AutoTower: Efficient Neural Architecture Search for Two-Tower Recommendation

Anonymous

ABSTRACT

Recommender systems are ubiquitous in people's daily lives. A common practice for large-scale recommendations is the retrieval-and-ranking method. The retrieval stage that retrieves a small fraction of related items from a large corpus plays a critical role in the performance of recommender system. Recently, two-tower models have gained ever-increasing interests in large-scale retrieval. However, most of existing architectures of two-tower models are based on deep neural networks, lacking the exploration of more complex and novel architectures. Moreover, the two-tower architecture that takes the available computation resources and the data characteristics into account should be designed. Inspired by neural architecture search (NAS) in automated machine learning, we propose an effective and efficient NAS-based method called AutoTower for the two-tower architecture search. First, by revisiting existing two-tower architectures, we design a simple yet general and expressive search space based on the single-path paradigm. Furthermore, we propose an efficient one-shot search strategy, which consists of a weight-sharing-based supernet training stage and a complexity-aware evolutionary search stage. Extensive experiments on four real-world datasets demonstrate that AutoTower can consistently outperform human-crafted models with much fewer parameters and FLOPs. Moreover, the architectures searched by AutoTower are transferable. Anonymous code of AutoTower is now available at https://github.com/autotower/AutoTower.

CCS CONCEPTS

• **Information systems** → **Recommender systems**; • **Computing methodologies** → **Machine learning**; **Search methodologies**.

KEYWORDS

Two-tower model, Recommender systems, Neural architecture search, Large-scale retrieval

ACM Reference Format:

Anonymous. 2018. AutoTower: Efficient Neural Architecture Search for Two-Tower Recommendation. In *Proceedings of Make sure to enter the correct conference title from your rights confirmation emai (Conference acronym 'XX).* ACM, New York, NY, USA, 10 pages. https://doi.org/XXXXXXX.XXXXXXX

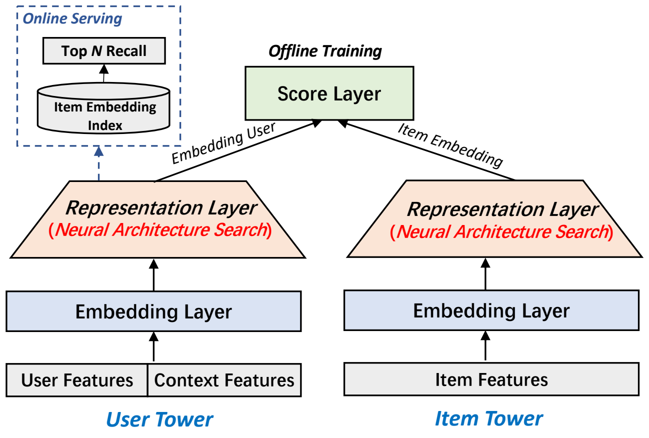


Figure 1: General framework of two-tower recommendation.

1 INTRODUCTION

Nowadays, recommender systems, which provide accurate and useful content to address the information overload problem, are ubiquitous in people's daily lives[3, 41]. One of the most critical challenges in building real-world recommender systems is to accurately score millions to billions of items in real time [41]. A common practice for large-scale recommendation is the two-stage method where the retrieval stage first retrieves a small fraction of related items from a large corpus as candidate items, and then the ranking stage re-ranks these candidate items based on clicks or user ratings. The retrieval stage plays a fundamental role in recommender systems [19], not only because it allows recommender systems to scale to billions of users and items, but also because the quality of retrieved candidate items is critical for the performance of the whole recommender system. In this work, we mainly focus on the retrieval stage.

Recently, the two-tower model consisting of a user tower and an item tower attains ever-increasing interests and has become popular in large-scale retrieval. Figure 1 shows the general framework of the two-tower model. The user tower and the item tower are responsible for learning the user and item representations, respectively. The preference of a user-item pair can be scored as the inner product of user representation and item representation. During online serving, the user features and the current context features are first fed into the user tower. Then, the approximate nearest neighbor (ANN) search such as Faiss [17] is employed to retrieve candidate items by calculating the similarity between the online-calculated user embedding and the offline-stored item embeddings.

As shown in Figure 1, the architecture of the representation layer in the two-tower model plays a critical role in the quality of representation learning. In the large-scale retrieval scenarios, considering the training efficiency and the online serving latency, existing two-tower models such as YoutubeDNN [4] and GoogleDNN [41] apply deep neural networks (DNNs) as the architecture of the representation layer. In practice, such relatively simple architecture design does not yet fulfill the following requirements:

(1) **Better model performance.** To enhance the representation quality, it is well worth exploring more complex and novel architectures beyond DNNs. For example, diverse representation operations (e.g., self-attention [36]) can be flexibly combined as the architecture of the representation layer.
(2) **Resource-specific architecture design.** Besides the model performance, the architecture complexity (e.g., FLOPs or number of parameters) of the two-tower model should be constrained to match the available computation resources. Especially for the user tower, when the online computation resource is limited, the architectures with low complexity should be designed.
(3) **Data-specific architecture design.** Since the user and item features usually have different characteristics (i.e., the number of instances, the number of features, the degree of sparseness), there is no single architecture that can perform well on different datasets so the data-specific architectures should be designed.

As the core of AutoML (Automated Machine Learning), Neural Architecture Search (NAS) has achieved great success in computer vision tasks [1, 46]. It aims to search for the optimal deep architecture in a data-driven manner without human intervention. Inspired by the idea of NAS, we propose to utilize the NAS method to search for resource-specific and data-specific architectures for the two-tower model. However, there exist two challenges in the design of NAS-based tower architecture.

First, designing a general and effective search space that can cover both the existing and unexplored tower architectures is challenging. Also, to improve the search efficiency for large-scale retrieval, the search space cannot be too large. Moreover, how to capture the user behavior features needs to be considered in the search space. Second, we should design a search strategy that can handle large-scale corpus efficiently, and support multi-objective (i.e., model performance and architecture complexity).

To address the two challenges, we propose an effective NAS-based method called AutoTower for the two-tower architecture search. First, by revisiting existing two-tower architectures, we design a simple, yet general and expressive search space based on the single-path with shortcut connections. Specifically, the overall search space is composed of a pooling block and a stack of choice blocks. The pooling block is used to capture the user behavior features and the choice block is used for representation learning. Multiple choice blocks are stacked to enhance the representation ability. Each block contains multiple candidate operations and only one operation will be activated.

Furthermore, based on the single-path search space, we propose an efficient one-shot search strategy, which consists of a supernet training stage and an architecture search stage. In the supernet training stage, we first leverage uniform sampling to sample child architectures from the weight-sharing supernet. Then, we employ mini-batch softmax to improve the training efficiency of the child architecture. In the architecture search stage, we utilize a model complexity-aware evolutionary algorithm to search optimal architectures by directly evaluating candidate architectures on the validation dataset, which can greatly improve the search efficiency.

In summary, our contributions can be summarized as follows:

- We approach the problem of two-tower recommendation from the perspective of neural architecture search and propose a NAS-based method called AutoTower to search for appropriate two-tower architectures under the model complexity constraints. To our best knowledge, we are the first to utilize NAS to search for two-tower models.
- First, we design a simple, yet general and expressive search space base on the single-path with shortcut connections. The overall search space is composed of a pooling block and a stack of choice blocks. Only one operation can be activated on each block.
- Then, we propose an efficient one-shot search strategy based on the single-path search space, which contains a weight-sharing-based supernet training stage and a complexity-aware evolutionary search stage.
- Extensive experiments on four real-world datasets demonstrate that AutoTower can consistently outperform human-crafted models by **28.95%-37.22%** average relative improvement of Recall@50, and **33.01%-43.20%** average relative improvement of NDCG@50 with **50.55%-88.03%** average decrease of parameters and **8.00%-70.06%** average decrease of FLOPs. Moreover, the architectures searched by AutoTower are transferable. The analysis of the searched architectures provides insightful guidance to help human experts to design effective two-tower models, e.g., simple models also can achieve better performance.

