-used deep recommendation models.

CCS CONCEPTS

• **Information systems** → **Recommender systems**; *Online advertising*.

KEYWORDS

recommender system, instance-wise, embedding dimension search

CIKM '23, October 21–25, 2023, Birmingham, United Kingdom

ACM ISBN 979-8-4007-0124-5/23/10…$15.00
https://doi.org/10.1145/3583780.3614925

ACM Reference Format:
Yakun Yu, Shi-ang Qi, Jiuding Yang, Liyao Jiang, and Di Niu. 2023. iHAS: Instance-wise Hierarchical Architecture Search for Deep Learning Recommendation Models. In *Proceedings of the 32nd ACM International Conference on Information and Knowledge Management (CIKM '23), October 21–25, 2023, Birmingham, United Kingdom.* ACM, New York, NY, USA, 10 pages. https://doi.org/10.1145/3583780.3614925

1 INTRODUCTION

Recommender systems, which aim to predict the preference of users, have been widely deployed in various real-world scenarios, e.g., online advertising [3, 27], social media [9], news apps [39], etc. Deep learning recommendation models (DLRMs) typically take a large amount of categorical (e.g., gender) or numerical (e.g., age) field features as input. These features are first encoded into high-dimensional sparse one-hot vectors, which are later mapped into real-valued dense vectors via embedding tables. The recommender model then feeds these embeddings into a feature interaction layer which usually consists of deep neural network (DNN) [6] or factorization machine (FM) [8, 14, 26, 34] to model user preferences for final prediction.

The embedding tables play a fundamental role in the recommendation system, as they dominate the majority of parameters. However, most existing methods construct their proposed recommender models with large-sized embedding tables and a uniform dimension size for all possible fields [8, 10, 29], which may lead to overfitting, high computational cost, and poor model generalization [15, 25, 33]. Therefore, the first objective for an optimal DLRM is to find optimal embedding dimensions for different fields and remove redundant dimensions. Performing embedding dimension search is also sufficient to include feature selection, i.e., the optimal dimension for a field feature could be zero, which means completely excluding this feature.

A common approach for embedding dimension search is to employ the ℓ_0 norm on the dimensions to penalize the count of non-zero dimension entries. However, as the ℓ_0 norm poses a computational challenge for gradient descent, researchers have attempted to substitute ℓ_0 with a surrogate function, such as the ℓ_1 norm for LASSO [32], yet achieving limited selection ability [35]. Recently, probabilistic approaches [18, 33] suggest utilizing Bernoulli random variables (RVs) with Gumbel-Softmax approximations to identify the top K features with the highest probabilities. Though these

methods can be applied for dimension selection, we have empirically observed that the probabilities learned are often indistinguishable. Therefore, selecting the top K features/dimensions based on these probabilities may inadvertently result in either the exclusion of critical features/dimensions or the inclusion of irrelevant ones.

Furthermore, prior approaches uniformly apply embedding dimension selection across all instances in the datasets, therefore disregarding the inherent variations between individual samples. This one-size-for-all approach can be inadequate in many scenarios, especially when dealing with highly heterogeneous populations where relevant features can significantly diverge across users or across items. For example, in a movie recommendation system, the feature "age" usually plays a crucial role in recommending Disney movies, thereby possibly necessitating a larger embedding dimension. Conversely, "age" is less relevant for comedy films, resulting in a smaller dimension size. Thus, it is evident that treating all instances identically may overlook these context-specific nuances. Intuitively speaking, when dimension selection is performed at the instance level, we can create neural architectures that are better suited to individual samples. Such an approach not only results in superior performance but also enables faster inference times by focusing on the most relevant dimensions of each sample.

In this paper, we propose an instance-wise Hierarchical Architecture Search framework, iHAS, which attempts to perform automatic architecture search on the instance level, using hierarchical training procedures for DLRMs. Specifically, iHAS includes three learning stages: searching, clustering, and retraining. The searching stage aims to find the optimal instance-wise embedding dimensions across different fields via a carefully designed "Bernoulli gate" with stochastic selection mode and a regularizer. After selecting instance-wise embedding dimensions, we separate samples into different groups based on a novel deterministic selection approach in the clustering stage. The retraining stage trains different recommender models, with optimal dimensions tailored to different groups. During inference time, each test sample will first be assigned to a suitable group, where predictions are made by the corresponding recommender model. We summarize our major contributions as:

- We propose a hierarchical training framework that uses instance-wise "Bernoulli gates" to facilitate effective dimension search for each sample.
- We apply a sparse and bi-polarization regularizer in the objective function to help the model learn distinguishable Bernoulli RVs, and use a threshold selector for downstream deterministic selection.
- To balance the trade-off between the one-size-for-all and full-customization (which is not applicable with finite data size) strategies, we propose to divide samples into clusters and develop tailored recommender models for each cluster.

We empirically evaluate the performance of our framework on two large-scale benchmark datasets. Our experimental results indicate a notable superiority of our approach over various state-of-the-art baseline models on both datasets. Furthermore, the transferability analysis demonstrates our framework can be effectively transferred to diverse deep recommender models, thereby enhancing their performance. Additionally, our framework offers an efficiency advantage as it requires less inference time than competing baseline models.

2 RELATED WORK

This section introduces the main related works to our study, focusing on feature-based recommender models and AutoML approaches for recommendation systems.

2.1 Feature-based Recommender Models

Feature-based recommender models take sparse, high-dimensional features from users and items as input and transform them into low-dimensional representations to capture user preferences for improved recommendations. For example, Cheng et al. [4] propose Wide&Deep (W&D), a model composed of a linear module and a Multi-Layer Perceptron (MLP) layer to combine the benefits of memorization and generalization for recommender systems. Guo et al. [8] propose DeepFM that further integrates the power of factorization machines based on W&D to learn high-order feature interactions for recommendations. Recently, advanced neural networks, such as attention-based models [29], have been developed. However, these techniques apply a fixed embedding dimension for all features, which would downgrade the model performance and consume substantial computational resources.

2.2 AutoML for Recommendations

Automated Machine Learning (AutoML) has recently become a research hotspot due to its potential to automate the design process for recommender systems, minimizing human involvement. The research directions include feature selection [15, 33], embedding dimension search [37, 38], model architecture search [5], and other component search [30, 31, 36]. Feature selection involves selecting a subset of field features in recommendation systems. For example, AutoField [33] uses a simple controller based on differentiable architecture search [16] to select the top K field features. AdaFS [15] enhances AutoField by modifying the controller to assign feature weights to fields for different samples. The objective of embedding dimension search is to find mixed embedding sizes for each field. For example, AutoEmb [38] finds the optimal dimension for each feature using differentiable search [16]. AutoDim [37] selects the best dimension for each field from a group of candidate dimensions in the same way as AutoEmb. Model architecture search explores various network architectures and selects the optimal one [5].

Our method is in alignment with embedding dimension search. All instances in the above methods share a uniform dimension size for each field. In contrast, our approach adaptively selects dimensions for each instance via the proposed Bernoulli gates, thereby considering the difference between individuals. Moreover, we introduce a polarization regularizer to overcome the shortcomings of the commonly-used top K selection strategy. Furthermore, rather than processing all samples through a single model, we propose to divide samples into clusters and train different recommender models with optimal dimensions tailored to different clusters. These unique and innovative designs in our proposed method have been proven effective in terms of both performance enhancement and inference cost saving.

