NASRec: Weight Sharing Neural Architecture Search for Recommender Systems

Tunhou Zhang*
Duke University
Durham, USA
tz86@duke.edu

Dehua Cheng
Meta AI
Menlo Park, USA
dehuacheng@fb.com

Yuchen He
Meta AI
Menlo Park, USA
yuchenhe@fb.com

Zhengxing Chen
Meta AI
Menlo Park, USA
czxttkl@fb.com

Xiaoliang Dai
Meta AI
Menlo Park, USA
xiaoliangdai@fb.com

Liang Xiong
Meta AI
Menlo Park, USA
lxiong@fb.com

Feng Yan
University of Nevada, Reno
Reno, USA
fyan@unr.edu

Hai Li
Duke University
Durham, USA
hai.li@duke.edu

Yiran Chen
Duke University
Durham, USA
yiran.chem@duke.edu

Wei Wen†
Meta AI
Menlo Park, USA
wewen@fb.com

ABSTRACT

The rise of deep neural networks provides an important driver in optimizing recommender systems. However, the success of recommender systems lies in delicate architecture fabrication, and thus calls for Neural Architecture Search (NAS) to further improve its modeling. We propose NASRec, a paradigm that trains a single supernet and efficiently produces abundant models/sub-architectures by weight sharing. To overcome the data multi-modality and architecture heterogeneity challenges in recommendation domain, NASRec establishes a large supernet (i.e., search space) to search the full architectures, with the supernet incorporating versatile operator choices and dense connectivity minimizing human prior for flexibility. The scale and heterogeneity in NASRec impose challenges in search, such as training inefficiency, operator-imbalance, and degraded rank correlation. We tackle these challenges by proposing single-operator any-connection sampling, operator-balancing interaction modules, and post-training fine-tuning. Our results on three Click-Through Rates (CTR) prediction benchmarks show that NASRec can outperform both manually designed models and existing NAS methods, achieving state-of-the-art performance.

1 INTRODUCTION

Deep learning is playing an essential role on designing modern recommender systems at web-scale in real-world applications. For example, the most widely used search engines and social medias harness recommender systems (or ranking systems) to optimize the Click-Through Rates (CTR) of personalized pages [6, 17]. The improvement of recommender systems is driven by neural architecture engineering of deep learning models.

Deep learning based recommender systems, especially CTR prediction, carries a neural architecture design upon multi-modality features. In practice, various challenges arise in such a procedure. The multi-modality features, such as floating-point, integer, and categorical features, present a concrete challenge in feature interaction modeling and neural network optimization. Finding a good backbone model with heterogeneous architectures assigning appropriate priors upon multi-modality features are common practices in deep learning recommender systems [5, 10, 12, 14, 17, 19–21]. Yet, these approaches still rely on significant manual efforts and suffer from limitations such as narrow design spaces and insufficient experimental trials bounded by available resources. As a result, these limitations add difficulty in designing a capable feature extractor.

The rise of Automated Machine Learning (AutoML), especially Neural Architecture Search (NAS) [3, 15, 29, 31], in the vision domain, sheds light in optimizing models of recommender systems. However, applying NAS to recommendation domain is much more challenging than vision domain because of the multi-modality in data and the heterogeneity in the architectures. For example, (1) in vision, inputs of building blocks are homogeneously 3D tensors, but recommender systems take in multi-modality features generating 2D and 3D tensors. (2) Vision models simply stack the same building blocks, and thus state-of-the-art NAS in vision converges to simply searching size configurations instead of architecture motifs, such as channel width, kernel sizes, and layer repeats [3, 30]. However, recommendation models are heterogeneous with each stage of the model using a completely different building block [5, 10, 14, 17]. (3) Vision models mainly use convolutional operator as the main building block while recommender systems are built over heterogeneous operators, such as, Fully-Connected layer, Gating, Sum, Dot-Product, Attention, etc.

Due to the aforementioned challenges, study of NAS in recommender systems is limited. For example, search spaces in AutoCTR [23] and DNAS [13] follow the design principle of human-crafted DLRM [17] and they only include fully-connected layer and dot product as searchable operators. They also heavily reply on manually crafted operators, such as Factorization Machine [23] or feature interaction module [8] in the search space to increase architecture heterogeneity. Moreover, existing works either suffer from huge computation cost [23] or challenging bi-level optimization [13], and thus they only employ narrow design spaces (sometimes with strong human priors [8]) to craft architectures. Although these approaches can explore better models than hand-crafted solutions, their limitations discourage diversified feature interactions and limit the potential of discovered models.

