AutoDPQ: Automated Differentiable Product Quantization for Embedding Compression

Xin Gan†
City University of Hong Kong
xingan98@yeah.net

Yuhao Wang†
City University of Hong Kong
yhwang25-c@my.cityu.edu.hk

Xiangyu Zhao ✉
City University of Hong Kong
xianzhao@cityu.edu.hk

Wanyu Wang
City University of Hong Kong
wanyuwang4-c@my.cityu.edu.hk

Yiqi Wang
National University of Defense Technology
yiq@nudt.edu.cn

Zitao Liu
Guangdong Institute of Smart Education, Jinan University
zitao.jerry.liu@gmail.com

ABSTRACT

Deep recommender systems typically involve numerous feature fields for users and items, with a large number of low-frequency features. These low-frequency features would reduce the prediction accuracy with large storage space due to their vast quantity and inadequate training. Some pioneering studies have explored embedding compression techniques to address this issue of the trade-off between storage space and model predictability. However, these methods have difficulty compacting the embedding of low-frequency features in various feature fields due to the high demand for human experience and computing resources during hyper-parameter searching. In this paper, we propose the AutoDPQ framework, which automatically compacts low-frequency feature embeddings for each feature field to an adaptive magnitude. Experimental results indicate that AutoDPQ can significantly reduce the parameter space while improving recommendation accuracy. Moreover, AutoDPQ is compatible with various deep CTR models by improving their performance significantly with high efficiency.

CCS CONCEPTS

• **Information systems** → **Recommender systems**.

KEYWORDS

Recommender Systems, AutoML, Compact Embedding

ACM Reference Format:
Xin Gan†, Yuhao Wang†, Xiangyu Zhao ✉, Wanyu Wang, Yiqi Wang, and Zitao Liu. 2023. AutoDPQ: Automated Differentiable Product Quantization for Embedding Compression. In *Proceedings of the 46th International ACM SIGIR Conference on Research and Development in Information Retrieval (SIGIR '23), July 23–27, 2023, Taipei, Taiwan.* ACM, New York, NY, USA, 5 pages. https://doi.org/10.1145/3539618.3591953

†Co-first author.
✉Corresponding author.

SIGIR '23, July 23–27, 2023, Taipei, Taiwan

ACM ISBN 978-1-4503-9408-6/23/07…$15.00
https://doi.org/10.1145/3539618.3591953

1 INTRODUCTION

Click-through rate (CTR) prediction is critical for industrial recommender systems and advertising platforms [19–23]. It is usually formulated as a binary classification problem and leverages a large number of sparse features to estimate whether a user will click the recommended item or not. In CTR models, an embedding table is utilized to map these sparse categorical features into dense latent space. However, this process suffers from two limitations: (i) insufficient training of low-frequency feature embeddings and (ii) low storage efficiency. Specifically, according to the Power Law distribution, the long-tailed low-frequency features take the majority of all features. Thus their insufficient training severely dampens the prediction accuracy. Moreover, the embedding component commonly accounts for more than 90% of the model parameters. Consequently, assigning each low-frequency feature a unique embedding vector would downgrade the storage and computation efficiency.

To address these two limitations, two straightforward solutions are adopted in commercial recommender systems (RS). The first is setting a field-wise "`Others`" feature and mapping all the low-frequency features into it for each feature field. However, it is challenging for a single "`Others`" embedding vector to accurately represent diversified information within different low-frequency features. Besides, the second solution is to compress the low-frequency features by embedding compression techniques [2, 7, 17]. For example, Deep Hash Embedding [7] uses a deep neural network to obtain embeddings on the fly without a heavyweight embedding table, whereas Double Hash [17] combines frequency hashing and double hashing techniques to reduce the model size. Notably, Differentiable Product Quantization (DPQ) [2] proposes a better end-to-end compression framework that compacts embedding to reduce a large number of parameters without sacrificing precision.