2 RELATED WORK

2.1 Deep Recommender System

With the success of deep learning in computer vision and natural language processing [18], developing deep recommendation models has become a research hotspot. NCF [13] and NFM [11] utilize neural networks to extend collaborative filtering (CF) and factorization machine (FM). Wide&Deep [3] and DeepFM [8] jointly train a wide model and a deep model. xDeepFM [21], PNN [30], DCN [37], and AutoInt [35] design specific structures to model high-order feature interactions. FiBiNET [16] utilizes Squeeze-Excitation network (SENET) [14] to learn the importance of features. DIN [44], DIEN [43], and DSIN [5] focus on capturing user interest from user behaviors. Such methods are mainly used for the ranking stage, especially for CTR prediction.

For the retrieval stage, CF is a widely used method. Item-based CF [23, 33] precomputes the item-to-item similarity matrix, items similar to clicked ones are recommended to the user. NAIS [12], a deep learning method for item-based CF, uses an attention mechanism to distinguish different importance of the user behavior. Unlike item-based CF, another line follows the inner product paradigm like matrix factorization, which is known as embedding-based methods.

One typical embedding-based method is the two-tower model, which has received ever-increasing interests in large-scale retrieval. DSSM [15] is a pioneering work that applies two-tower model to web search. YoutubeDNN [4] and GoogleDNN [41] use two

DNNs to model user features and item features respectively and formulate the learning problem as an extreme classification problem. MIND [19] uses capsule network to capture multiple interests from user behaviors for better modeling user representation. The works in [7, 42, 45] focus on the retrieval on entire corpus to eliminate the mismatch caused by ANN. In this paper, we follow the two-tower model and propose a novel NAS approach to find optimal architectures for the two-tower model.

2.2 Neural Architecture Search

Neural architecture search (NAS) aims to automatically design optimal neural networks in a data-driven manner. Recently, many NAS methods such as evolutionary-based [31], reinforcement-learning-based [29], and gradient-based [26] methods have been proposed. As a pioneering NAS work, [46] uses an RNN controller to generate architectures and employs the policy gradient algorithm to train the controller. However, the search efficiency of this approach is very low. To improved the search efficiency, the cell-based micro search space has been proposed [47]. Moreover, ENAS [29] employs a weight-sharing mechanism to further improve search efficiency. DARTS [26] and NASP [40] take a differentiable approach by relaxing the discrete search space to be continuous and applying bi-level optimization to optimize architecture parameters and model weights alternatively. SPOS [9] first trains the supernet with randomly sampled architectures, then searches the optimal architecture by an evolutionary algorithm.

2.3 AutoML for Recommender Systems

With the success of NAS, there is an increasing interest in applying NAS to recommender systems. AutoFIS [25] and AutoGroup [24] automatically identify important cross features in a differentiable manner. AutoCTR [34] modularizes representative feature interactions as virtual building blocks and performs evolutionary architecture exploration for feature interaction architecture. AutoPI [27] designs a comprehensive search space for feature interaction architecture and adopts a gradient-based search strategy. SIF [40] exploits the NASP method to search for proper interaction functions for CF. AutoCF [6], an extension of SIF, splits CF methods into disjoint stages, and utilizes random search to find optimal data-specific CF models.

3 THE PROPOSED METHOD: AUTOTOWER

3.1 Problem Definition

3.1.1 A Unified Framework of Two-Tower Model. As shown in Figure 1, we abstract existing two-tower models into a unified framework. In general, the two-tower model is composed of a user tower and an item tower. Each tower can be divided into three layers, i.e., the embedding layer, the representation layer, and the score layer. During offline training, the two-tower model is trained by specific loss function (e.g., cross-entropy loss or log-loss) according to the outputs of the score layer. After the training process, the learned representations of all items are stored offline. During online serving, when a user query arrives, the user tower calculates the user's representation in real-time. Then, the ANN search is used to retrieve the most relevant items as the retrieval result based on the score function. The details of the three layers are as follows:

- **Embedding Layer.** Following the common paradigm of the embedding layer in recommender system, the embedding lookup tables are used to transform sparse features to dense embedding vectors. The embeddings of all user (or item) features are concatenated as the output of the embedding layer in the user (or item) tower. We also take the user behavior sequence into consideration, which contains features of items interacted with before.
- **Representation Layer.** As the core of the two-tower model, the representation layer encodes the input feature as a low-dimensional representation based on the deep neural architecture. Let e_u and e_v denote the user representation and the item representation respectively. Typical architectures of user tower and item tower are DNNs. In this paper, We utilize the NAS-based method to search for novel architectures.
- **Score Layer.** The user representation e_u and the item representation e_v then go through a score layer to obtain a similarity score (e.g., inner product $\langle e_u, e_v\rangle$ or Euclidean distance $\|e_u - e_v\|$). Usually e_u and e_v will be normalized before output, thereby the Euclidean distance and inner product are equivalent. With the similarity score, losses can be calculated according to the loss function and the two-tower model can be trained through gradient descent.

3.1.2 Formulating as An AutoML Problem. Users and items are represented by feature vectors $\{\mathbf{u}_i\}_{i=1}^N$ and $\{\mathbf{v}_j\}_{j=1}^M$, where N is the number of the users and M is the number of items. For each user $\mathbf{u}_i$, we have a sequence of user historical behaviors $\mathbf{h}_i = (\mathbf{h}_i^{(1)}, \mathbf{h}_i^{(2)}, \cdots, \mathbf{h}_i^{(l)})$, sorted by occurrence time. $\mathbf{h}_i^{(t)}$ records the t^{th} item interacted by user $\mathbf{u}_i$. Let t_u and t_v represent the user tower and the item tower. The corresponding model parameters are θ_u and θ_v. Let $t_u(\mathbf{u}, \mathbf{h}, \theta_u)$ and $t_v(\mathbf{v}, \theta_v)$ denote the user and item representations generated by user tower and item tower, respectively. In this work, the inner product is used as the score function, namely

$$s(\mathbf{u}, \mathbf{h}, \mathbf{v}) = \langle t_u(\mathbf{u}, \mathbf{h}, \theta_u), t_v(\mathbf{v}, \theta_v)\rangle \tag{1}$$

The goal of AutoML is to design optimal architectures for user tower and item tower given the training dataset $\mathcal{S}_{\text{train}}$ and the validation dataset $\mathcal{S}_{\text{val}}$, denoted by

$$\mathcal{S}_{\text{train}} := \{(\mathbf{u}_i, \mathbf{h}_i, \mathbf{v}_i, r_i)\}_{i=1}^T$$
$$\mathcal{S}_{\text{val}} := \{(\mathbf{u}_i, \mathbf{h}_i, \mathbf{v}_i, r_i)\}_{i=1}^V$$

where $(\mathbf{u}_i, \mathbf{h}_i, \mathbf{v}_i)$ denotes the tuple of observed user $\mathbf{u}_i$, user behavior sequence $\mathbf{h_i}$, and item $\mathbf{v}_i$. $r_i \in \mathbb{R}$ is the associated reward for each tuple. T and V denote the number of instances in $\mathcal{S}_{\text{train}}$ and $\mathcal{S}_{\text{val}}$, respectively. The architecture search of the two-tower model can be formulated as a bi-level optimization problem under the model complexity constraints.