In this paper, we hereby propose NASRec, a new paradigm to fully enable **NAS** for **rec**ommender systems via Weight Sharing Neural Architecture Search (WS-NAS) under data modality and architecture heterogeneity. Table 1 summarizes the advancement

*A majority of this work was done when the first author was an intern at Meta Platforms, Inc.
†Corresponding author. Intern Manager.

Table 1: Comparison of NASRec vs. existing NAS methods for recommender systems.

Method	Building Operators?	Dense Connectivity?	Full arch Search?	Criteo Log Loss	Training Cost
DNAS [13]	FC, Dot-Product	✓		0.4442	One supernet
PROFIT [8]	FC, FM			0.4427	One supernet
AutoCTR [23]	FC, Dot-Product, FM, EmbedFC	✓	✓	0.4413	Many models
NASRec	FC, Gating, Sum, Attention Dot-Product, EmbedFC	✓	✓	0.4407	One supernet

of NASRec over other NAS approaches. We achieve this by first building up a supernet that incorporates much more heterogeneous operators than previous works, including Fully-Connected (FC) layer, Gating, Sum, Dot-Product, Self-Attention, and Embedded Fully-Connected (EmbedFC) layer. In the supernet, we densely connect a cascade of blocks, each of which includes all operators as options. As any block can take in any raw feature embeddings and intermediate tensors by dense connectivity, the supernet is not limited by any particular data modality. Such supernet design minimizes the encoding of human priors by introducing "NASRec Search Space", supporting the nature of data modality and architecture heterogeneity in recommender systems, and covering models beyond popular recommendation models such as Wide & Deep [5], DLRM [17], AutoCTR [23], DNAS [13] and PROFIT [8].

The supernet itself forms a search space. We obtain a model by zeroing out some operators and connections in the supernet, that is, a subnet of the supernet is equivalent to a model. As all subnets share weights from the same supernet, it is dubbed as Weight Sharing NAS. To efficiently search models/subnets in the NASRec search space, we advance one-shot approaches [3, 30] to recommendation domain. We propose *Single-operator Any-connection sampling* to decouple operator selections and increase connection coverage, *operator-balancing interaction* blocks to fairly train subnets in the supernet, and *post-training fine-tuning* to reduce weight co-adaptation. These approaches allow a better ranking of subnet models in the supernet (i.e., 0.16 Pearson ρ improvement and 0.11 Kendall τ improvement on full NASRec search space) and a more efficient training of the supernet (i.e., up to 2.5× lower training time). An improved ranking quality and training efficiency gives searchers a stronger signal and lower cost to distinguish models.

We evaluate NASRec on three popular CTR benchmarks and demonstrate the significant improvements compared to both handcrafted models and NAS-crafted models. Remarkably, NASRec advances the state-of-the-art with log loss reduction of $\sim$ 0.001, $\sim$ 0.002 on Criteo and KDD Cup 2012, respectively. On Avazu, NASRec advances the state-of-the-art PROFIT [8] with AUC improvement of 0.002 and on-par log loss, while outperforming PROFIT [8] on Criteo by 0.002 log loss reduction.

In addition, NASRec only needs to train a single supernet thanks to the efficient weight sharing mechanism, and thus greatly reduces the search cost. We summarize our major contributions below.

- We propose NASRec, a new paradigm to scale up automated modeling of recommender systems. NASRec establishes a flexible supernet (search space) with minimal human priors, overcoming data modality and architecture heterogeneity challenges in recommendation domain.
- We advance weight sharing NAS to recommendation domain by introducing single-operator any-connection sampling, operator-balancing interaction modules, and post-training fine-tuning.
- NASRec outperforms both manually crafted models and models discovered by NAS methods with smaller search cost.

2 RELATED WORK

Deep learning based recommender systems. Machine-based recommender systems such as click-through rate prediction has been thoroughly investigated in various approaches, such as Logistic Regression [20], and Gradient-Boosting Decision Trees [12]. More recent approaches study deep learning based interaction of different types of features via Wide & Deep Neural Networks [5], DeepCrossing [21], Factorization Machines [10, 14], Dot Product [17] and gating mechanism [27, 28]. Yet, these works operate the cost of tremendous manual efforts and suffer from sub-optimal performance and constrained design choices due to the limitations in resource supply. Our work establishes a new paradigm on learning effective recommendation models by crafting a scalable "NASRec search space" that incorporates all popular design motifs in existing works. The new NASRec search space supports a wide range of design choices and enables scalable optimization to craft recommendation models of varying requirements.