3 METHOD

This section introduces the technical specifications of the proposed iHAS framework, as visualized in Figure 1. We first provide a concise overview of the entire hierarchical training framework. Subsequently, the primary modules within our framework are described, as well as how to optimize them within each hierarchical stage.

3.1 Overview

The methodology for the iHAS framework comprises three stages: searching, clustering, and retraining. It involves three principal modules: deep recommendation models (consisting of an embedding layer and an MLP component) for predicting user preferences, a Bernoulli gates layer responsible for dimension selection, and a K-means cluster algorithm that partitions the heterogeneous data.

In the searching stage, the key objective is to identify the optimal, instance-wise embedding dimensions across different fields, thus facilitating an accurate recommendation prediction. As shown in Figure 1, the categorical field features are directed to an embedding layer to generate embedding representations. These representations are then processed through the Bernoulli gates to produce embedding masks using a stochastic selection mode (see Section 3.3.1). Each embedding mask comprises a binary vector that serves as a gate on whether the corresponding dimension should be incorporated into the downstream architecture. Then the framework conducts an element-wise multiplication between a sample's embedding representations and embedding masks. This resultant masked embedding representation is then directed to the base MLP component to predict user preference.

In the clustering stage, the main objective is to utilize K-Means algorithm [28] to cluster samples into groups. This stage mostly mirrors the procedure used in the searching stage: using the embedding layer and Bernoulli gates (but using deterministic selection mode, see Section 3.3.2) to calculate the masked embedding representations. These masked embedding representations are then used to train a mini-batch K-Means cluster [28]. As shown in Figure 1, the K-means separates the red and yellow samples from the blue and green samples.

The retraining stage aims to develop cluster-customized DLRMs, considering the variation in dimension patterns across different clusters. In each cluster, we calculate the embedding masks (using deterministic selection mode) for all samples. The resultant masks are then averaged to obtain one vector, which is used to determine the final embedding dimensions of each DLRM.

3.2 Deep Learning Recommender Models

In this subsection, we provide a brief introduction to the basic architecture of the DLRM. It typically comprises two primary components: an embedding layer and an MLP component.

3.2.1 Embedding Layer. In classic DLRM, the embedding layer is commonly used to convert the categorical inputs into a dense vector of real numbers.

Let us denote the input of N categorical field features for sample i as $\boldsymbol{X}_i = [\boldsymbol{x}_{i,1}, \cdots, \boldsymbol{x}_{i,n}, \cdots, \boldsymbol{x}_{i,N}]$, where $\boldsymbol{x}_{i,n} \in \mathbb{Z}^{|n|}$ represents the one-hot vector comprising sparse, high-dimensional binary values. The term $|n|$ denotes the number of unique values for n-th categorical field. For instance, a categorical field such as "gender" with unique values – `male`, `female`, and `unknown` – can be expressed through three-bit vectors $[1, 0, 0]$, $[0, 1, 0]$, and $[0, 0, 1]$, respectively. To process a numerical field feature, we will discretize it through custom-width binning, followed by applying a one-hot operation. Then, the operation of the embedding layer can be represented as:

$$\boldsymbol{e}_{i,n} = \boldsymbol{v}_n \boldsymbol{x}_{i,n},$$

where $\boldsymbol{v}_n \in \mathbb{R}^{d \times |n|}$ is the embedding table of the n-th field, d is the predefined embedding dimension (typically consistent across all fields), and $\boldsymbol{e}_{i,n}$ is the low-dimensional embedding representation. Therefore, the final embedding of the input data $\boldsymbol{X}_i$ through N embedding tables is $\boldsymbol{E}_i = [\boldsymbol{e}_{i,1}, \cdots, \boldsymbol{e}_{i,n}, \cdots, \boldsymbol{e}_{i,N}]$.

Notably, the embedding dimension search techniques we discussed earlier in Section 2.2 (also the focus of this paper) aim at searching the optimal dimensions for embedding tables $\boldsymbol{V} = [\boldsymbol{v}_1, \cdots, \boldsymbol{v}_n, \cdots, \boldsymbol{v}_N]$. Specifically, our goal is to discover the optimal individual embedding dimension for each field, given the inherent diversity in the heterogeneous dataset. This could potentially enhance prediction performance.

3.2.2 MLP Component. The MLP component plays a crucial role in DLRMs, tasked with encoding embedding representations and predicting the recommendation. Empirically, it comprises multiple fully-connected (FC) layers (characterized by parameter $\boldsymbol{\theta}$) and is also equipped with non-linear activation functions such as ReLU [1] or Sigmoid, thereby facilitating the nonlinear encoding process of these representations.

In the iHAS system, we will train three different DLRMs, as illustrated in Figure 1. Each DLRM consists of an embedding layer and an MLP component. These three models are named the base recommender model, recommender model 1, and recommender model 2, which are characterized by the parameter groups $\{\boldsymbol{V}, \boldsymbol{\theta}\}_b$, $\{\boldsymbol{V}, \boldsymbol{\theta}\}_1$, and $\{\boldsymbol{V}, \boldsymbol{\theta}\}_2$, respectively.

3.3 Bernoulli Gates

Bernoulli gates operate as switches, facilitating the transmission of a sample's information from embedding tables to the downstream MLP component. Analogous to the ℓ_0 norm, we hope these "switches" to be capable of fully opening or closing without compromising the information integrity (shrinking the embedding representation). Inspired by the approach presented in [18, 35], we use Bernoulli gates to predict each sample's relevant dimensions given its embedding representation. The detailed process of the Bernoulli gates is graphically depicted in Figure 2.

The Bernoulli gates operate in two distinct modes: stochastic selection and deterministic selection. Under the stochastic selection mode, the gates operate as independent Bernoulli distributions, to independently "open" or "close" dimensions given the probabilities. The principle behind stochastic selection rests on the assumption that, given a sufficiently large number of training iterations, the gates will stochastically and comprehensively traverse all potential combinations of dimensions. This prompts the Bernoulli parameters to increase for beneficial dimensions and penalize unhelpful ones.