Nevertheless, considering the diversity of distributions among different feature fields, to search for their different magnitude of compacting requires a great deal of human effort and experience on feature engineering thus inefficient. Recently, Automated Machine Learning (AutoML) [1, 6, 18] prevails as a highly efficient technique aiming at automatically constructing the optimal machine learning frameworks for time-consuming and iterative real-world tasks [8–10, 12, 14, 16]. To this end, based on DPQ [2] algorithm and AutoML technique, we propose the AutoDPQ framework to automatically compress the embeddings of low-frequency features for different feature fields in this paper. Significantly, compared with other embedding compression method, here we focus more on the

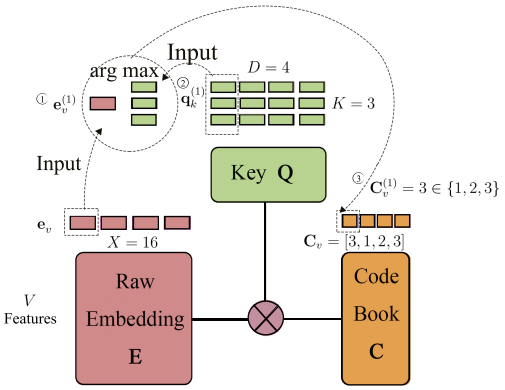

Figure 1: A toy example of the encoding process of the DPQ embedding framework. The input embedding vector $\mathbf{e}_v$ is encoded as [3, 1, 2, 3], meaning the first part of the compacted embedding is of the 3^{rd} class and so forth. At inference, only the Codebook $\mathbf{C} \in \{1, ..., K\}^{X \times D}$ and the Key Q are used to derive the output embedding vector.

trade-off between model size and accuracy rather than compression itself, which makes it meaningless to make comparisons. Our major contributions can be summarized as follows:

- We propose an adaptive field-wise embedding compression framework, AutoDPQ, for representation learning of low-frequency features in all feature fields;
- AutoDPQ leverages AutoML technique to automatically search and determine the optimal compression ratio for various feature fields, thereby saving high labor costs;
- Extensive experiments on two publicly available datasets demonstrate the effectiveness with high storage and computation efficiency of the AutoDPQ framework. Meanwhile, its compatibility with different CTR models are also validated.

2 PRELIMINARY

Differentiable Product Quantization (DPQ) [2] is a recent well-known end-to-end embedding compression approach, which is based on a highly compact encoding scheme called K-way D-dimensional (KD) encoding [3]. Figure 1 shows the encode-decode process of the DPQ framework, which is completely differentiable and hence can be trained in an end-to-end manner with the downstream tasks. Among the two approximations proposed in [2], we only focus on the softmax-based approximation (DPQ-SX) as suggested in the paper for experimental convenience. To be specific, V features in the vocabulary are firstly divided into K groups and the embedding dimension X is partitioned into D parts. Next, two steps are conducted: (i) the raw embedding vector of each feature v are split as $\mathbf{e}_v = [\mathbf{e}_v^{(1)}; \ldots; \mathbf{e}_v^{(D)}] \in \mathbb{R}^X$, where D is the number of subspace and $\mathbf{e}_v^{(i)} \in \mathbb{R}^{\frac{X}{D}}$ is the embedding vector of the i-th subspace. (ii) Then, in the i-th subspace, DPQ trains K learnable centroid vectors $\mathbf{q}_k^{(i)} \in \mathbb{R}^{\frac{X}{D}}$ to replace the original embedding, where $k = 1, \ldots, K$. These KD codes are generalized through a product quantization process in the next step. (iii) As an encoding process, for each feature v, the index of the nearest centroid in each subspace is recorded in the codebook $\mathbf{C}$:

$$\mathbf{C_v} = [\arg\max_k \langle \mathbf{e}_v^{(1)}, \mathbf{q}_k^{(1)} \rangle, \ldots, \arg\max_k \langle \mathbf{e}_v^{(D)}, \mathbf{q}_k^{(D)} \rangle]. \quad (1)$$

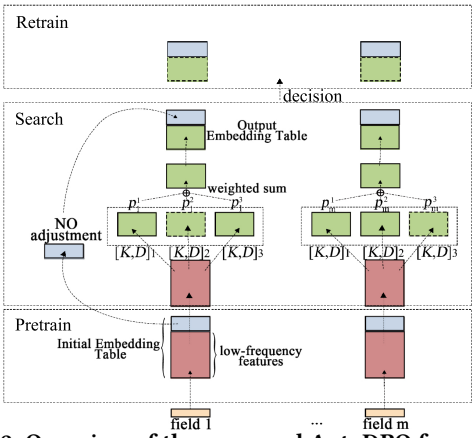

Figure 2: Overview of the proposed AutoDPQ framework.