Neural Architecture Search. Neural Architecture Search automates the design of Deep Neural Networks in various applications: the popularity of Neural Architecture Search is consistently growing in brewing Computer Vision [3, 15, 29, 31], Natural Language Processing [22, 26], and Recommendation Systems [8, 13, 23]. Recently, Weight-Sharing NAS (WS-NAS) [3, 26] attracts the attention of researchers: it trains a supernet that represents the whole search space directly on target tasks, and efficiently evaluate subnets (i.e., sub architectures of supernet) with shared supernet weights. Yet, carrying WS-NAS on recommender systems is challenging because recommender systems are brewed upon heterogeneous architectures that are dedicated for interacting multi-modality data, thus require more flexible search spaces and effective supernet training algorithms. Those challenges make WS-NAS provide a lower rank correlation to distinguish models. NASRec addresses them by proposing single-operator any-connection sampling, operator-balancing interaction modules, and post-training fine-tuning.

3 HIERARCHICAL NASREC SPACE FOR RECOMMENDER SYSTEMS

To support data modality and architecture heterogeneity in recommender systems, the flexibility of search space is the key. We establish a new paradigm free of human priors by introducing

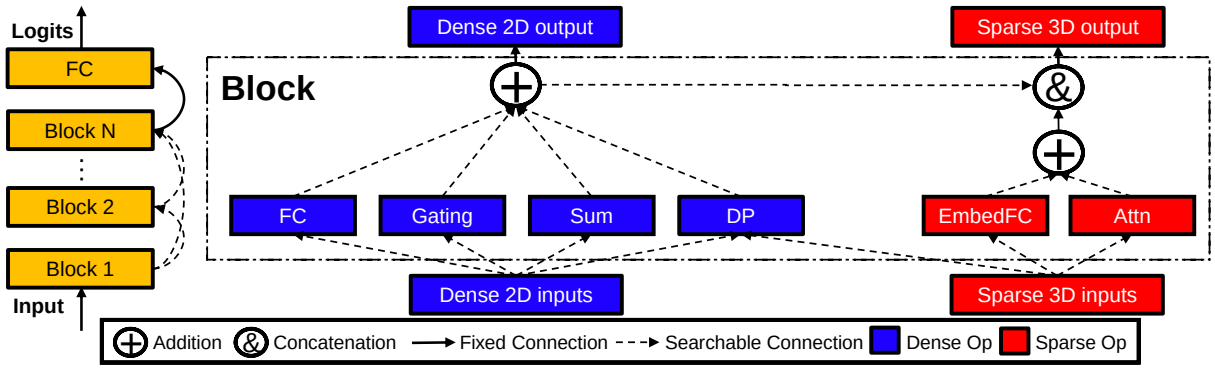

Figure 1: Overview of NASRec search space. NASRec search space enables a full architecture search on building operators and dense connectivity. Here, "blue" blocks produce dense outputs, and "red" blocks produce sparse outputs.

NASRec search space, a hierarchical search space design that incorporates heterogeneous building operators and dense connectivity, see Figure 1. The major manual process in designing the search space is simply collecting common operators used in existing approaches [10, 14, 17, 23, 27, 28]. Beyond that, we further incorporate the prevailing self-attention operator into the NASRec search space for better flexibility and higher potential in searched architectures, thanks to its dominance in applications such as ViT [7] for image recognition, Transformer [25] for natural language processing, and its emerging exploration in recommender systems [4, 9]. Next, we demonstrate the NASRec search space.

3.1 NASRec Search Space

In recommender systems, we define a dense input as $X_d \in \mathbb{R}^{B \times dim_d}$ which is a 2D tensor from either raw dense features or generated by operators, such as FC, Gating, Sum, and Dot-Product. A sparse input $X_s \in \mathbb{R}^{B \times N_s \times dim_s}$ is a 3D tensor of sparse embeddings either generated by raw sparse/categorical features or by operators such as EmbedFC and self-attention. Similarly, a dense or sparse output (i.e., Y_d or Y_s) is respectively defined as a 2D or 3D tensor produced via a corresponding building blocks/operators. In NASRec, all sparse inputs and outputs share the same dim_s, which equals to the dimension of raw sparse embeddings. Accordingly, we define a dense (sparse) operator as an operator that produces a dense (sparse) output. In NASRec, dense operators include FC, Gating, Sum, and Dot-Product which form the "dense branch" (marked in blue), and sparse operators include EmbedFC and self-attention, which form the "sparse branch" (marked in red).