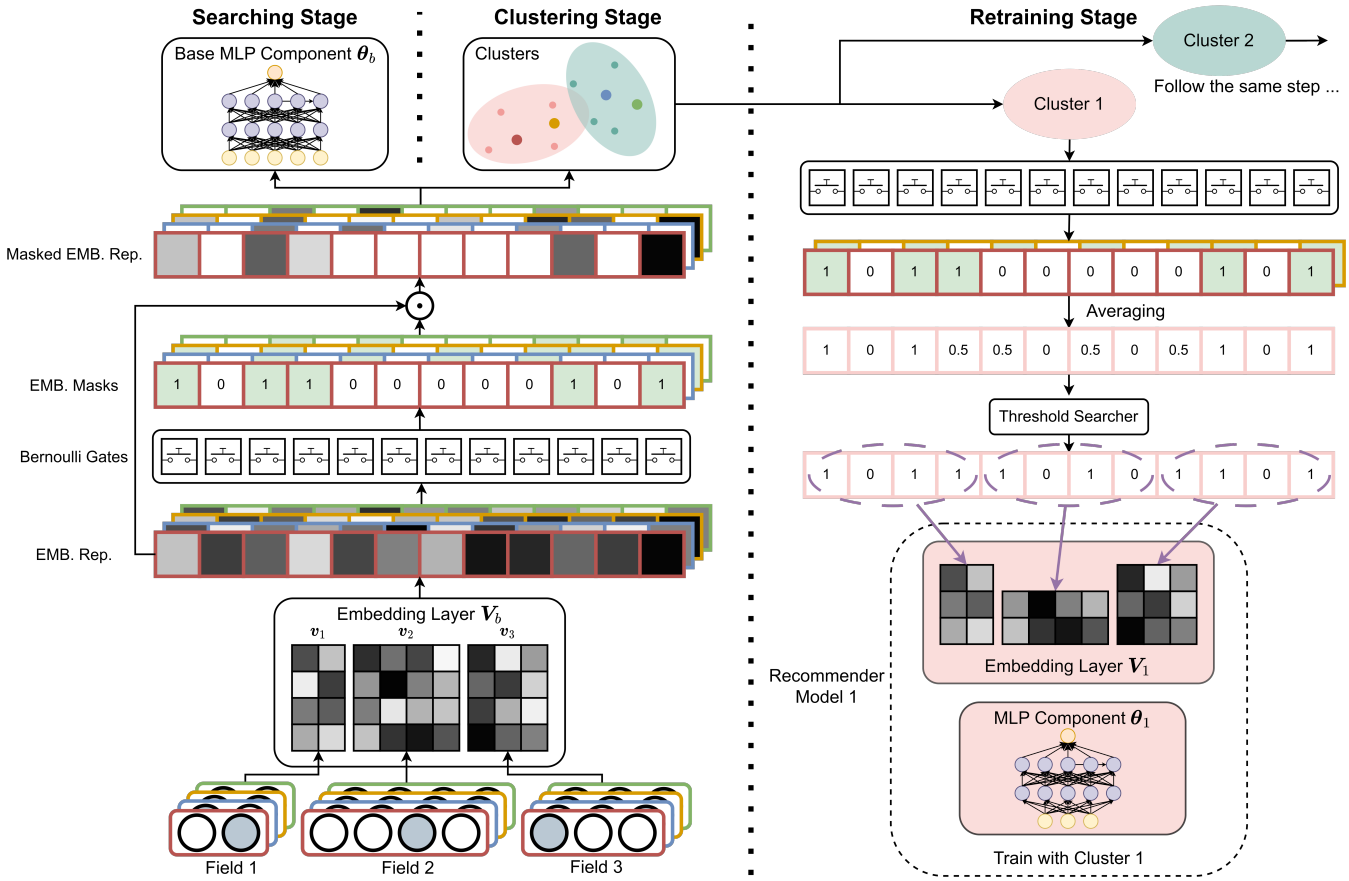

Figure 1: Overview of the three-stage training process in the iHAS framework. The four edge colors of the vectors (red, blue, yellow, and green) correspond to four different samples, which also correspond to the points in the clusters. The brightness level indicates the values of an element. "$\odot$" represents the element-wise product operation.

Once the Bernoulli distributions (gates) have been fully explored, we capitalize on the learned distribution by deterministically opening the most advantageous dimensions in the deterministic selection mode. However, learned Bernoulli probabilities often exhibit heavy-tailedness, making it challenging to distinguish between important and unimportant dimensions. To mitigate this, we suggest employing a polarization regularizer and an automatic threshold searcher (both discussed in Section 3.3.3) inside Bernoulli gates.

3.3.1 Stochastic Selection. In our previous discussion, we aim for Bernoulli gates to function as independent Bernoulli distributions in the stochastic selection mode during the searching stage. The first objective is to encode the embedding representations to the desired independent Bernoulli probabilities. To this end, we employ an FC layer (with parameter $\boldsymbol{w}$) and a Sigmoid activation layer (σ) to project these embedding representations of the i-th sample onto Bernoulli probabilities (upper left of Figure 2), denoted by $\{p_{i,j}\}_{j=1}^{N^*} = \sigma(\boldsymbol{w}\,\boldsymbol{E}_i)$, where $N^* = N \times d$ is the total length of the embedding representations. This enables us to initiate a combinatorial search process over the space of Bernoulli probabilities and FC parameters.

However, optimizing a loss function, which includes discrete RVs (Bernoulli distributions), incurs high variance [23]. To overcome this obstacle, we adopt Gumbel-Softmax [11] (aka Concrete distribution [21]), which offers a viable continuous approximation to the Bernoulli distribution, as visualized in Figure 2 (right).

Recall that Gumbel-Max [7, 22] is an effective method for drawing samples from a Bernoulli distribution (or any type of discrete random variables), as long as we provide the class probabilities. For instance, if we use $p_{i,j}$ and $1 - p_{i,j}$ as the probabilities for selecting and not selecting the j-th embedding dimension for sample i, respectively, the Gumbel-Max trick to approximate Bernoulli distribution sampling can then be expressed as:

$$z_{i,j} = \text{one hot}\left(\arg\max\left(\log p_{i,j} + G_{i,j}\,,\ \log(1 - p_{i,j}) + G'_{i,j}\right)\right), \quad (1)$$

where $G_{i,j}$ and $G'_{i,j}$ are i.i.d. samples drawn from a Gumbel distribution with the location at 0 and scale of 1, denoted as Gumbel(0, 1). However, both the "one hot()" and "arg max()" operations are non-differentiable, making them intractable for gradient descent optimization. Therefore, the softmax function is used as a continuous,