Then, the decode function is similar to look-up table operation simply retrieving centroid embeddings based on the codebook $\mathbf{C}$, and then concatenates them as the output: $\widehat{\mathbf{e}_v} = [\mathbf{q}_{\mathbf{C}_v^{(1)}}^{(1)}, \ldots, \mathbf{q}_{\mathbf{C}_v^{(D)}}^{(D)}]$, where $\mathbf{C}_v^{(i)}$ is the i-th KD code computed in the codebook $\mathbf{C}_v$. Although this encode-decode process is not differentiable due to the arg min operation, a gumble-softmax approximation trick [5] can solve this problem and turn it into an end-to-end training process.

3 AUTODPQ FRAMEWORK

Low-frequency features are dominant in data collected in recommender systems, the existence of which leads to a serious storage inefficiency and dampens the prediction accuracy. Considering the diversity of the low-frequency feature distributions, applying the same and fixed compression ratio for different feature fields is sub-optimal. Among state-of-the-art embedding compression methods, the DPQ framework [2] is highly flexible in terms of compression ratio by simply adjusting the hyper-parameter K and D, i.e., the number of centroids and subspaces, respectively. However, it is still challenging to manually choose the appropriate hyper-parameter pairs of (K, D) for each feature field due to the huge search space. To solve this problem, we resort to AutoML and propose the AutoDPQ framework to implement field-wise embedding compacting automatically for the low-frequency features, achieving higher prediction accuracy of recommendation and high efficiency considering the storage space and training time.

To be specific, our AutoDPQ framework is composed of three steps: (i) **pretrain step** to accelerate training, (ii) **search step** to find the best decision through an end-to-end training manner to find a best trade-off of storage space and predicting performance, and (iii) **retrain step** to obtain the optimal parameters of the model. The overview of AutoDPQ framework is depicted in Figure 2 and the details are described as follows.

3.1 Pretrain Step

Most embedding compression methods typically include raw embedding learning and compact embedding training [2], which are time-consuming if they are simultaneously optimized from scratch. To accelerate the optimization process, we design a pretrain step before the search step. Considering the satisfactory performance of the primitive models, we can extract the predictive information directly from the primitive embeddings without any operation on the low-frequency features so as to accelerate training. Consequently,

Algorithm 1 Optimization Algorithm of Search Step

Require: users-item interaction features $(u_1, u_2, ..., u_M)$, the corresponding ground-truth label y, and the recommender system model h;

1: **while** not converge **do**
2: Sample a mini-batch of **validation data**
3: Update $\boldsymbol{\alpha}$ by descending $\nabla_{\boldsymbol{\alpha}} \mathcal{L}_{val}(h[\boldsymbol{W}^*(\boldsymbol{\alpha}), \boldsymbol{\alpha}])$ with approximation:
$\arg\min_{\boldsymbol{W}} \mathcal{L}_{train}(\boldsymbol{W}, \boldsymbol{\alpha}^*) \approx \boldsymbol{W} - \xi \nabla_{\boldsymbol{W}} \mathcal{L}_{train}(\boldsymbol{W}, \boldsymbol{\alpha})$, where ξ is the learning rate
4: Sample a mini-batch of **training data**
5: Make predictions $\hat{y} = h(\boldsymbol{W}, \boldsymbol{\alpha})$ via current $\boldsymbol{W}$ and $\boldsymbol{\alpha}$.
6: Update $\boldsymbol{W}$ by descending $\nabla_{\boldsymbol{W}} \mathcal{L}_{train}(h[\boldsymbol{W}, \boldsymbol{\alpha}])$
7: **end while**

we simply obtain the initial embedding table from the original RS model in the pretrain step, and then apply a compression transformation on it. This particular pretrain step can save a great deal of training time without sacrificing the prediction accuracy.