$$\begin{aligned} (t_u^*, t_v^*) &= \underset{t_u \in \mathcal{T}_u, t_v \in \mathcal{T}_v}{\arg\min} \mathcal{L}_{\text{val}}(t_u, t_v, \theta_u^*, \theta_v^*, \mathcal{S}_{\text{val}}) \\ \text{s.t.} \quad (\theta_u^*, \theta_v^*) &= \underset{(\theta_u, \theta_v)}{\arg\min} \mathcal{L}_{\text{train}}(t_u, t_v, \theta_u, \theta_v, \mathcal{S}_{\text{train}}), \\ C(t_u) &\le c_u, \\ C(t_v) &\le c_v \end{aligned} \tag{2}$$

where $\mathcal{T}_u$ and $\mathcal{T}_v$ are architecture search spaces of the user tower and the item tower, respectively. Let $\mathcal{L}_{\text{train}}$ and $\mathcal{L}_{\text{val}}$ denote the

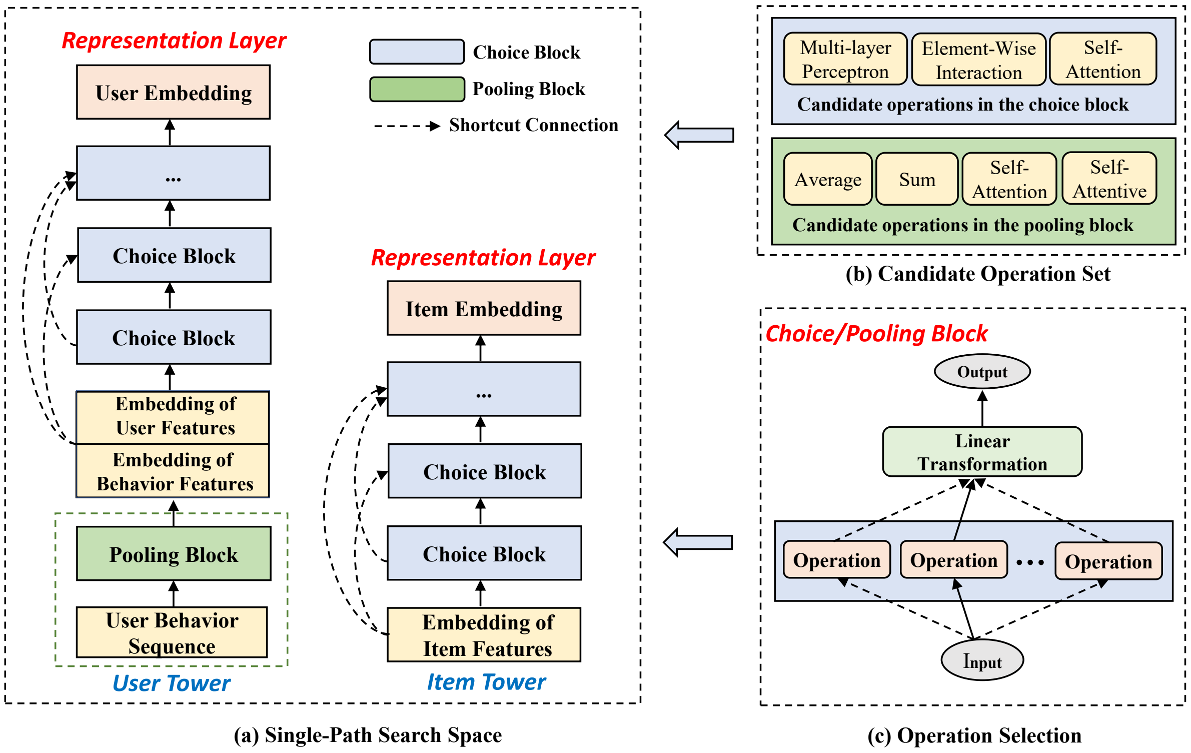


Figure 2: (a) Overview of the search space in AutoTower. (b) Candidate operation set in the pooling block and choice block. (c) Discrete operation selection.

Table 1: Details of choice blocks.

Operation in Choice Block	Number of Block Types	Hyperparameter of Each Block	Operation on Inputs
Multi-layer Perceptron (MLP)	7	*The number of units in the hidden layer:* **16, 32, 64, 128, 256, 512, 1024**	Concatenation
Element-Wise Interaction (EW)	5	*Element-wise operations:* **sum, average, inner product, max, min**	Element-wise
Self-Attention (SA)	4	*The number of head:* **1, 2, 3, 4**	Concatenation

losses on the training and validation datasets. C represents the complexity of architecture. c_u and c_v are complexity constraints for the user tower and the item tower. In general, we can use FLOPs or number of parameters to evaluate the model complexity [39].

3.2 Search Space Design

3.2.1 Design Criterion. In large-scale retrieval scenarios, the design of the architecture search space of the two-tower model should satisfy two criteria as far as possible.

- **Generality and Expressiveness.** The search space can cover not only the existing tower architectures but also the potential architectures not yet explored.
- **Simplicity.** In practice, to improve the search efficiency of the large-scale recommendation, the search space should be simple and compact.

To meet the two criteria, we propose a simple yet effective search space in this paper.

3.2.2 Single-Path Based Space Design. As shown in Figure 2, the overall search space can be represented by single-path with shortcut connections, which consists of a pooling block and a stack of choice blocks. Both the pooling block and the choice block are composed of multiple candidate operations. In each block, only one operation can be selected and activated. The pooling block is used to capture the user behavior features from the sequence of interacted items. The choice block is used for representation learning.

In the pooling block, we choose four commonly used pooling operations to model the user behavior sequence. Specifically,

- **Average Pooling** over the sequence length.
- **Sum Pooling** over the sequence length.
- **Self-Attention Pooling**. The user behavior sequence first goes through a standard multi-head self-attention layer [36], then the average pooling is adopted over the sequence length. We only use one head here.
- **Self-Attentive Pooling**. Following the self-attentive [22], we use the attention mechanism to obtain a weighted sum

over the sequence, i.e.,

$$\mathbf{a} = \text{softmax}(\mathbf{w}_2^\top \tanh(\mathbf{W}_1\mathbf{h}))^\top,$$
$$\mathbf{u}_h = \mathbf{h}\mathbf{a}$$

where $\mathbf{w}_2$ and $\mathbf{W}_1$ are trainable parameters. The vector $\mathbf{a}$ represents the attention weight of user behaviors, and $\mathbf{u}_h$ is the representation after pooling.

In the choice block, the commonly used representation operations such as Multi-Layer Perceptron (MLP), Element-Wise Interaction (EW), and Self-Attention (SA) are employed. Table 1 shows the details of the representation operations. One operation can correspond to multiple block types with different hyperparameters, e.g., different numbers of hidden units in MLP represent different block types. Also, other representation operations such as SENET Layer [14], Product Layer [30] can be easily integrated into the search space.

3.2.3 Shortcut Connection. Moreover, we further introduce a shortcut connection technique [10, 32], which enables the low-level feature representation directly flowing to high-level choice blocks. The shortcut connection can help low-level block to obtain more information from the final user-item interaction during back-propagation. Each choice block takes the outputs of previous blocks and original input features as the input. We further perform dimension alignment for each choice block (except the last one) to assist the aggregation of multiple inputs. As shown in Figure 2(b), the dimension of the output of each choice block is transformed to the same size D_{blk} by an additional linear transformation. The output of the last choice block is viewed as the user representation or item representation.

The user tower and the item tower have the same architecture search space (except the pooling block) shown in Figure 2. However, considering the differences in application scenarios and input features, the architectural complexities and numbers of choice blocks of user tower and item tower can be different.

3.2.4 Search Space Size Analysis. Given the single-path search space, the search objective is the types of all choice blocks, i.e., the operation activated in each choice block or pooling block. Correspondingly, the size of search space is determined by the number of pooling types $|\mathcal{P}|$, the number of block types $|\mathcal{B}|$, the number of choice blocks in the user tower N_u, and the number of choice blocks in the item tower N_v. Thus, the space size can be calculated as $|\mathcal{P}| \cdot |\mathcal{B}|^{N_u+N_v}$. In this work, we use 4 types of pooling, 16 different types of choice blocks, and N_u and N_v are from 3 to 7. The size of search space is at least 6.72×10^7. Although the proposed search space is relatively simple, the space size is still substantial. To improve search efficiency within such a search space, we further propose an efficient one-shot search strategy.