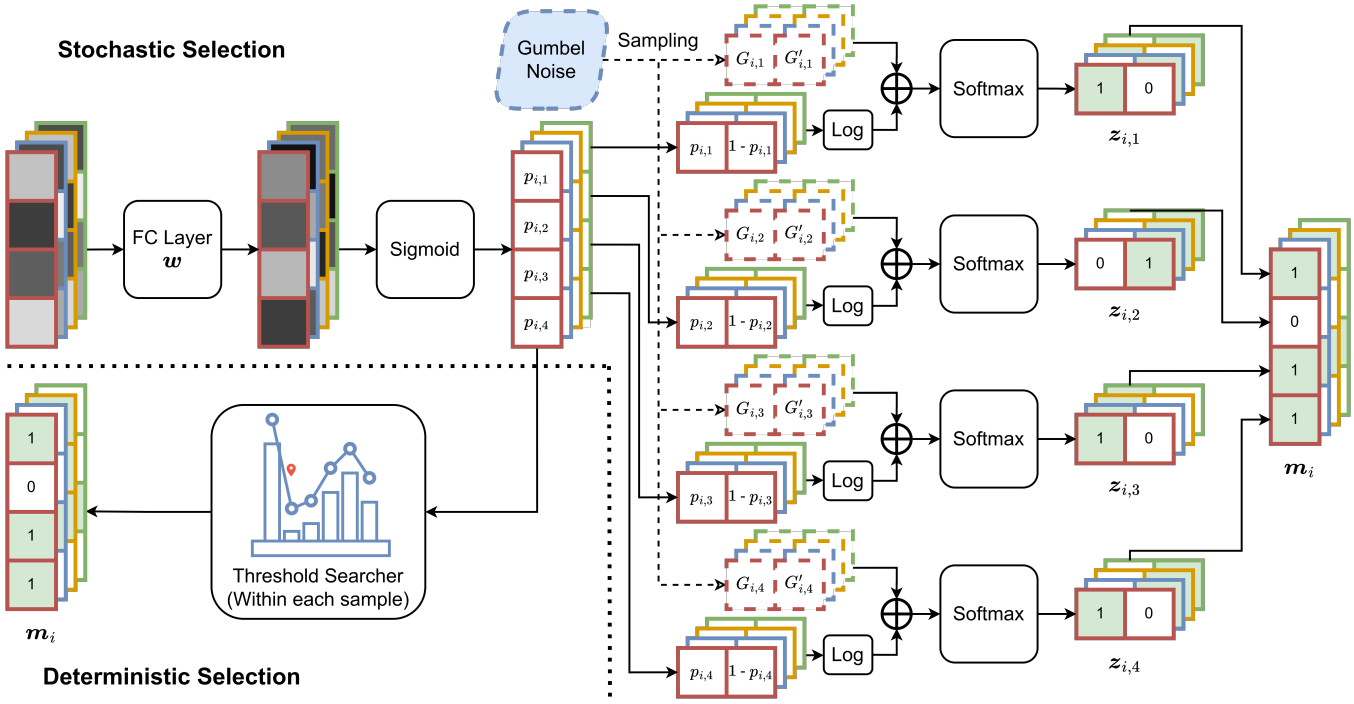

Figure 2: The detailed process of Bernoulli gates to generate embedding masks from the embedding representation. "⊕" represents the element-wise summation operation.

differentiable approximation of these operations. The softmax function uses a temperature parameter $\tau \in \mathbb{R}^+$ to regulate the approximation degree (or the entropy of the distribution), as formalized:

$$z_{i,j} = \frac{\left[\exp(\,(\log p_{i,j} + G_{i,j})\,/\,\tau\,),\ \exp(\,(\log(1-p_{i,j}) + G'_{i,j})\,/\,\tau\,)\right]}{\exp(\,(\log p_{i,j} + G_{i,j})\,/\,\tau\,)\ +\ \exp(\,(\log(1-p_{i,j}) + G'_{i,j})\,/\,\tau\,)}. \tag{2}$$

As τ approaches 0, $z_{i,j}$ approximates the true binary vector, making the Gumbel-Softmax distribution become identical to the desired Bernoulli distribution. Then the final embedding masks, $\boldsymbol{m}_i$, are created by concatenating the first bit of $\{z_{i,j}\}_{j=1}^{N^*}$.

However, our goal remains to produce true binary masks, which would effectively eliminate information from unimportant dimensions, as opposed to significantly shrinking them. The straight-through (ST) Gumbel-Softmax [2, 11] serves well in this context. In the ST variant, the operation from Equation 1 is implemented in the forward pass while the continuous approximation from Equation 2 is used in the backward gradient descent. This approach enables sparse selection even when the temperature τ is high, while still allowing the gradient to propagate and update the parameters.

3.3.2 Deterministic Selection. After training the Bernoulli probabilities ($p_{i,j}$) during the searching phase, we utilize these probabilities to determine which dimensions will contribute to the accuracy of the recommendation predictions. However, $p_{i,j}$ are characterized by high variance and heavy-tailedness, as shown by the histogram in Figure 3 (left). These present two complications: (1) distinguishing important dimensions from unimportant ones becomes challenging;

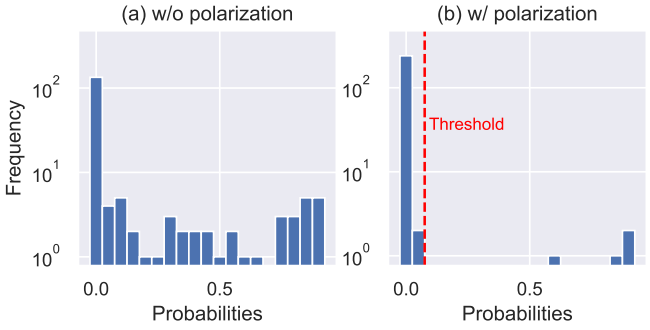

Figure 3: Histogram of the Bernoulli probabilities for a sample from Avazu dataset, trained (a) with and (b) without polarization regularizer. Note that y-axes use log scales, and within the same range, to facilitate better visual comparison.

and (2) even the unimportant dimensions still possess a small probability of being selected. Moreover, masks created using Bernoulli gates introduce an element of randomness (Gumbel noise), which hinders their direct application during inference (where given the same data each time, consistent results should be generated).

To overcome these limitations, we propose a deterministic selection mode that directly selects the beneficial dimensions using the knowledge derived from the well-trained Bernoulli probabilities. This process is outlined in Figure 2 (bottom left). Firstly, we use the same FC layer and sigmoid layer to estimate the Bernoulli probabilities, $p_{i,j}$, analogous to the first step in the stochastic selection mode.

Then, for each sample i, we search a threshold among the probabilities $\{p_{i,j}\}_{j=1}^{N^*}$ (see details in Section 3.3.3). We automatically adjust the gates to be open for probabilities exceeding this threshold and closed for those falling below it. The resulting embedding masks are utilized during the clustering and retraining phase (see Figure 1).

3.3.3 Polarization and Automatic Threshold Searcher. Let's consider an empirical optimization procedure with an ℓ_0 regularization on the embedding masks during the searching stage:

$$\mathcal{R}(\{\boldsymbol{V}, \boldsymbol{\theta}\}_b, \boldsymbol{m}) = \mathbb{E}_i \mathbb{E}_{\boldsymbol{m}_i} \left[\mathcal{L}(f_{\boldsymbol{\theta}_b}(\boldsymbol{V}_b \cdot \boldsymbol{X}_i \odot \boldsymbol{m}_i), y_i) \right], \quad (3)$$

$$\text{with regularizer} \quad \mathcal{R}(\boldsymbol{m}) = \mathbb{E}_i \mathbb{E}_{\boldsymbol{m}_i} [\, \lambda \|\boldsymbol{m}_i\|_0 \,], \quad (4)$$

where $\mathcal{L}(\cdot, \cdot)$ represents the binary cross-entropy (BCE) loss, $f_{\boldsymbol{\theta}_b}(\cdot)$ represents the base MLP component with parameters $\boldsymbol{\theta}_b$, y_i is the ground truth label for the i-th sample, and λ is a balancing factor for the regularizer. Since we use Gumbel-Softmax to produce the embedding masks, $\boldsymbol{m}_i$, the term $\mathbb{E}_i \mathbb{E}_{\boldsymbol{m}_i} \|\boldsymbol{m}_i\|_0$ effectively equals to the sum of Bernoulli probabilities, $\sum_i \sum_{j=1}^{N^*} p_{i,j}$.

Our experimentation, however, indicates that simply using this ℓ_0 regularizer still experiences the heavy-tailedness and distinction difficulties, as discussed in Section 3.3.2. Inspired from Zhuang et al. [41], we incorporate a polarization regularizer into Equation 4:

$$\mathcal{R}(\boldsymbol{m}) = \sum_i \sum_{j=1}^{N^*} \lambda\, p_{i,j} - |p_{i,j} - \bar{p}_i|, \quad \text{with } \bar{p}_i = \frac{1}{N^*} \sum_{j=1}^{N^*} p_{i,j}. \quad (5)$$

The intuition of the second term (which is the polarization regularizer) is to maximally distance $p_{i,j}$ from their mean $\bar{p}_i$. Empirically, we have observed this polarization term effectively separates the probabilities of important and unimportant dimensions into two groups, thereby making them distinguishable (see Figure 3).