A candidate architecture in NASRec search space is a stack of N choice blocks, followed by a final FC layer to compute logit. Each choice block admits an arbitrary number of multi-modality inputs, each of which is $X = (X_d, X_s)$ from a previous block or raw inputs, and produces a multi-modality output $Y = (Y_d, Y_s)$ of both a dense tensor Y_d and a sparse tensor Y_s via internal building operators. Within each choice block, we can sample operators for search.

We construct a supernet to represent the NASRec search space, see Figure 1. The supernet subsumes all possible candidate models/subnets and performs weight sharing among subnets to simultaneously train all of them. We formally define the NASRec supernet $\mathcal{S}$ as a tuple of connections C, operators O, and dimensions $\mathcal{D}$ as follows: $\mathcal{S} = (C, \mathcal{D}, O)$ over all N choice blocks. Specifically, the operators: $O = [O^{(1)}, ..., O^{(N)}]$ enumerates the set of building operators from choice block 1 to N. The connections: $C = [C^{(1)}, ..., C^{(N)}]$ contains the connectivity $< i, j >$ between choice block i and choice block j. The dimension: $\mathcal{D} = [D^{(1)}, ..., D^{(N)}]$ contains the dimension settings from choice block 1 to N.

A subnet $S_{sample} = (O_{sample}, C_{sample}, \mathcal{D}_{sample})$ in the supernet $\mathcal{S}$ represents a model in NASRec search space. A block uses addition to aggregate the outputs of sampled operators in each branch (i.e. "dense branch" or "sparse branch"). When the operator output dimensions do not match, we apply a zero masking to mask out the extra dimension. A block uses concatenation *Concat* to aggregate the outputs from sampled connections. Given a sampled subnet S_{sample}, the input $X^{(N)}$ to choice block N is computed as follows given a list of previous block outputs $\{Y^{(1)}, ..., Y^{(N-1)}\}$ and the sampled connections $C^{(N)}_{sample}$:

$$X_d^{(N)} = Concat_{i=1}^{N-1}[Y_d^{(i)} \cdot \mathbf{1}_{<i,N> \in C^{(N)}_{sample}}], \quad (1)$$

$$X_s^{(N)} = Concat_{i=1}^{N-1}[Y_s^{(i)} \cdot \mathbf{1}_{<i,N> \in C^{(N)}_{sample}}]. \quad (2)$$

Here, $\mathbf{1}_b$ is 1 when b is true otherwise 0. An optional FC/embedded FC layer is inserted to match dimension of 2-D/3-D inputs if needed.

A building operator $o \in O^{(N)}_{sample}$ transforms the concatenated input $X^{(N)}$ into an intermediate output with a sampled dimension $D^{(N)}_{sample}$. This is achieved by a mask function that applied on the last dimension for dense output and middle dimension for sparse output. For example, a dense output $Y_d^{(N)}$ is obtained as follows:

$$Y_d^{(N)} = \sum_{o \in O} \mathbf{1}_{o \in O^{(N)}_{sample}} \cdot Mask(o(X_d^{(N)}), D^{(N)}_{sample,o}). \quad (3)$$

where

$$Mask(V, d) = \begin{cases} V_{:,i}, & \text{if } i < d \\ 0, & \text{Otherwise.} \end{cases}. \quad (4)$$

Next, we clarify the set of building operators as follows:

- **Fully-connected (FC) layer.** Fully-connected layer is the backbone of DNN models for recommender systems [5] that extracts dense representations. FC is applied on 2D dense inputs, and followed by a ReLU activation.