3.3 One-Shot Search Strategy

3.3.1 Overview of Search Strategy. The proposed single-path based search space can be viewed as a supernet. In each iteration of the search process, a child architecture will be sampled from the supernet. Training the two-tower model from scratch incurs huge computation overhead especially for large-scale retrieval. To address the issue, we propose a one-shot search strategy, which contains a supernet training stage and an architecture search stage. The supernet training stage trains the supernet through a weight-sharing mechanism. By this way, we can avoid training each sampled child architecture from scratch. After the supernet is well-trained, we utilize a model complexity-aware evolutionary algorithm to search optimal architectures by directly computing the validation loss without training. Finally, the searched candidate architectures are re-trained on the full training dataset to obtain the best architecture.

3.3.2 Supernet Training Based on Weight-Sharing. The supernet training algorithm is described in Algorithm 1. In the beginning, the weights of the supernet are randomly initialized. Then, the overall supernet training is composed of many training epochs. In each epoch, we first randomly sample architectures from the supernet for both user tower and item tower. Inspired by [9], we first leverage uniform sampling to sample as many child architectures as possible. Then, the sampled child architectures are trained in batches.

To improve the training efficiency, we adopt two optimization techniques: weight-sharing and mini-batch softmax. First, we force all children architectures to share common weights to eschew training each child architecture from scratch to convergence. When the training of the child architecture is finished, we update the corresponding weights of the supernet.

Second, following previous works [4, 41], we treat the large-scale retrieval as an extreme classification problem and employ the mini-batch softmax as loss function. Given a mini-batch B user-item pairs $\{(\mathbf{u}_i, \mathbf{h}_i, \mathbf{v}_i, r_i)\}_{i=1}^{B}$, for each $i \in [B]$, we have:

$$P_B^c(\mathbf{v}_i|\mathbf{u}_i, \mathbf{h}_i; \theta_u, \theta_v) = \frac{e^{s^c(\mathbf{u}_i,\mathbf{h}_i,\mathbf{v}_i)}}{e^{s^c(\mathbf{u}_i,\mathbf{h}_i,\mathbf{v}_i)} + \sum_{j\in[B],j\neq i} e^{s^c(\mathbf{u}_i,\mathbf{h}_i,\mathbf{v}_j)}}, \quad (3)$$

$$s^c(\mathbf{u}_i, \mathbf{h}_i, \mathbf{v}_j) = s(\mathbf{u}_i, \mathbf{h}_i, \mathbf{v}_j)/\tau - \log(p_j) \quad (4)$$

where $s(\mathbf{u}, \mathbf{h}, \mathbf{v})$ is the score function, temperature τ is a hyperparameter added to sharpen the predicted logits. $\log(p_j)$ is a correction to the sampling bias, and p_j denotes the sampling probability of item j in a random batch. As in GoogleDNN [41], p_j is estimated by its occurrence intervals. The loss function can be defined as weighted negative log-likelihood:

$$\mathcal{L}_B = -\frac{1}{B}\sum_{i\in[B]} r_i \cdot \log(P_B^c(\mathbf{v}_i|\mathbf{u}_i, \mathbf{h}_i; \theta_u, \theta_v)). \quad (5)$$

Algorithm 1 Supernet training in AutoTower

```
Input: Search space of user tower $\mathcal{T}_u$, search space of item tower
    $\mathcal{T}_v$, maximum training epoch $\mathcal{M}_{\text{train}}$, training dataset $\mathcal{S}_{\text{train}}$;
Output: Weights of supernet $W$;
1: $W$ := initialize_weights();
2: for $i = 1 : \mathcal{M}_{\text{train}}$ do
3:     $t_u$ := uniform_sampling($\mathcal{T}_u$);        # sample user tower
4:     $t_v$ := uniform_sampling($\mathcal{T}_v$);        # sample item tower
       # Train the corresponding weights of supernet
5:     $W$ := train($t_u, t_v, W, \mathcal{S}_{\text{train}}$);
6: end for
7: return Weights of the trained supernet $W$;
```

Since the batches of data for the supernet training are sampled from the training dataset, another advantage of the proposed one-shot search strategy is that its training time is irrelevant to the

Algorithm 2 Evolutionary search in AutoTower

```
Input: Search space of user tower $\mathcal{T}_u$, search space of item tower $\mathcal{T}_v$, supernet weights $\boldsymbol{W}$, population size $P$, complexity constraint of user tower $C_u$, complexity constraint of item tower $C_v$, maximum search iterations $\mathcal{M}_{\text{search}}$, validation dataset $\mathcal{S}_{\text{val}}$;
Output: Top architectures under complexity constraints;
1:  $\mathcal{P}_0 := population\_initialize(P, \mathcal{T}_u, \mathcal{T}_v, C_u, C_v)$;
2:  $n := P/2$;     # the number of crossovers
3:  $m := P/2$;     # the number of mutations
4:  $p := 0.1$;     # the probability of mutation
5:  $top\_k = \emptyset$;
6:  for $i = 1 : \mathcal{M}_{\text{search}}$ do
7:      $loss_{i-1} := batch\_softmax\_loss(\boldsymbol{W}, \mathcal{S}_{val}, \mathcal{P}_{i-1})$;
8:      $top\_k := update\_topk(top\_k, \mathcal{P}_{i-1}, loss_{i-1})$;
9:      $\mathcal{P}_{\text{crossover}} := crossover(top\_k, n, C_u, C_v)$;
10:     $\mathcal{P}_{\text{mutation}} := mutation(top\_k, m, p, C_u, C_v)$;
11:     $\mathcal{P}_i := \mathcal{P}_{\text{crossover}} \cup \mathcal{P}_{\text{mutation}}$;
12: end for
13: return Architectures with the lowest losses in top_k set;
```

dataset scale, which enables the search strategy to adapt to industry large-scale scenarios.

3.3.3 Complexity-Aware Evolutionary Search. After the supernet is well-trained, we begin to search for excellent architectures based on the supernet. Since the supernet has been fully trained, we no longer need to train from scratch for evaluation when we sample an architecture. In other words, we can approximate its performance through directly evaluating it on the validation set. In this case, the evolutionary algorithm is very suitable. Meanwhile, we can explicitly impose model complexity constraints during the evolutionary search process.

As shown in Figure 2, for the user (or item) tower, the architecture is a stack of pooling or choice blocks. The complexity of the whole architecture can be decoupled into the complexity of each pooling or choice block. Thus, in the evolutionary search process, we can effectively restrict the model complexity to ensure that it does not exceed the specific upper bound. Therefore, this is actually a multi-objective evolutionary algorithm that simultaneously takes the model performance and complexity into account. The model complexity can be represented by number of parameters, FLOPs, and inference time [39]. Since the inference time cannot be precisely broken down into the blocks of the architecture, coupled with the fact that it depends on specific hardware and runtime environment, we adopt the number of parameters and FLOPs as complexity constraints in this work.

The overall evolutionary search process is described in Algorithm 2. We first initialize the population by random sampling tower architectures from the search space $\mathcal{T}_u$ and $\mathcal{T}_v$ according to predefined complexity constraints. The fitness of the individual is determined by its validation loss. We always keep top_k architectures with the lowest losses and perform crossover and mutation on them. Specifically, the crossover and mutation are applied to randomly selected choice blocks. Also, the complexity constraints need to be satisfied when performing crossover and mutation. The search process stops when the number of iterations $\mathcal{M}_{\text{search}}$ is reached. We perform crossover and mutation in each iteration and constantly update the top_k set.

Table 2: Statistics of the datasets.