Despite employing the regularizer as stated in Equation 5, a threshold searcher is still required to identify the threshold for $p_{i,j}$. Currently, the histogram of $p_{i,j}$ trained with the polarization regularizer should have at least two peaks, with one of them located close to 0. Following the strategy from [41], we scan the histogram from left to right to identify the first saddle point, i.e., the bin that contains the local minimum (the red pin in Figure 2). The lower bound of this bin is subsequently set as the threshold.

3.4 Clusters

Once obtaining a sample's unique embedding masks, our aim is to pass this sample through a recommender model tailored to its distinct dimension selection. This implies that, for every sample with its individual selected dimensions, there always exists a customized recommender model capable of handling this distinct input and generating a precise prediction. Nonetheless, training an extensive number of recommender models is impractical due to the constraint of finite data size. For example, with a obviously underestimated dimension, $N^* = 100$, the number of possible combinations of dimensions is $\sum_{k=0}^{100} \binom{100}{k} = 2^{100}$, meaning at least 2^{100} distinct samples should be collected.

Consequently, we must strike a balance between using a one-size-fits-all recommender model (inadequate for highly heterogeneous samples) and using a fully-customized recommender model for each sample. Our resolution is to leverage clustering algorithms to segregate samples into separate groups. We anticipate that samples within the same group display similar patterns, which could be used to explore identical dimension patterns and identical recommender models. Furthermore, partitioning the data into clusters facilitates the training of small and manageable models for each cluster, thus accelerating the inference speed.

In this iHAS framework, we opt for the mini-batch K-Means clustering algorithm [28] to partition the samples. We chose this variant due to its computational efficiency in handling large datasets (comprised of tens of millions of samples), and its capacity to avoid getting stuck at local optima.

Algorithm 1 Optimization Strategy for iHAS

Require: Training dataset $\mathcal{D}_{\text{train}}$, validation dataset $\mathcal{D}_{\text{val}}$
Ensure: Bernoulli gates parameters $\boldsymbol{w}$, K-Means with two clusters, base recommender model $\{\boldsymbol{V}, \boldsymbol{\theta}\}_b$, recommender model 1 $\{\boldsymbol{V}, \boldsymbol{\theta}\}_1$, and recommender model 2 $\{\boldsymbol{V}, \boldsymbol{\theta}\}_2$

1: ### **Searching** ###
2: Pretrain $\{\boldsymbol{V}, \boldsymbol{\theta}\}_b$ for 5 epoch using $\mathcal{D}_{\text{train}}$
3: **while** not converge on $\mathcal{D}_{\text{val}}$ **do**
4: sample a mini-batch $\in \mathcal{D}_{\text{train}}$, get $\boldsymbol{m}$ by stochastic selection
5: update $\{\boldsymbol{V}, \boldsymbol{\theta}\}_b$ according the objective from Eq. 3
6: sample a mini-batch $\in \mathcal{D}_{\text{val}}$, get $\boldsymbol{m}$ by stochastic selection
7: update $\boldsymbol{w}$ according the objective from Eq. 3 and 5
8: **end while**
9: ### **Clustering** ###
10: **while** not converge on cluster centroids **do**
11: sample a mini-batch $\in \mathcal{D}_{\text{train}}$, get $\boldsymbol{m}$ by deterministic selection, assign samples to their closest centroid
12: update the cluster centroids of K-means
13: **end while**
14: ### **Retraining** ###
15: Based on the K-Means cluster, split $\mathcal{D}_{\text{train}} \rightarrow \mathcal{D}_{\text{train 1}}, \mathcal{D}_{\text{train 2}}$, and split $\mathcal{D}_{\text{val}} \rightarrow \mathcal{D}_{\text{val 1}}, \mathcal{D}_{\text{val 2}}$
16: find the optimal dimensions for $\mathcal{D}_{\text{train 1}}$ via Bernoulli gates, and initialize $\{\boldsymbol{V}, \boldsymbol{\theta}\}_1$ with the corresponding dimensions
17: **while** not converge on $\mathcal{D}_{\text{val 1}}$ **do**
18: sample a mini-batch from $\mathcal{D}_{\text{train 1}}$
19: update $\{\boldsymbol{V}, \boldsymbol{\theta}\}_1$ according the objective from Eq. 3
20: **end while**
21: repeat lines 16-20 with $\mathcal{D}_{\text{train 2}}$, $\mathcal{D}_{\text{val 2}}$, and $\{\boldsymbol{V}, \boldsymbol{\theta}\}_2$

3.5 Optimization Strategy

In the iHAS framework, we adopt a hierarchical training strategy to optimize different modules in different stages, inspired by [12, 15, 24, 33], to tackle the encoding issue of jointly optimizing all modules. The detailed optimization strategy in each stage is illustrated in Algorithm 1.

3.5.1 Searching Stage. As mentioned in [35], the use of Gumbel-Softmax approximation for the discrete random variable suffers from high variance, which can lead to inconsistency in the set of selected dimensions. Inspired by [15], we first pretrain the base recommender model for a few epochs to obtain a tentative reliable embedding representation. During these pretrain epochs, the Bernoulli gates are always deterministically open (no matter what embedding representation it receives). Then, after we've obtained a

Table 1: The statistics of Avazu and Criteo datasets.

Dataset	#Instances	#Fields	#Features
Avazu	40,400,000	22	645,394
Criteo	45,840,617	39	1,086,810

tentative reliable embedding representation, we initialize the parameters for the Bernoulli gates and start the stochastic selection. Later we adopt the bi-level optimization strategy [12, 24] to disjointly update the parameters $\boldsymbol{w}$ in Bernoulli gates and the parameters $\{\boldsymbol{V}, \boldsymbol{\theta}\}_{\mathrm{b}}$ in the base recommender model.

3.5.2 Clustering Stage. In the clustering stage, we can obtain the masked embedding representations using the embedding tables and Bernoulli gates which have been trained during the searching stage. Remember we use deterministic selection mode for Bernoulli gates to generate embedding masks in this clustering stage. For each sample, compute its Euclidean distance to the centroid using the masked embedding representations, then assign it to the nearest centroid (group), and later update the centroids.

3.5.3 Retraining Stage. As to the retraining stage, we first divide all samples via the trained K-Means cluster. Then we find the majority dimensions for each group from their embedding masks using the deterministic mode. After that, we initialize the deep recommender model 1 and 2 by their corresponding optimal dimensions and train them separately using the samples of each cluster.

4 EXPERIMENT

In this section, we conduct extensive experiments to evaluate our proposed framework. Specifically, the main research questions we care about are as follows:

- **RQ1**: How does iHAS perform compared with other mainstream selection methods?
- **RQ2**: Can the proposed iHAS be successfully transferred to more powerful recommender models?
- **RQ3**: How does each component contribute to the overall performance of the proposed iHAS?
- **RQ4**: Does the proposed iHAS demonstrate efficiency when compared to baseline models?
- **RQ5**: Does iHAS construct rational recommender model structures?

4.1 Datasets

We conduct our experiments mainly on two commonly used public datasets, Avazu[1] and Criteo[2], which are both large-scale real-world datasets and serve as benchmarks in click-through rate (CTR) prediction tasks. Table 1 presents the detailed statistics of both datasets. Each dataset has been randomly segmented into training/validation/testing sets based on the proportions of 80%, 10%, and 10%.