3.2 Search Step

The search step aims at searching for the best hyper-parameters of embedding compression. It is comprised of two modules: candidate choice union and hard-selection relaxation.

3.2.1 ***Unify all candidate choices.*** In each feature field, we allocate T hyper-parameter pairs of (K, D) as candidate choices. In order to train the model with respect to each candidate's selection and conduct a comparison of their performance, first we need to unify all the candidate choices. In this sense, the output embedding of each feature takes the form of a weighted sum to fuse and unify all the candidate choices.

However, it would be inappropriate to neglect the significant difference in the magnitude of embedding among different features in potential. Consequently, a BatchNorm transformation on the candidate compressed embeddings is applied to unify the order of the magnitude of each feature fields.

$$\widetilde{\mathbf{e}}_m^t \leftarrow \frac{\widetilde{\mathbf{e}}_m^t - \boldsymbol{\mu}_m^t}{\sqrt{{\sigma_m^t}^2 + \epsilon}}, \forall m \in [1, M], t \in [1, T], \tag{2}$$

where $\widetilde{\mathbf{e}}_m^t$ is the output embedding in terms of the candidate choice t for the feature field m, and $\boldsymbol{\mu}_m^t$ and ${\sigma_m^t}^2$ are the mean and variance of mini-batch for $\forall m \in [1, M], t \in [1, T]$. ϵ is a small constant added to stabilize numerical calculation when the variance is slight. After this BatchNorm transformation, we are able to compare the embeddings within the same order of magnitude.

3.2.2 ***Hard-selection relaxation.*** Compared with a weighted sum of all candidate embeddings, deriving the best decision with hard selection can reduce the storage space. Therefore, we approximate this hard selection by the differentiable function Gumbel-Softmax [5], where continuous distribution can be smoothly annealed into a categorical distribution. Its parameter gradients can be easily computed via the reparameterization trick. Hence, for the m-th feature field, a learnable structural weight α_m^t is adopted to derive the probability p_m^t of being selected for the t-th candidate choice through the Gumbel-Softmax function. We have $\sum_{t=1}^{T} p_m^t = 1$, where T is the number of candidate choices.

The output embedding vector of search step is the weighted sum of all the candidate output embeddings:

$$\mathbf{e}_m = \sum_{t=1}^{T} p_m^t \cdot \widetilde{\mathbf{e}}_m^t, \quad \forall m \in [1, M]. \tag{3}$$

After obtaining the final feature embeddings, since we only deal with the embedding without any assumption on feature interaction functions, various deep CTR models (such as DeepFM [4], PNN [13], DCN [15]) can be applied to conduct CTR prediction task. We will further validate this through experiment in Section 4.2.

As for the optimization process in this step, the parameters of AutoDPQ can be divided into two classes: the structural weight $\boldsymbol{\alpha}$ and other model parameters $\boldsymbol{W}$. Specifically, $\boldsymbol{W}$ is updated to maximize model training predicting accuracy, while the weight $\boldsymbol{\alpha}$ is trained to determine the choice of (K, D) considering the trade-off between the compression magnitude and generalization capability in terms of various low-frequency features in different feature fields. Hence, similar to the Differentiable Architecture Search (DARTS) [11] technique , $\boldsymbol{\alpha}$ and $\boldsymbol{W}$ are alternately updated on the training and validation sets following:

$$\begin{aligned} &\min_{\boldsymbol{\alpha}} \mathcal{L}_{val}(\mathbf{W}^*(\boldsymbol{\alpha}), \boldsymbol{\alpha}) \\ &s.t. \mathbf{W}^*(\boldsymbol{\alpha}) = \arg\min_{\mathbf{W}} \mathcal{L}_{train}(\mathbf{W}, \boldsymbol{\alpha}^*) \end{aligned} \tag{4}$$

By alternative training, the weight $\boldsymbol{\alpha}$ can be updated correctly in terms of its natural function. In summary, the overall optimization algorithm of the search step is detailed in Algorithm 1.