Dataset	#Users	#Categories	#Items	#Categories	#Records
ML-100K	943	4	1,682	3	100,000
ML-1M	6,040	4	3,706	3	1,000,209
Yelp	60,543	6	74,249	6	2,880,522
Amazon	165,036	1	159,551	2	5,073,469

4 EXPERIMENTS

In this section, we conduct extensive experiments to answer the following questions.

- **RQ1**: How are the performances of the architectures searched by AutoTower under different complexity constraints compared with state-of-the-art human-crafted architectures?
- **RQ2**: How do different design strategies (i.e., search strategy and complexity constraint) in AutoTower influence the performance?
- **RQ3**: How do the hyperparameters influence the performance of AutoTower?
- **RQ4**: Are the searched architectures able to be transferred across different datasets?

4.1 Datasets

Experiments are conducted on the following four public datasets, whose statistics are summarized in Table 2. For all datasets, we view all observed instances as positive samples for the retrieval task, and aggregate user behavior sequences by the timestamps.

- **MovieLens-100K**[1] **(ML-100K) & MovieLens-1M**[2] **(ML-1M)**. Two widely used movie-rating datasets containing different numbers of ratings on movies. We set the maximum length of the behavior sequence for each user in both datasets to 20.
- **Yelp**[3]. It is published officially by Yelp, a crowd-sourced review forum website where users can write their comments and reviews for various POIs, such as hotels, restaurants, etc. The maximum length of user behavior sequence for each user is set to 40.
- **Amazon**[4]. It consists of product reviews and metadata from Amazon. We use *Books* category of the Amazon dataset in our experiments. The maximum length of user behavior sequence for each user is set to 50.

With a commonly used manner, we filter out users and items with less than 20 records in Amazon and Yelp. For all preprocessed datasets, we split the data with 2 : 1 : 1 to generate training set, validation set, and test set.

[1]https://grouplens.org/datasets/movielens/100k
[2]https://grouplens.org/datasets/movielens/1m
[3]https://www.yelp.com/dataset/download
[4]https://nijianmo.github.io/amazon/

Table 3: Three different complexity constraint scenarios. "L", "M", and "H" denote low, medium, and high, respectively. "Parms" denotes the number of parameters.

	AutoTower		Baseline
	Parms Constraint	FLOPs Constraint	DNN Layers
L	0.24×10^6	0.15×10^6	[256, 192, 128]
M	0.48×10^6	0.28×10^6	[512, 256, 128]
H	1.20×10^6	0.80×10^6	[1024, 512, 256, 128]

4.2 Experimental Settings

4.2.1 Evaluation Metrics. We adopt two popular standard metrics Recall@K and NDCG@K to evaluate recommendation performance. We use the full-ranking protocol to calculate Recall@K and NDCG@K [6]. Precisely, for each test instance, we calculate scores between the user and all unobserved items, and then take top-K items. Besides the two performance metrics, we also report FLOPs and the number of parameters of the searched architecture. We extend *torchprofile*[5] to accurately account FLOPs.

4.2.2 Baselines. We compare with four widely-used human-crafted models to verify the effectiveness of the proposed NAS method.

- **GoogleDNN** [41], a state-of-the-art two-tower model, which is a variation of *YoutubeDNN* [4]. GoogleDNN uses efficient in-batch negative sampling and introduces a correction into the batch softmax to correct the sampling-bias.
- **SENET**. Borrowing the idea of applying SENET [14] to assign importance to features in FiBiNet [16], we also add a SENET layer after the embedding layer of each tower.
- **BST** [2] uses the powerful *Transformer* structure to capture sequential signals underlying users' behavior sequences. Then, it combines these signals with other user features and feeds them into a DNN structure to form the user tower. The item tower remains the same with GoogleDNN.
- **MIND** [19] is a recent state-of-the-art model for the retrieval stage of recommendation. It designs a multi-interest extractor layer based on the capsule routing mechanism, which is applicable for clustering past behaviors and extracting diverse user interests.

Note that all baselines and AutoTower use exactly the same loss function for fair comparison, i.e. in-batch softmax with correlation as described in Section 3.3.2.

4.2.3 Complexity Constraint Scenarios. Since a main advantage of AutoTower is that it can explicitly constrain the complexity (i.e. number of parameters and FLOPs) of the searched architecture. As Table 3 shows, we design three complexity scenarios of different sizes (i.e., low, medium, and high) to verify its effectiveness under different constraints. For the baselines, the complexity is distinguished by the layer number and the number of units of each DNN layer. For AutoTower, we set the constraints according to the complexity of GoogleDNN under the same scenario.

4.2.4 Implementation Details. All the baselines and AutoTower are implemented in PyTorch[28]. We use the Adam optimizer and the batch size is 1024. The supernet training step is set to 50, 000, and the number of iterations for evolutionary search is set to 50. The population size and the mutation probability are set to 50 and 0.1, respectively. The learning rates for supernet training and retraining are both 0.001. The embedding dimension is set to 16 and the number of choice blocks for AutoTower is set to 5, expect for experiments in the hyperparameter study. The output dimension of each tower is set to 64. We perform early-stopping with patience of 5 based on the validation loss. All experiments run five times with different random seeds and the average performance is reported.

[5]https://github.com/zhijian-liu/torchprofile

4.3 Overall Performance Comparison (RQ1)

Table 4 shows the overall performance comparison between AutoTower and baselines under three different complexity constraints. In addition to Recall@50 and NDCG@50, we also report the numbers of parameters and FLOPs. Moreover, we calculate the average performance improvement relative to the widely used GoogleDNN. From Table 4, we have the following conclusions:

- For human-designed models, a more complex model with much more parameters or sophisticated structures may perform worse than a simpler one, which is mainly due to the overfitting problem. Moreover, there is no single human-designed model that can achieve relatively better performance on all datasets. MIND performs better on *Yelp* and *Amazon*, which indicates that it can better capture users' interests with longer behavior sequences. But MIND performs much worse on *MovieLens*. By observing the training process, we find that the capsule network in MIND is more prone to overfitting on *MovieLens*. **It illustrates the necessity of AutoTower that can design suitable two-tower models automatically in a data-specific manner.**
- AutoTower can achieve the best performances in all datasets and complexity scenarios with much fewer parameters or FLOPs, especially on *Amazon*, which is the most widely-used industry dataset. Compared with GoogleDNN, AutoTower achieves **28.95%-37.22%** average relative improvement of Recall@50, and **33.01%-43.20%** average relative improvement of NDCG@50 with **50.55%-88.03%** average decrease of parameters and **8.00%-70.06%** average decrease of FLOPs. **The significant improvements demonstrate that AutoTower can effectively find better-performing architectures with much lower complexity.**
- The architectures searched by AutoTower under different complexity constraints can achieve comparable performances. **It indicates that complex models may not be necessary for many retrieval applications.** A simpler model can also achieve good performance while having a shorter inference time during online serving.

4.4 Ablation Study (RQ2)

4.4.1 Search Strategy. We also compare AutoTower with other widely used search strategies in our search space to verify the effectiveness of the proposed one-shot search strategy. We choose three representative search strategies for comparison: *differentiable search NASP* [40], *reinforcement learning (RL)* [29], and *random search* [20, 38]. The best results of these search strategies on

Table 4: Overall performance comparison. "R@50" and "N@50" denote Recall@50 and NDCG@50, respectively. The values of "Parms" and "FLOPs" omit unit 10^6, and other values are percentage numbers with "%" omitted. The best results are marked in bold, and the best baselines are marked by underline. * denotes statistically significant improvement (measured by t-test with p-value < 0.005) over the best baselines under the corresponding complexity constrain scenario.