- **Sigmoid Gating (Gating) layer.** We follow the intuition in [4, 28] and employ a dense building operator, Sigmoid Gating, to enhance the potential of the search space. Given two dense inputs $X_d = (X_{d1} \in \mathbb{R}^{B \times dim_{d1}}, X_{d2} \in \mathbb{R}^{B \times dim_{d2}})$, Sigmoid Gating interacts these two inputs as follows: $Gating(X_{d1}, X_{d2}) = sigmoid(FC(X_{d1})) * X_{d2}$. If the dimension of two dense inputs does not match, a FC layer is applied on X_{d2} as a projection.
- **Sum layer.** This dense building operator adds two dense inputs: $X_d = (X_{d1} \in \mathbb{R}^{B \times dim_{d1}}, X_{d2} \in \mathbb{R}^{B \times dim_{d2}})$ and merges these two inputs from different levels of the recommender system models by simply performing $Sum(X_{d1}, X_{d2}) = X_{d1} + X_{d2}$. Similar to Sigmoid Gating, a FC layer is utilized as a projection if the dimensions are unmatched.
- **Dot-Product (DP) layer.** We leverage Dot-Product to grasp the interactions among multi-modality inputs via a pairwise inner products. Dot-Product can take dense and/or sparse inputs, and produce a dense output. These sparse inputs, after being sent to "dense branch", can later take advantage of the dense operators to learn better representations and interactions. Given a dense input $X_d \in \mathbb{R}^{B \times dim_d}$ and a sparse input $X_s \in \mathbb{R}^{B \times N_c \times dim_s}$, a Dot-Product first concatenate them as $X = Concat[X_d, X_s]$, and then performs pair-wise inner products: $Dot-Product(X_d, X_s) = Triu(XX^T)$. dim_d is first projected to dim_s if they do not match.
- **Embedded Fully-Connected (EmbedFC) layer** Embedded FC layer is a sparse building operator that applies FC along the middle dimension. Specifically, an EmbedFC with weights $W \in \mathbb{R}^{N_{in} \times N_{out}}$ transforms an input $X_s \in \mathbb{R}^{B \times N_{in} \times dim_s}$ to $Y_s \in \mathbb{R}^{B \times N_{out} \times dim_s}$
- **Self-Attention (Attn) layer.** We use attention [25] mechanism in vision and natural language processing to learn the weighting of different sparse inputs and better exploit their interaction in recommendation systems. Specifically, self-attention is applied on a given sparse input $X_s \in \mathbb{R}^{B \times N_s \times dim_s}$. Self-attention uses identical queries, keys, and values to interact different sparse inputs as follows: $Attn(X_c) = softmax(\frac{X_c X_c^T}{\sqrt{dim_s}} X_c)$.

We observe that the aforementioned set of building operators provide opportunities for the sparse inputs to transform into the "dense branch". Yet, these operators do not permit a transformation of dense inputs towards the "sparse branch". To address this limitation, we allow dense outputs to optionally merge into the "sparse branch" and design a new module **Project-Concatenate** to optionally interact dense and sparse outputs. Specifically, Project-Concatenate first projects the dense output to match the sparse dimension (i.e., embedding dimension) and concatenates it with sparse outputs. As Project-Concatenate module is a new way to interact outputs, it enhances data modality to the NASRec search space with more heterogeneous architecture choices.

Beyond the rich choices of building operators, each choice block can also receive inputs from any preceding choice blocks, and raw input features. This involves an exploration of any connectivity among choice blocks and raw inputs, extending the wiring heterogeneity for search. Next, we introduce the search components in NASRec and provide two search spaces that we use in experiments.

3.2 Search Components

In NASRec search space, we search the connectivity, operator dimensions, and building operators in each choice block. We illustrate the three key search components as follows:

- **Connection.** We place no restrictions on the number of connections that a choice block can receive: each block can choose inputs from an arbitrary number of preceding blocks and raw inputs. Specifically, the n-th choice block can connect to any previous $n-1$ choice blocks and the raw dense (sparse) features. The outputs from all preceding blocks are concatenated as inputs for dense (sparse) building blocks. We separately concatenate the dense (sparse) outputs from preceding blocks.
- **Dimension.** In a choice block, different operators may produce different tensor dimensions. In NASRec, we set the output sizes of FC and EmbedFC to dim_d and N_s, respectively; and other operator outputs in dense (sparse) branch are linearly projected to dim_d (N_s). This ensures operator outputs in each branch have the same dimension and can add together. This also give the maximum dimensions dim_d and N_s for the dense output $Y_d \in \mathbb{R}^{B \times dim_d}$ and the sparse output $Y_s \in \mathbb{R}^{B \times N_s \times dim_s}$. Given a dense or sparse output, a mask in Eq. 4 zeros out the extra dimensions, which allows flexible selections of dimensions of building operators.
- **Operator.** Each block can choose at least one dense (sparse) building operator to transform inputs to a dense (sparse) output. Each block should maintain at least one operator in the dense (sparse) branch to ensure the flow of information from inputs to logit. We independently sample building operators in the dense (sparse) branch to form a validate candidate architecture. In addition, we independently sample Project-Concatenate path to allow optional dense-to-sparse interaction.

We craft two NASRec search spaces as examples to demonstrate the power of NASRec search space.