- **Avazu** dataset consists of 40 million users' click records on ads over 11 days. Each record contains 22 categorical field features. Following the general preprocessing steps [29, 40], we group fields of which frequency is less than ten as a single field "others".
- **Criteo** dataset consists of 46 million users' click records on display ads. Each record contains 26 categorical fields and 13 numerical fields. we use the preprocessing method as Avazu for the low-frequency fields (less than ten) and transform each numerical field x by $log^2(x)$ if $x > 2$.

4.2 Evaluation Metrics

Following the previous works [17, 25], we evaluate the performance of our method using two common metrics: **AUC** and **Logloss**. AUC refers to the area under the ROC curve, which means the probability that a model will rank a randomly selected positive instance higher than a randomly selected negative one. A higher AUC value indicates superior model performance. On the other hand, Logloss, aka binary cross-entropy loss, directly quantifies the model's performance, with a lower score denoting more accurate predictions. Note that a marginal **0.001-level** improvement in AUC (increase) or Logloss (decrease) is perceived as a significant enhancement in model performance [19, 33, 37].

4.3 Baseline Methods

We compare our proposed method with the following state-of-the-art methods:

- **PEP** [17]: It adopts trainable thresholds to prune redundant embedding dimensions.
- **AutoField** [33]: It utilizes neural architecture search techniques [16] to select important field features.
- **OptEmbed** [20]: It trains a supernet with various selected embedding dimensions, then uses evolution search to find the optimal embedding dimensions based on the supernet.
- **AdaFS** [15]: It assigns weights to different fields in a soft manner (AdaFS-soft) or masks unimportant fields in a hard manner (AdaFS-hard) via a novel controller network.
- **OptFS** [19]: It simultaneously selects optimal field features and the optimal interactions between these features using "binary gates".

4.4 Implementation Details

We implement our method based on a public library[3] that involves sixteen commonly-used DLRMs. As our framework is model-agnostic, it can be seamlessly integrated with any of these models, see Section 4.6. For the embedding layer, we set the initial embedding size of all fields as 16 in accordance with the previous works [15, 33]. For the MLP component, we adopt two fully-connected layers of size $(16, 8)$ with the ReLU activation function. We use Adam optimizer [13] with an initial learning rate of 0.001, and weight decay of 1e-6. The batch size is set to 2048. We sample one validation batch every 100 training batches for bi-level optimization. The temperature τ for ST Gumbel-Softmax is set to 0.1.

The baseline models are implemented by the codes provided by their authors. For a fair comparison, we set the initial embedding dimension as 16 for all baselines. All the experiments are run on a single machine with an Nvidia RTX 3090 GPU.

[1] https://www.kaggle.com/c/avazu-ctr-prediction/
[2] https://www.kaggle.com/c/criteo-display-ad-challenge/
[3] https://github.com/rixwew/pytorch-fm

Table 2: Performance comparison between iHAS and baseline models.

Dataset	Metric	Methods						
		PEP	AutoField	OptEmbed	AdaFS-soft	AdaFS-hard	OptFS	iHAS
Avazu	AUC ↑	0.7665	0.7773	0.7630	0.7777	0.7763	0.7724	**0.7815**
	Logloss ↓	0.3874	0.3813	0.3894	0.3812	0.3821	0.3840	**0.3791**
Criteo	AUC ↑	0.8006	0.8029	0.7962	0.8039	0.8031	0.8015	**0.8043**
	Logloss ↓	0.4507	0.4490	0.4543	0.4484	0.4560	0.4504	**0.4478**

4.5 Overall Performance (RQ1)

Table 2 compares the overall performance of our proposed iHAS and other baseline models on the Avazu and Criteo datasets. We summarize our observations below.

First, our iHAS outperforms all the state-of-the-art baseline methods as it can achieve higher AUC and lower Logloss on both datasets, demonstrating the effectiveness of iHAS in deep recommendation systems. Specifically, iHAS outperforms the runner-ups by 0.0038 (AUC) and 0.0021 (Logloss) on the Avazu datasets, and by 0.0004 (AUC) and 0.0006 (Logloss) on the Criteo datasets.

Secondly, among all baselines, AdaFS-soft is the most effective model for the Avazu and Criteo datasets. However, it only shrinks the field features using a feature weighting layer and therefore does not completely eliminate the effect of unimportant fields. Although AdaFS-hard attempts to mask unimportant fields by uniformly keeping the top K features, the trained feature weights may still exhibit a high variance pattern (remember the non-distinguishable probabilities in Figure 3, left panel). Therefore, this top K selection manner may lead to selecting unimportant features or omitting the important feature in their final model, further compromising the model performance. Our polarization regularizer and threshold searcher can help with this issue, as detailed in Section 3.3.3 and evidenced by the empirical ablation study in Section 4.7.

Lastly, other baselines apply global feature/dimension selection across all samples, which fails to account for inherent variations among heterogeneous individuals, and consequently leads to suboptimal performance. Additionally, PEP mainly emphasizes on the model size, i.e., it stops searching once the embedding table reaches a predefined parameter size. This approach may result in a suboptimal embedding table due to overlooking the model performance. AutoField[4] also employs the top K selection manner, again leading to feature misselection and inferior model performance.

4.6 Transferability Analysis (RQ2)

In this subsection, we explore the transferability of iHAS. Specifically, we freeze the parameters of the well-trained Bernoulli gates and utilize them to help train other popular deep recommendation models, including FM [26], W&D [4], and DeepFM [8].

Table 3 shows the experimental results on Avazu, where "original" refers to the corresponding model without any selection. We can observe that: (i) all the recommendation models have great improvement by adopting iHAS, which again demonstrates the importance of performing selection in the recommendations; (ii) The transferability of iHAS is better than the best baseline (by comparing the iHAS and AdaFS-soft in Table 3), which validates the effectiveness of our Bernoulli gates.

In summary, we conclude that iHAS has outstanding transferability across different recommendation models, which enables it to be leveraged in complicated real-world recommender systems.

[4]The performance score for AutoField is borrowed from its original paper [33] as they use the same experimental settings and have not publicly released the codes.

Table 3: Transferability of iHAS on the Avazu dataset.

Model	Metric	Transfer Type		
		Original	AdaFS-soft	iHAS
FM	AUC ↑	0.7766	0.7799	**0.7826**
	Logloss ↓	0.3815	0.3797	**0.3793**
W&D	AUC ↑	0.7772	0.7790	**0.7797**
	Logloss ↓	0.3815	0.3802	**0.3800**
DeepFM	AUC ↑	0.7806	0.7817	**0.7840**
	Logloss ↓	0.3795	0.3786	**0.3784**

Table 4: Ablation study on the Avazu datasets.