3.3 Retrain Step

Retrain step aims at selecting the optimal compression choice for each feature field. Firstly, the final decision of (K, D) is derived for each field by selecting the largest weight. Then these selected hyper-parameter pairs are applied to make field-wise embedding compression, and obtain the final compact embeddings after epochs of retrain in the traditional manner, fixing the choice of (K, D) in different fields determined in search step and fine-tune all the other parameters including RS through back-propagation. Notably, the BatchNorm transformation is discarded in the retrain step without comparing different choices of (K, D) in different feature fields.

4 EXPERIMENT

4.1 Experiment Settings

4.1.1 ***Dataset & Evaluation metrics.*** In our experiments, we evaluate our framework on the benchmark datasets Movielens1M[1] containing 1 million anonymous ratings of movies with 8 features fields used and Avazu[2] composed of 40 million samples with 22 feature fields. Samples with a rating of less than 3 are regarded as negative ones in Movielens1M. The training/validation/test set is split into 8:1:1 in all the experiments. Following previous work [4, 15], we use popular metrics **AUC** (Area Under ROC) and **Logloss** (cross entropy) to evaluate the performance. It is noteworthy that a higher AUC or lower Logloss value at **0.001-level** is regarded as significant for the CTR prediction task [4]. Besides, our framework is easy to be applied with different CTR models, and we show the compatibility with DeepFM [4], DCN [15], and PNN [13].

4.1.2 ***Implementation details.*** In Movielens1M, the frequency threshold of judging low-frequency features is set to be 40. Similarly, in Avazu, the threshold is set at 7. For parameters in KD encoding, we set τ_{KD} = 1e-4 as a small constant. The temperature parameter τ used for deriving the best choice is set annealing as

[1] https://grouplens.org/datasets/movielens/
[2] https://www.kaggle.com/competitions/avazu-ctr-prediction/data

Table 1: Performance comparison of different hyper-parameter searching strategy.

Dataset	Model	Metrics	Searching strategy					Rel.Improv.
			FE	Others	Grid	Random	AutoDPQ	
Movielens1M	DeepFM	AUC ↑	0.8079	0.8098	0.8111	0.8112	**0.8147***	**0.43%**
		Logloss ↓	0.5239	0.5216	0.5251	0.5193	**0.5141***	**1.00%**
		Space ↓	216k	171k	194k	192k	190k	/
Avazu	DeepFM	AUC ↑	0.7849	0.7828	0.7846	0.7879	**0.7894***	**0.19%**
		Logloss ↓	0.3783	0.3781	0.3784	0.3751	**0.3732***	**0.51%**
		Space ↓	151M	16M	31M	47M	49M	/

"*" indicates the statistically significant improvements (i.e., two-sided t-test with $p < 0.05$) over the best baseline.

Table 2: Compatibility of AutoDPQ

Model	Metrics	Searching strategy		
		Others	Random	AutoDPQ
DeepFM [4]	AUC ↑	0.8098	0.8112	**0.8147***
	Logloss ↓	0.5216	0.5193	**0.5141***
	Space ↓	171k	192k	190k
PNN [13]	AUC ↑	0.8072	0.8116	**0.8144***
	Logloss ↓	0.5232	0.5197	**0.5153***
	Space ↓	171k	193k	189k
DCN [15]	AUC ↑	0.8105	0.8119	**0.8140***
	Logloss ↓	0.5193	0.5208	**0.5149***
	Space ↓	171k	195k	191k

"*" indicates the statistically significant improvements (i.e., two-sided t-test with $p < 0.05$) over the best baseline.

$\tau = max(0.01, 1 - s \cdot 5e\text{-}5)$, where s is the training epochs. Moreover, for each feature field in Movielens1M and Avazu, we select the best decision from candidate hyper-parameter pairs $K = (8, 16, 32)$, $D = (2, 4, 8)$, and $K = (8, 16)$, $D = (2, 4)$ respectively. Under these settings, the search space would be 9^8 and 4^{22}, which is quite large if random search is applied. Other parameters in terms of DeepFM, PNN, and DCN models are: embedding dimension=16, hidden layers of MLP=(16, 16), learning rate=1e-3, and batch size=2048. All the experiment results are the average of three runs.