	Model	MovieLens-100K				MovieLens-1M				Yelp				Amazon				$\Delta_{R@50}\uparrow$	$\Delta_{N@50}\uparrow$	$\Delta_{Parms}\downarrow$	$\Delta_{Flops}\downarrow$
		R@50	N@50	Parms	FLOPs	R@50	N@50	Parms	FLOPs	R@50	N@50	Parms	FLOPs	R@50	N@50	Parms	FLOPs				
L	GoogleDNN	36.49	40.86	0.21	**0.14**	32.12	43.64	0.21	**0.14**	8.11	13.95	0.24	0.15	41.68	32.81	0.19	0.13	-	-	-	-
	SENET	34.65	39.53	0.21	**0.14**	33.01	44.89	0.21	**0.14**	7.73	13.52	0.24	0.15	41.68	32.81	0.19	0.13	-1.74	-0.87	+0.00	+0.00
	BST	35.71	39.75	0.43	**0.14**	32.14	44.07	0.43	**0.14**	8.15	13.91	0.72	0.16	49.18	41.68	0.33	0.13	+4.10	+6.25	+120.80	+1.67
	MIND	35.53	37.32	0.10	0.16	22.72	30.87	0.10	0.16	9.10	14.95	0.11	0.17	51.78	44.51	**0.08**	0.14	+1.14	+1.23	**−54.21**	+12.40
	AutoTower	**39.22***	**43.39***	**0.05**	**0.14**	**39.44***	**51.56***	**0.09**	**0.14**	**10.81***	**17.08***	**0.10**	**0.05**	**62.85***	**64.92***	0.17	**0.11**	**+28.59**	**+36.16**	-50.55	**−20.51**
M	GoogleDNN	36.48	40.36	0.43	0.25	31.73	43.61	0.43	0.25	8.17	13.91	0.50	**0.28**	46.43	37.98	0.39	**0.23**	-	-	-	-
	SENET	34.95	38.74	0.43	0.25	33.56	45.69	0.43	0.25	7.99	13.71	0.50	**0.28**	46.43	37.98	0.39	**0.23**	-0.16	-0.17	+0.00	+0.00
	BST	36.03	40.20	0.65	0.25	32.18	44.06	0.65	0.25	8.29	14.04	0.98	0.29	51.41	45.24	0.53	**0.23**	+3.09	+5.17	+58.56	+0.89
	MIND	35.68	37.50	**0.21**	0.32	19.08	26.44	0.21	0.32	8.52	14.26	0.22	0.35	55.00	46.94	0.18	0.29	-4.83	-5.09	-53.04	+26.77
	AutoTower	**40.28***	**43.21***	0.22	**0.23**	**39.77***	**51.84***	**0.15**	**0.19**	**12.65***	**19.34***	**0.20**	**0.28**	**62.16***	**63.45***	**0.12**	**0.23**	**+31.12**	**+33.01**	**−60.80**	**−8.00**
H	GoogleDNN	34.40	39.35	1.57	0.81	31.51	43.36	1.57	0.81	8.07	13.82	1.70	0.88	41.45	31.39	1.49	0.77	-	-	-	-
	SENET	33.09	38.18	1.57	0.81	30.84	42.35	1.57	0.81	7.96	13.78	1.70	0.88	41.45	31.39	1.49	0.77	-1.82	-1.40	+0.00	+0.00
	BST	35.33	39.46	1.79	0.82	31.89	43.77	1.79	0.82	8.01	13.73	2.18	0.89	46.95	38.64	1.63	0.77	+4.11	+5.92	+16.41	+0.90
	MIND	36.58	38.87	0.77	1.17	22.64	30.62	0.77	1.17	8.67	14.46	0.81	1.22	53.30	46.25	0.73	1.10	+3.55	+5.34	-51.32	+42.60
	AutoTower	**39.60***	**43.72***	**0.19**	**0.22**	**39.58***	**51.62***	**0.17**	**0.19**	**12.72***	**19.43***	**0.23**	**0.30**	**62.39***	**63.42***	**0.17**	**0.27**	**+37.22**	**+43.20**	**−88.03**	**−70.06**

Table 5: Comparison between different search strategies. The best results are marked in bold. The complexity constraint scenario in AutoTower is medium.

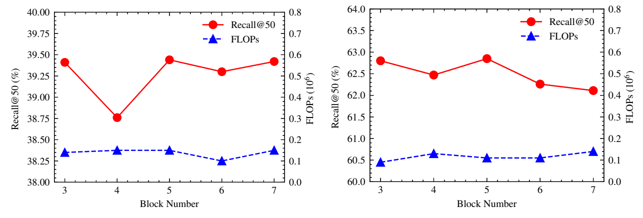
	MovieLens-1M				Amazon			
Strategy	R@50	N@50	Parms	FLOPs	R@50	N@50	Parms	FLOPs
Random	38.50	50.15	0.41	0.24	58.98	58.03	0.38	**0.22**
RL	38.58	50.43	0.39	0.22	61.60	62.53	0.32	0.25
NASP	38.94	50.74	0.29	0.20	59.49	58.90	0.40	0.26
AutoTower	**39.77**	**51.84**	**0.15**	**0.19**	**62.16**	**63.45**	**0.12**	0.23

Table 6: Performances with constraint and without constraint. Recall@50 and FLOPs are reported.

	MovieLens-1M		Amazon	
Constraint	R@50	FLOPs	R@50	FLOPs
Without	39.42	0.39	60.98	0.32
With	**39.44**	**0.15**	**62.85**	**0.11**

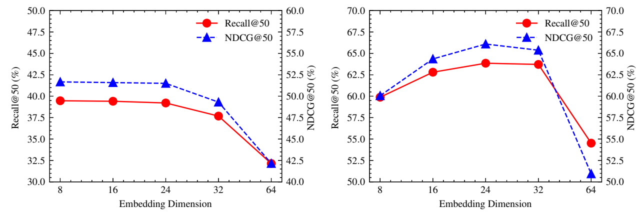
MovieLens-1M and *Amazon* are reported. From Table 5, we can see that AutoTower can consistently outperform other search strategies. Moreover, all search strategies can achieve better performance than the best human-crafted models, which also verifies the effectiveness of the proposed search space. Another advantage of the proposed one-shot strategy is that it can constrain the model complexity in an explicit and flexible manner. As a result, AutoTower can find better models with fewer parameters and FLOPs.

4.4.2 Complexity Constraint. To investigate the influence of complexity constraint in AutoTower, we further compare experimental results with low-complexity constraint and without constraint in Table 6. After removing the constraints, the FLOPs of searched architectures significantly increase. However, the searched architectures do not perform as well as those searched with complexity constraints, which shows that a complex model may not provide additional advantages in the retrieval task and imposing complexity constraint is effective for the architecture search.



(a) MovieLens-1M **(b) Amazon**

Figure 3: Recall@50 and FLOPs with different numbers of choice blocks.



(a) MovieLens-1M **(b) Amazon**

Figure 4: Recall@50 and NDCG@50 with different embedding dimensions.

4.5 Hyperparameter Study (RQ3)

4.5.1 Number of Choice Blocks. Figure 3 shows Recall@50 and FLOPs of searched architectures with different numbers of choice blocks (i.e., from 3 to 7) under the low-complexity constraint. The FLOPs of the searched architectures keep stable as the number of choice blocks increases, due to the explicit complexity constraint. Meanwhile, the Recall performance of AutoTower is also not sensitive to the number of choice blocks.

Table 7: Case study. IP is short for Inner Product, avg is short for average.