Metric	Methods					
	Base	iHAS-1	iHAS-2	iHAS-3	iHAS-4	iHAS
AUC ↑	0.7765	0.7772	0.7767	0.7801	0.7768	**0.7815**
Logloss ↓	0.3818	0.3813	0.3816	0.3800	0.3817	**0.3791**

4.7 Ablation Study (RQ3)

In this subsection, we conduct the ablation study of key components in iHAS, as shown in Table 4. The Base model keeps all fields and the uniform embedding dimensions without any selections, and we derive four variants from iHAS: (i) iHAS-1: This variant is the model directly obtained in the searching stage, i.e., we remove the clustering and retraining stages; (ii) iHAS-2: This variant consists of a searching stage and a retraining stage. After we have the well-trained Bernoulli gates, we select dimensions across all samples to retrain one recommender model instead of separating samples into different clusters for retraining cluster-customized recommender models; (iii) iHAS-3: This variant is the standard iHAS without using the polarization regularizer described in Section 3.3.3; (iv) iHAS-4: This variant doesn't consider instance-wise differences by disconnecting the Bernoulli probabilities with the embedding representations of each sample. That means the Bernoulli probabilities become $\{p_j\}_{j=1}^{N^*} = \sigma(w^*)$ where w^* is simply a randomly

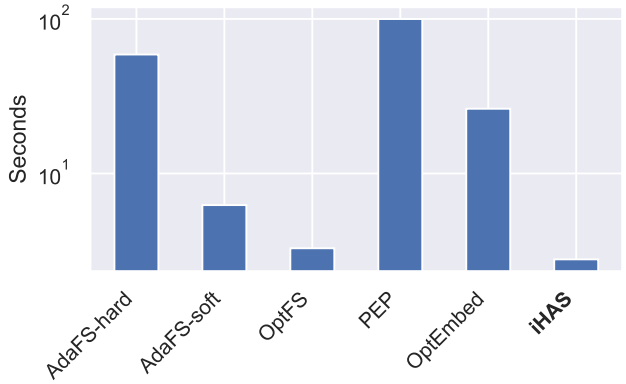

Figure 4: Inference time (in log scale) of iHAS and other baselines on the Avazu dataset.

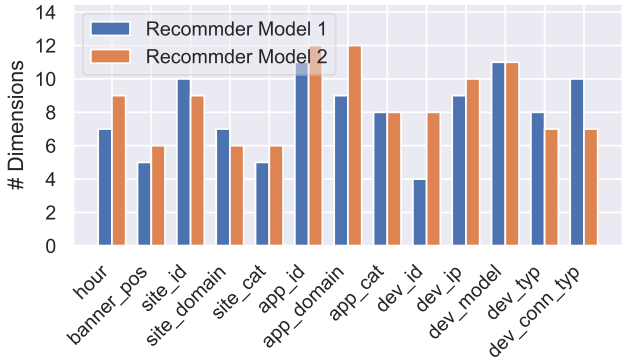

Figure 5: Case study of selected dimensions of each field for each DLRM in iHAS on the Avazu dataset.

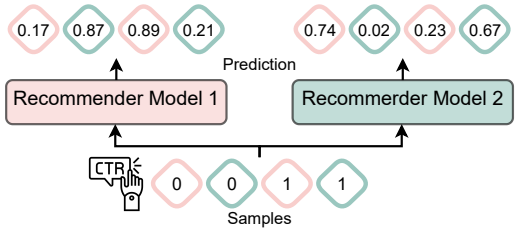

Figure 6: Example data predictions on the Avazu dataset, where the samples with the same edge color belong to the same cluster. The ground truths and prediction scores are displayed in the diamonds.

initialized vector of the dimension length, making every sample share the same probability for every dimension.

Based on the results in Table 4, we can find: (i) iHAS and its variants can increase the AUC and decrease the Logloss compared with the Base model, which indicates the necessity of performing selection on embedding dimensions for boosting model performance; (ii) iHAS performs better than iHAS-1, which indicates the necessity of the subsequent clustering and retraining stages; (iii) iHAS-1 outperforms iHAS-2, therefore, separating instances into different groups, i.e., the clustering stage, is beneficial for boosting the performance; (iv) Polarization is vital for acquiring better Bernoulli gates by comparing iHAS with iHAS-3; (v) Respecting the difference between instances can further boost the performance by comparing iHAS with iHAS-4.

4.8 Efficiency Analysis (RQ4)

In addition to model performance, efficiency is vital when deploying the recommendation model into online systems, especially inference efficiency. We report the inference time on the whole test set of iHAS and other baselines in Figure 4. We can find that iHAS achieves the least inference time. This is because iHAS feed different test data into its preferred recommender model of smaller size instead of feeding all test data into a single model which may lead to additional inference cost on some data. On the contrary, PEP requires the longest inference time because its embedding table is usually sparse and hardware-unfriendly.

4.9 Case Study (RQ5)

In this subsection, we first use a case study to investigate the optimal embedding dimensions for each cluster from iHAS. We show the results on Avazu as an example and exclude all anonymous field features in Figure 5.

We can observe that: (i) Each field's optimal dimensions greatly vary from one to another (from 4 to 12), which highlights the necessity of dimension search in recommender systems; (ii) id-related features, e.g., site_id and app_id, typically possess more dimensions. This aligns with human intuition as the id-related features are the core of recommender systems; (iii) Samples within different clusters tend to select different dimensions for each field, which validates our claim that different clusters present different patterns and should be trained separately to enhance performance and reduce inference time in Section 3.4.

Furthermore, we use four samples to illustrate the effectiveness of the iHAS framework consisting of group-customized recommender models. Figure 6 shows four samples grouped into two clusters (two in pink and two in cyan). Each cluster has its customized recommender model. We can find that the predictions are more correct (lower Logloss) if we feed the sample into its corresponding model. However, if feeding all of them together into one of the recommender models, we will receive some wrong predictions.

5 CONCLUSION

This study proposes an instance-wise Hierarchical Architecture Search framework, iHAS, as an innovative solution to the challenges associated with identifying optimal embedding dimensions for DLRMs. iHAS employs a three-stage hierarchical training strategy including searching, clustering, and retraining. The searching stage aims to identify the optimal embedding dimensions for each sample across different fields. Subsequent stages of clustering and retraining provide a mechanism for gathering similar samples as clusters and training cluster-customized DLRMs based on the individual optimal dimensions, thereby enhancing recommendation predictions. We conduct extensive experiments on two large-scale datasets to authenticate the efficacy of the proposed framework. The results demonstrate that iHAS could boost the performance of deep recommendations while reducing inference costs. Additionally, iHAS exhibits outstanding transferability to popular DLRMs.

REFERENCES

[1] Abien Fred Agarap. 2018. Deep learning using rectified linear units (relu). *arXiv preprint arXiv:1803.08375* (2018).

[2] Yoshua Bengio, Nicholas Léonard, and Aaron Courville. 2013. Estimating or propagating gradients through stochastic neurons for conditional computation. *arXiv preprint arXiv:1308.3432* (2013).

[3] Olivier Chapelle, Eren Manavoglu, and Romer Rosales. 2014. Simple and scalable response prediction for display advertising. *ACM Transactions on Intelligent Systems and Technology (TIST)* 5, 4 (2014), 1–34.

[4] Heng-Tze Cheng, Levent Koc, Jeremiah Harmsen, Tal Shaked, Tushar Chandra, Hrishi Aradhye, Glen Anderson, Greg Corrado, Wei Chai, Mustafa Ispir, et al. 2016. Wide & deep learning for recommender systems. In *Proceedings of the 1st workshop on deep learning for recommender systems*. 7–10.

[5] Mingyue Cheng, Zhiding Liu, Qi Liu, Shenyang Ge, and Enhong Chen. 2022. Towards Automatic Discovering of Deep Hybrid Network Architecture for Sequential Recommendation. In *Proceedings of the ACM Web Conference 2022*. 1923–1932.

[6] Paul Covington, Jay Adams, and Emre Sargin. 2016. Deep neural networks for youtube recommendations. In *Proceedings of the 10th ACM conference on recommender systems*. 191–198.

[7] Emil Julius Gumbel. 1954. *Statistical theory of extreme values and some practical applications: a series of lectures*. Vol. 33. US Government Printing Office.