4.2 CTR Prediction Performance

In our experiment, we firstly compare the performance of different searching strategies of hyper-parameters on different datasets. Baseline 1 (FE) is to include all the features without any compression operation. Baseline 2 (Others) is the common way to combine low-frequency features into one "`Others`" feature and share the same embedding vector. Baseline 3 (Grid) uses Grid Search method to find the optimal global (K, D) pair for all feature fields. Baseline 4 (Random) uses Random Search method to select different (K, D) pairs randomly for each feature field.

From the Table 1, we can see that compared with FE, the Others method can significantly reduce the storage space and increase the predicting accuracy on Movielens1M dataset, but it achieves lower AUC on Avazu dataset, indicating that the Others method is not always ideal for all circumstances. By using grid search to adjust the DPQ framework for embedding compression, the predicting accuracy is improved with great reduction of storage space. Instead of using an end-to-end training manner, Random Searching it ends up with finding a sub-optimal compression decision. By contrast, our AutoDPQ framework outperforms the other four baselines in terms of AUC and logloss, which indicates that AutoDPQ can achieve better performance and save a lot of storage space.

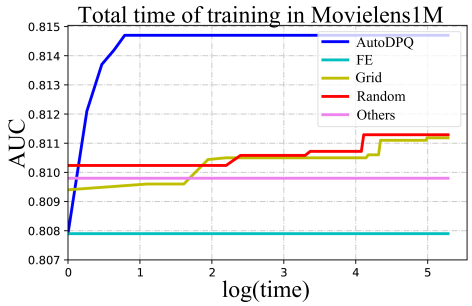

Figure 3: Time consuming of each searching strategy. The unit of log(time) is the training time of FE.

Then we applied our AutoDPQ framework on various deep CTR models, i.e., DeepFM[4], PNN[13], and DCN[15]. From Table 2 we can see that our AutoDPQ framework improves the prediction accuracy of deep CTR models consistently and requires less storage space than random search.

4.3 Computation Efficiency Analysis

Apart from the significant improvement of the storage space as well as the predicting accuracy, our AutoDPQ is also efficient in terms of search time. As shown in Figure 3, it takes Random Searching much longer time to search for the hyper-parameter pair (K, D). Meanwhile, our AutoDPQ spends nearly two times longer than simply training the original model and Others strategy. We can see that given limited time, both Grid Search and Random Search strategies get lower AUC than our AutoDPQ framework while costing much more time. In conclusion, our AutoDPQ framework is able to find the best hyper-parameter pairs with high efficiency.

5 CONCLUSION

In this paper, we propose the AutoDPQ framework to automatically conduct adaptive field-wise embedding compression. The use of AutoML enables AutoDPQ to achieve better performance on CTR prediction task and possess high storage and computation efficiency. Besides, AutoDPQ is highly compatible and it can be easily applied to any RS model with the embedding component. Future work can be done in applying this field-wise compression with different compact embedding methods by AutoML.

ACKNOWLEDGEMENT

This work was supported by APRC - CityU New Research Initiatives (No.9610565, Start-up Grant for New Faculty of CityU), SIRG - CityU Strategic Interdisciplinary Research Grant (No.7020046, No.7020074), CityU - HKIDS Early Career Research Grant (No.9360163), Huawei (Huawei Innovation Research Program) and Ant Group (CCF-Ant Research Fund, Ant Group Research Fund).

REFERENCES

[1] Bo Chen, Xiangyu Zhao, Yejing Wang, Wenqi Fan, Huifeng Guo, and Ruiming Tang. 2022. Automated Machine Learning for Deep Recommender Systems: A Survey. *arXiv preprint arXiv:2204.01390* (2022).

[2] Ting Chen, Lala Li, and Yizhou Sun. 2020. Differentiable product quantization for end-to-end embedding compression. In *ICML*. 1617–1626.

[3] Ting Chen, Martin Renqiang Min, and Yizhou Sun. 2018. Learning k-way d-dimensional discrete codes for compact embedding representations. In *ICML*. 854–863.

[4] Huifeng Guo, Ruiming Tang, Yunming Ye, Zhenguo Li, and Xiuqiang He. 2017. DeepFM: a factorization-machine based neural network for CTR prediction. In *IJCAI*.