DataSet	Pooling	User Tower	Parms	FLOPs	Item Tower	Parms	FLOPs	R@50	N@50
ML-100K	SelfAttention	<MLP-512, SA-1, SA-4>	0.10	0.17	<MLP-128, MLP-512, SA-1>	0.12	0.06	40.28	43.21
ML-1M	SelfAttention	<SA-3, EW-IP, MLP-128>	0.06	0.14	<MLP-512, MLP-64, SA-2>	0.09	0.05	39.77	51.84
Yelp	SelfAttentive	<MLP-32, EW-max, EW-avg, EW-IP, EW-min, EW-max, SA-3>	0.07	0.24	<SA-2, EW-IP, EW-max, MLP-1024, EW-avg, SA-2, MLP-256>	0.50	0.26	12.88	19.66
Amazon	SelfAttentive	<SA-1, EW-avg, SA-2, EW-IP, EW-min, MLP-32, SA-1>	0.05	0.06	<SA-2, EW-IP, EW-avg, EW-avg, SA-1, EW-avg, MLP-256>	0.16	0.08	63.33	65.60

Table 8: Different complexity constraint settings for the user tower and the item tower. "UP" and "UF" denote the number of parameters and FLOPs of the user tower. Similarly, "IP" and "IF" are used for the item tower.

Complexity Constraint	MovieLens-1M					Amazon				
	UP	UF	IP	IF	R@50	UP	UF	IP	IF	R@50
User Tower: L	0.04	0.13	0.20	0.11	39.56	0.04	0.05	0.19	0.09	62.71
Item Tower: H	0.06	0.15	0.25	0.16	39.54	0.03	0.20	0.03	0.02	59.67
User Tower: H	0.10	0.17	0.02	0.01	38.76	0.07	0.22	0.02	0.01	61.54
Item Tower: L	0.09	0.18	0.03	0.02	38.97	0.05	0.22	0.04	0.03	60.77

4.5.2 Embedding Dimension. The embedding dimension is important to the performance of recommendation model. As shown in Figure 4, we further verify the influence of embedding dimension to AutoTower. As the embedding dimension grows larger, the performance on *MovieLens-1M* gradually drops and the performance on *Amazon* first increases then significantly drops. It is easier to suffer from overfitting when the embedding dimension is too large, especially for the dataset with small size (e.g., *MovieLens-1M*).

4.5.3 Different Constrains for User Tower and Item Tower. In real world application scenarios, we may expect different complexities for the user tower and the item tower, e.g. a more complex user tower for richer user features. AutoTower supports for searching architectures with different complexity constraints for the two towers. In Table 8, the first two lines use the high-complexity constraint for the item tower, and the last two lines use the high-complexity constraint for the user tower. We set the constraints according to Table 3. We can see that all searched architectures can achieve comparable performances, which indicates the robustness of AutoTower. Also, different datasets may prefer different configurations. *Amazon* prefers a simpler user tower since it contains limited user features. But *MovieLens-1M* does not show obvious preference.

4.6 Case Study

Table 7 shows the optimal architectures of user towers and item towers searched by AutoTower. Different datasets correspond to different optimal architectures. AutoTower can obtain the most suitable architecture automatically for a specific dataset. Additionally, we observe that *SelfAttention* and *SelfAttentive* show great advantages in processing user behavior sequence.*Yelp* and *Amazon* prefer deeper architectures. Element-Wise blocks that contain few additional parameters and FLOPS are common choices for deeper architectures. To some extent, it explains why our method can achieve better performance with much simpler models. These observations inspire human experts to design more powerful models with the considerations of element-wise operations and shortcut connections.

Table 9: Transferability of the architectures found by AutoTower. Recall@50 is reported.

Source \ Target	ML-100K	ML-1M	Yelp	Amazon
ML-100K	40.28	38.61	12.68	60.97
ML-1M	39.63	39.77	12.36	60.96
Yelp	38.04	38.84	12.88	63.14
Amazon	38.29	39.13	12.41	63.33

4.7 Architecture Transferability (RQ4)

To verify the transferability of the searched architectures, we choose the best architecture searched on each dataset and transfer it to other datasets. The transferred architectures can be found in Table 7. Table 9 shows that the searched architecture on one dataset can also achieve comparable performance on other datasets, indicating that AutoTower has good architecture transferability. It can also demonstrate that the blocks incorporated in the designed search space are general enough to capture user or item representations in the retrieval stage.

5 CONCLUSION AND FUTURE WORK

In this paper, we proposed a NAS-based method called AutoTower to design novel architectures for two-tower recommendation in a resource-specific and data-specific manner. To our best knowledge, we are the first to utilize NAS to search for two-tower models. Specifically, we first designed a single-path based search space, which is simple yet general and expressive. The overall search space is composed of a pooling block and a stack of choice blocks. Moreover, we introduced a shortcut connection technique to the single-path search space. Then, we proposed an efficient one-shot search strategy based on the single-path search space, which contains a weight-sharing based supernet training stage and a complexity-aware evolutionary search stage. Extensive experiments on real-world datasets demonstrate that AutoTower can consistently outperform human-crafted models with much fewer parameters or FLOPs. Moreover, the architectures searched by AutoTower are transferable.

In the future, we plan to introduce more effective operations in the search space. Furthermore, we plan to perform neural architecture search for modeling user behavior sequence.

REFERENCES

[1] Bowen Baker, Otkrist Gupta, Nikhil Naik, and Ramesh Raskar. 2016. Designing neural network architectures using reinforcement learning. *arXiv preprint arXiv:1611.02167* (2016).

[2] Qiwei Chen, Huan Zhao, Wei Li, Pipei Huang, and Wenwu Ou. 2019. Behavior sequence transformer for e-commerce recommendation in alibaba. In *Proceedings of the 1st International Workshop on Deep Learning Practice for High-Dimensional Sparse Data*. 1–4.

[3] 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.

[4] 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.

[5] Yufei Feng, Fuyu Lv, Weichen Shen, Menghan Wang, Fei Sun, Yu Zhu, and Keping Yang. 2019. Deep session interest network for click-through rate prediction. *arXiv preprint arXiv:1905.06482* (2019).

[6] Chen Gao, Quanming Yao, Depeng Jin, and Yong Li. 2021. Efficient Data-specific Model Search for Collaborative Filtering. *arXiv preprint arXiv:2106.07453* (2021).

[7] Weihao Gao, Xiangjun Fan, Chong Wang, Jiankai Sun, Kai Jia, Wenzi Xiao, Ruofan Ding, Xingyan Bin, Hui Yang, and Xiaobing Liu. 2021. Learning An End-to-End Structure for Retrieval in Large-Scale Recommendations. In *Proceedings of the 30th ACM International Conference on Information & Knowledge Management*. 524–533.

[8] Huifeng Guo, Ruiming Tang, Yunming Ye, Zhenguo Li, and Xiuqiang He. 2017. DeepFM: a factorization-machine based neural network for CTR prediction. *arXiv preprint arXiv:1703.04247* (2017).

[9] Zichao Guo, Xiangyu Zhang, Haoyuan Mu, Wen Heng, Zechun Liu, Yichen Wei, and Jian Sun. 2020. Single path one-shot neural architecture search with uniform sampling. In *European Conference on Computer Vision*. Springer, 544–560.

[10] Kaiming He, Xiangyu Zhang, Shaoqing Ren, and Jian Sun. 2016. Deep residual learning for image recognition. In *Proceedings of the IEEE conference on computer vision and pattern recognition*. 770–778.

[11] 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.

[12] Xiangnan He, Zhankui He, Jingkuan Song, Zhenguang Liu, Yu-Gang Jiang, and Tat-Seng Chua. 2018. Nais: Neural attentive item similarity model for recommendation. *IEEE Transactions on Knowledge and Data Engineering* 30, 12 (2018), 2354–2366.

[13] Xiangnan He, Lizi Liao, Hanwang Zhang, Liqiang Nie, Xia Hu, and Tat-Seng Chua. 2017. Neural collaborative filtering. In *Proceedings of the 26th international conference on world wide web*. 173–182.

[14] Jie Hu, Li Shen, and Gang Sun. 2018. Squeeze-and-excitation networks. In *Proceedings of the IEEE conference on computer vision and pattern recognition*. 7132–7141.