[8] 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*. 1725–1731.

[9] Ido Guy, Naama Zwerdling, Inbal Ronen, David Carmel, and Erel Uziel. 2010. Social media recommendation based on people and tags. In *Proceedings of the 33rd international ACM SIGIR conference on Research and development in information retrieval*. 194–201.

[10] Xiangnan He and Tat-Seng Chua. 2017. Neural factorization machines for sparse predictive analytics. In *Proceedings of the 40th International ACM SIGIR conference on Research and Development in Information Retrieval*. 355–364.

[11] Eric Jang, Shixiang Gu, and Ben Poole. [n. d.]. Categorical Reparameterization with Gumbel-Softmax. In *International Conference on Learning Representations*.

[12] Neil Jethani, Mukund Sudarshan, Yindalon Aphinyanaphongs, and Rajesh Ranganath. 2021. Have We Learned to Explain?: How Interpretability Methods Can Learn to Encode Predictions in their Interpretations.. In *International Conference on Artificial Intelligence and Statistics*. PMLR, 1459–1467.

[13] Diederik P Kingma and Jimmy Ba. 2014. Adam: A method for stochastic optimization. *arXiv preprint arXiv:1412.6980* (2014).

[14] Jianxun Lian, Xiaohuan Zhou, Fuzheng Zhang, Zhongxia Chen, Xing Xie, and Guangzhong Sun. 2018. xdeepfm: Combining explicit and implicit feature interactions for recommender systems. In *Proceedings of the 24th ACM SIGKDD international conference on knowledge discovery & data mining*. 1754–1763.

[15] Weilin Lin, Xiangyu Zhao, Yejing Wang, Tong Xu, and Xian Wu. 2022. AdaFS: Adaptive Feature Selection in Deep Recommender System. In *Proceedings of the 28th ACM SIGKDD Conference on Knowledge Discovery and Data Mining*. 3309–3317.

[16] Hanxiao Liu, Karen Simonyan, and Yiming Yang. 2018. DARTS: Differentiable Architecture Search. *arXiv preprint arXiv:1806.09055* (2018).

[17] Siyi Liu, Chen Gao, Yihong Chen, Depeng Jin, and Yong Li. [n. d.]. Learnable Embedding sizes for Recommender Systems. In *International Conference on Learning Representations*.

[18] Christos Louizos, Max Welling, and Diederik P Kingma. [n. d.]. Learning Sparse Neural Networks through L_0 Regularization. In *International Conference on Learning Representations*.

[19] Fuyuan Lyu, Xing Tang, Dugang Liu, Liang Chen, Xiuqiang He, and Xue Liu. 2023. Optimizing Feature Set for Click-Through Rate Prediction. In *Proceedings of the ACM Web Conference 2023*. 3386–3395.

[20] Fuyuan Lyu, Xing Tang, Hong Zhu, Huifeng Guo, Yingxue Zhang, Ruiming Tang, and Xue Liu. 2022. OptEmbed: Learning Optimal Embedding Table for Click-through Rate Prediction. In *Proceedings of the 31st ACM International Conference on Information & Knowledge Management*. 1399–1409.

[21] Chris J Maddison, Andriy Mnih, and Yee Whye Teh. [n. d.]. The Concrete Distribution: A Continuous Relaxation of Discrete Random Variables. In *International Conference on Learning Representations*.

[22] Chris J Maddison, Daniel Tarlow, and Tom Minka. 2014. A* sampling. *Advances in neural information processing systems* 27 (2014).

[23] Andriy Mnih and Danilo Rezende. 2016. Variational inference for monte carlo objectives. In *International Conference on Machine Learning*. PMLR, 2188–2196.

[24] Hieu Pham, Melody Guan, Barret Zoph, Quoc Le, and Jeff Dean. 2018. Efficient neural architecture search via parameters sharing. In *International conference on machine learning*. PMLR, 4095–4104.

[25] Liang Qu, Yonghong Ye, Ningzhi Tang, Lixin Zhang, Yuhui Shi, and Hongzhi Yin. 2022. Single-shot Embedding Dimension Search in Recommender System. In *Proceedings of the 45th International ACM SIGIR Conference on Research and Development in Information Retrieval*. 513–522.

[26] Steffen Rendle. 2010. Factorization machines. In *2010 IEEE International conference on data mining*. IEEE, 995–1000.

[27] Matthew Richardson, Ewa Dominowska, and Robert Ragno. 2007. Predicting clicks: estimating the click-through rate for new ads. In *Proceedings of the 16th international conference on World Wide Web*. 521–530.

[28] David Sculley. 2010. Web-scale k-means clustering. In *Proceedings of the 19th international conference on World wide web*. 1177–1178.

[29] Weiping Song, Chence Shi, Zhiping Xiao, Zhijian Duan, Yewen Xu, Ming Zhang, and Jian Tang. 2019. Autoint: Automatic feature interaction learning via self-attentive neural networks. In *Proceedings of the 28th ACM International Conference on Information and Knowledge Management*. 1161–1170.

[30] Yixin Su, Rui Zhang, Sarah Erfani, and Zhenghua Xu. 2021. Detecting beneficial feature interactions for recommender systems. In *Proceedings of the AAAI conference on artificial intelligence*, Vol. 35. 4357–4365.

[31] Yixin Su, Yunxiang Zhao, Sarah Erfani, Junhao Gan, and Rui Zhang. 2022. Detecting arbitrary order beneficial feature interactions for recommender systems. In *Proceedings of the 28th ACM SIGKDD Conference on Knowledge Discovery and Data Mining*. 1676–1686.

[32] Robert Tibshirani. 1996. Regression shrinkage and selection via the lasso. *Journal of the Royal Statistical Society: Series B (Methodological)* 58, 1 (1996), 267–288.

[33] Yejing Wang, Xiangyu Zhao, Tong Xu, and Xian Wu. 2022. Autofield: Automating feature selection in deep recommender systems. In *Proceedings of the ACM Web Conference 2022*. 1977–1986.

[34] 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*. 3119–3125.

[35] Yutaro Yamada, Ofir Lindenbaum, Sahand Negahban, and Yuval Kluger. 2020. Feature selection using stochastic gates. In *International Conference on Machine Learning*. PMLR, 10648–10659.

[36] Xiangyu Zhao, Haochen Liu, Wenqi Fan, Hui Liu, Jiliang Tang, and Chong Wang. 2021. Autoloss: Automated loss function search in recommendations. In *Proceedings of the 27th ACM SIGKDD Conference on Knowledge Discovery & Data Mining*. 3959–3967.

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

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

[39] Guanjie Zheng, Fuzheng Zhang, Zihan Zheng, Yang Xiang, Nicholas Jing Yuan, Xing Xie, and Zhenhui Li. 2018. DRN: A deep reinforcement learning framework for news recommendation. In *Proceedings of the 2018 world wide web conference*. 167–176.

[40] Jieming Zhu, Jinyang Liu, Shuai Yang, Qi Zhang, and Xiuqiang He. 2021. Open benchmarking for click-through rate prediction. In *Proceedings of the 30th ACM International Conference on Information & Knowledge Management*. 2759–2769.

[41] Tao Zhuang, Zhixuan Zhang, Yuheng Huang, Xiaoyi Zeng, Kai Shuang, and Xiang Li. 2020. Neuron-level structured pruning using polarization regularizer. *Advances in neural information processing systems* 33 (2020), 9865–9877.