[5] Eric Jang, Shixiang Gu, and Ben Poole. 2016. Categorical reparameterization with gumbel-softmax. *arXiv preprint arXiv:1611.01144* (2016).

[6] Wei Jin, Xiaorui Liu, Xiangyu Zhao, Yao Ma, Neil Shah, and Jiliang Tang. 2022. Automated Self-Supervised Learning for Graphs. In *10th International Conference on Learning Representations (ICLR 2022)*.

[7] Wang-Cheng Kang, Derek Zhiyuan Cheng, Tiansheng Yao, Xinyang Yi, Ting Chen, Lichan Hong, and Ed H Chi. 2021. Learning to embed categorical features without embedding tables for recommendation. In *Proceedings of the 27th ACM SIGKDD Conference on Knowledge Discovery & Data Mining*. 840–850.

[8] Muyang Li, Zijian Zhang, Xiangyu Zhao, Wanyu Wang, Minghao Zhao, Runze Wu, and Ruocheng Guo. 2023. AutoMLP: Automated MLP for Sequential Recommendations. In *Proceedings of the Web Conference 2023*.

[9] 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.

[10] Weilin Lin, Xiangyu Zhao, Yejing Wang, Yuanshao Zhu, and Wanyu Wang. 2023. AutoDenoise: Automatic Data Instance Denoising for Recommendations. In *Proceedings of the Web Conference 2023*.

[11] Hanxiao Liu, Karen Simonyan, and Yiming Yang. 2018. Darts: Differentiable architecture search. *arXiv preprint arXiv:1806.09055* (2018).

[12] Haochen Liu, Xiangyu Zhao, Chong Wang, Xiaobing Liu, and Jiliang Tang. 2020. Automated Embedding Size Search in Deep Recommender Systems. In *Proceedings of the 43rd International ACM SIGIR Conference on Research and Development in Information Retrieval*. 2307–2316.

[13] Yanru Qu, Han Cai, Kan Ren, Weinan Zhang, Yong Yu, Ying Wen, and Jun Wang. 2016. Product-based neural networks for user response prediction. In *ICDM*. IEEE, 1149–1154.

[14] Fengyi Song, Bo Chen, Xiangyu Zhao, Huifeng Guo, and Ruiming Tang. 2022. AutoAssign: Automatic Shared Embedding Assignment in Streaming Recommendation. In *2022 IEEE International Conference on Data Mining (ICDM)*. IEEE, 458–467.

[15] Ruoxi Wang, Bin Fu, Gang Fu, and Mingliang Wang. 2017. Deep & cross network for ad click predictions. In *ADKDD*. 1–7.

[16] 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.

[17] Caojin Zhang, Yicun Liu, Yuanpu Xie, Sofia Ira Ktena, Alykhan Tejani, Akshay Gupta, Pranay Kumar Myana, Deepak Dilipkumar, Suvadip Paul, Ikuhiro Ihara, et al. 2020. Model size reduction using frequency based double hashing for recommender systems. In *RecSys*. 521–526.

[18] Xiangyu Zhao. 2022. Adaptive and automated deep recommender systems. *ACM SIGWEB Newsletter* Spring (2022), 1–4.

[19] Xiangyu Zhao, Changsheng Gu, Haoshenglun Zhang, Xiwang Yang, Xiaobing Liu, Hui Liu, and Jiliang Tang. 2021. DEAR: Deep Reinforcement Learning for Online Advertising Impression in Recommender Systems. In *Proceedings of the AAAI Conference on Artificial Intelligence*, Vol. 35. 750–758.

[20] 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 International Conference on Knowledge Discovery & Data Mining*. 3959–3967.

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

[22] 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.

[23] Chenxu Zhu, Bo Chen, Huifeng Guo, Hang Xu, Xiangyang Li, Xiangyu Zhao, Weinan Zhang, Yong Yu, Ruiming Tang, Wenwu Ou, et al. 2023. AutoGen: An Automated Dynamic Model Generation Framework for Recommender System. In *Proceedings of the 16th ACM International Conference on Web Search and Data Mining*. 445–453.