[15] Po-Sen Huang, Xiaodong He, Jianfeng Gao, Li Deng, Alex Acero, and Larry Heck. 2013. Learning deep structured semantic models for web search using clickthrough data. In *Proceedings of the 22nd ACM international conference on Information & Knowledge Management*. 2333–2338.

[16] Tongwen Huang, Zhiqi Zhang, and Junlin Zhang. 2019. FiBiNET: combining feature importance and bilinear feature interaction for click-through rate prediction. In *Proceedings of the 13th ACM Conference on Recommender Systems*. 169–177.

[17] Jeff Johnson, Matthijs Douze, and Hervé Jégou. 2017. Billion-scale similarity search with GPUs. *arXiv preprint arXiv:1702.08734* (2017).

[18] Yann LeCun, Yoshua Bengio, and Geoffrey Hinton. 2015. Deep learning. *nature* 521, 7553 (2015), 436–444.

[19] Chao Li, Zhiyuan Liu, Mengmeng Wu, Yuchi Xu, Huan Zhao, Pipei Huang, Guoliang Kang, Qiwei Chen, Wei Li, and Dik Lun Lee. 2019. Multi-interest network with dynamic routing for recommendation at Tmall. In *Proceedings of the 28th ACM International Conference on Information and Knowledge Management*. 2615–2623.

[20] Liam Li and Ameet Talwalkar. 2020. Random search and reproducibility for neural architecture search. In *Uncertainty in artificial intelligence*. PMLR, 367–377.

[21] 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.

[22] Zhouhan Lin, Minwei Feng, Cicero Nogueira dos Santos, Mo Yu, Bing Xiang, Bowen Zhou, and Yoshua Bengio. 2017. A structured self-attentive sentence embedding. *arXiv preprint arXiv:1703.03130* (2017).

[23] Greg Linden, Brent Smith, and Jeremy York. 2003. Amazon. com recommendations: Item-to-item collaborative filtering. *IEEE Internet computing* 7, 1 (2003), 76–80.

[24] Bin Liu, Niannan Xue, Huifeng Guo, Ruiming Tang, Stefanos Zafeiriou, Xiuqiang He, and Zhenguo Li. 2020. AutoGroup: Automatic feature grouping for modelling explicit high-order feature interactions in CTR prediction. In *Proceedings of the 43rd International ACM SIGIR Conference on Research and Development in Information Retrieval*. 199–208.

[25] Bin Liu, Chenxu Zhu, Guilin Li, Weinan Zhang, Jincai Lai, Ruiming Tang, Xiuqiang He, Zhenguo Li, and Yong Yu. 2020. Autofis: Automatic feature interaction selection in factorization models for click-through rate prediction. In *Proceedings of the 26th ACM SIGKDD International Conference on Knowledge Discovery & Data Mining*. 2636–2645.

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

[27] Ze Meng, Jinnian Zhang, Yumeng Li, Jiancheng Li, Tanchao Zhu, and Lifeng Sun. 2021. A General Method For Automatic Discovery of Powerful Interactions In Click-Through Rate Prediction. *arXiv preprint arXiv:2105.10484* (2021).

[28] Adam Paszke, Sam Gross, Francisco Massa, Adam Lerer, James Bradbury, Gregory Chanan, Trevor Killeen, Zeming Lin, Natalia Gimelshein, Luca Antiga, et al. 2019. Pytorch: An imperative style, high-performance deep learning library. *Advances in neural information processing systems* 32 (2019).

[29] 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.

[30] 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 *2016 IEEE 16th International Conference on Data Mining (ICDM)*. IEEE, 1149–1154.

[31] Esteban Real, Alok Aggarwal, Yanping Huang, and Quoc V Le. 2019. Regularized evolution for image classifier architecture search. In *Proceedings of the aaai conference on artificial intelligence*, Vol. 33. 4780–4789.

[32] Olaf Ronneberger, Philipp Fischer, and Thomas Brox. 2015. U-net: Convolutional networks for biomedical image segmentation. In *International Conference on Medical image computing and computer-assisted intervention*. Springer, 234–241.

[33] Badrul Sarwar, George Karypis, Joseph Konstan, and John Riedl. 2001. Item-based collaborative filtering recommendation algorithms. In *Proceedings of the 10th international conference on World Wide Web*. 285–295.

[34] Qingquan Song, Dehua Cheng, Hanning Zhou, Jiyan Yang, Yuandong Tian, and Xia Hu. 2020. Towards automated neural interaction discovery for click-through rate prediction. In *Proceedings of the 26th ACM SIGKDD International Conference on Knowledge Discovery & Data Mining*. 945–955.

[35] 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.

[36] Ashish Vaswani, Noam Shazeer, Niki Parmar, Jakob Uszkoreit, Llion Jones, Aidan N Gomez, Łukasz Kaiser, and Illia Polosukhin. 2017. Attention is all you need. In *Advances in neural information processing systems*. 5998–6008.

[37] Ruoxi Wang, Bin Fu, Gang Fu, and Mingliang Wang. 2017. Deep & cross network for ad click predictions. In *Proceedings of the ADKDD'17*. 1–7.

[38] Saining Xie, Alexander Kirillov, Ross Girshick, and Kaiming He. 2019. Exploring randomly wired neural networks for image recognition. In *Proceedings of the IEEE/CVF International Conference on Computer Vision*. 1284–1293.

[39] Sirui Xie, Hehui Zheng, Chunxiao Liu, and Liang Lin. 2018. SNAS: stochastic neural architecture search. *arXiv preprint arXiv:1812.09926* (2018).

[40] Quanming Yao, Ju Xu, Wei-Wei Tu, and Zhanxing Zhu. 2020. Efficient neural architecture search via proximal iterations. In *Proceedings of the AAAI Conference on Artificial Intelligence*, Vol. 34. 6664–6671.

[41] Xinyang Yi, Ji Yang, Lichan Hong, Derek Zhiyuan Cheng, Lukasz Heldt, Aditee Kumthekar, Zhe Zhao, Li Wei, and Ed Chi. 2019. Sampling-bias-corrected neural modeling for large corpus item recommendations. In *Proceedings of the 13th ACM Conference on Recommender Systems*. 269–277.

[42] Han Zhang, Hongwei Shen, Yiming Qiu, Yunjiang Jiang, Songlin Wang, Sulong Xu, Yun Xiao, Bo Long, and Wen-Yun Yang. 2021. Joint Learning of Deep Retrieval Model and Product Quantization based Embedding Index. *arXiv preprint arXiv:2105.03933* (2021).

[43] Guorui Zhou, Na Mou, Ying Fan, Qi Pi, Weijie Bian, Chang Zhou, Xiaoqiang Zhu, and Kun Gai. 2019. Deep interest evolution network for click-through rate prediction. In *Proceedings of the AAAI conference on artificial intelligence*, Vol. 33. 5941–5948.

[44] Guorui Zhou, Xiaoqiang Zhu, Chenru Song, Ying Fan, Han Zhu, Xiao Ma, Yanghui Yan, Junqi Jin, Han Li, and Kun Gai. 2018. Deep interest network for click-through rate prediction. In *Proceedings of the 24th ACM SIGKDD International Conference on Knowledge Discovery & Data Mining*. 1059–1068.

[45] Jingwei Zhuo, Ziru Xu, Wei Dai, Han Zhu, Han Li, Jian Xu, and Kun Gai. 2020. Learning optimal tree models under beam search. In *International Conference on Machine Learning*. PMLR, 11650–11659.

[46] Barret Zoph and Quoc V Le. 2016. Neural architecture search with reinforcement learning. *arXiv preprint arXiv:1611.01578* (2016).

[47] Barret Zoph, Vijay Vasudevan, Jonathon Shlens, and Quoc V Le. 2018. Learning transferable architectures for scalable image recognition. In *Proceedings of the IEEE/CVF International Conference on Computer Vision*. 8697–8710.