- *NASRec-Small.* We limit the choice of operators within each block to FC, EmbedFC, and Dot-Product as inspired by AutoCTR [23][1]. We allow any connectivity between blocks.
- *NASRec-Full.* We enable all building operators and connections to construct an aggressive search space for exploration with minimal human priors. Under the constraint that at least one operator must be sampled in both dense and sparse branch, the *NASRec-Full* search space size is $15^N \times$ of *NASRec-Small*, where N is the number of choice blocks. This full search space extremely tests the capability of NASRec.

The combination of full dense connectivity search and independent dense/sparse dimension configuration gives the NASRec search space a large cardinality. *NASRec-Full* has $N = 7$ blocks, 5 possible dimension choices for both dense and sparse dimensions. NASRec search space contains up to 5×10^{29} architectures with strong heterogeneity. With minimal human priors and such unconstrained search space, brutal-force sample-based methods may take enormous time to find a state-of-the-art model. This calls for a NASRec supernet to cover the overall search space and brew all candidate architectures in an efficient manner.

[1] We do not utilize FM because it does not appear in the best found models in AutoCTR.

4 WEIGHT SHARING NEURAL ARCHITECTURE SEARCH FOR RECOMMENDER SYSTEMS

A NASRec supernet simultaneously brews different subnet models in the NASRec search space, yet imposes challenges to training efficiency and ranking quality due to its large cardinality. In this section, we first propose a novel path sampling strategy, *Single-operator Any-connection* sampling, that decouples operator sampling with a good connection sampling converge. We further observe the operator imbalance phenomenon induced by some over-parameterized operators, and tackle this issue by *operator-balancing interation* to improve supernet ranking. Finally, we employ *post-training fine-tuning* to alleviate weight co-adaptation, and further utilize regularized evolution to obtain the subnet with the best performance. WE also provide a set of insights that effectively explore the best recommender models.

4.1 Single-operator Any-Connection Sampling

The supernet training adopts a drop-out like approach. At each mini-batch, we sample and train a subnet. During training, we train lots of subnets under weight sharing, with the goal that subnets are well trained to predict the performance of models. Sampling strategies are important to meet the goal. We explore three path sampling strategies depicted in Figure 2 and discover Single-operator Any-Connection sampling is the most effective way:

- **Single-operator Single-connection strategy.** This path sampling strategy has its root in Computer Vision [11]: it uniformly samples a single dense and a single sparse operator in each choice block, and uniformly samples a single connection as an input to a block. The strategy is efficient because, on average, only a small subnet is trained at one mini-batch, however, this strategy only encourages chain-like formulation of models without extra connectivity patterns. The lack of connectivity coverage yields slower convergence, poor performance, and inaccurate ranking of models as we will show.
- **Any-operator Any-connection Strategy.** This sampling strategy increases the coverage of sub-architectures of supernet during subnet training: it uniformly samples an arbitrary number of dense and sparse operators in each choice block, and uniformly sample an arbitrary number of connections to aggregate different block outputs. Yet, the training efficiency is poor when training sampled large subnets. More importantly, the weight co-adaptation of multiple operators within a choice block may affect independent evaluation of the subnets, and thus eventually lead to poor ranking quality as we will show.
- **Single-operator Any-connection**. We propose this path sampling strategy to combine the strengths from above two strategies. Single-operator Any-connection samples a single dense and a single sparse operator in each choice block, and samples an arbitrary number of connections to aggregate the outputs from different choice blocks. The key insight of this strategy is separating the sampling of parametric operators to avoid the co-adaptation of weights, and allowing arbitrarily sample of non-parametric connections to gain a good coverage of the NASRec search space. Thus, it provides a more accurate performance evaluation and ranking of the subnets.

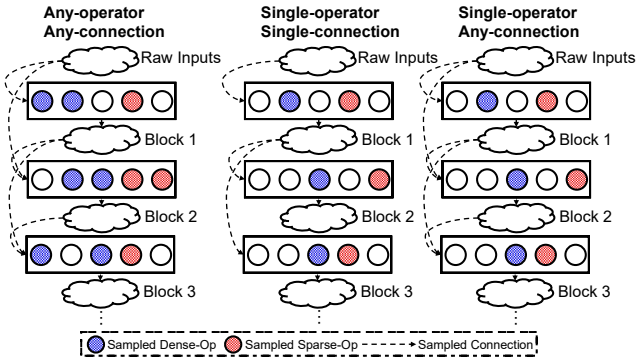

Figure 2: We propose Single-operator Any-connection path sampling by combining the advantages of the first two sampling strategies. Here, dashed connections and operators denotes a sampled path in supernet.

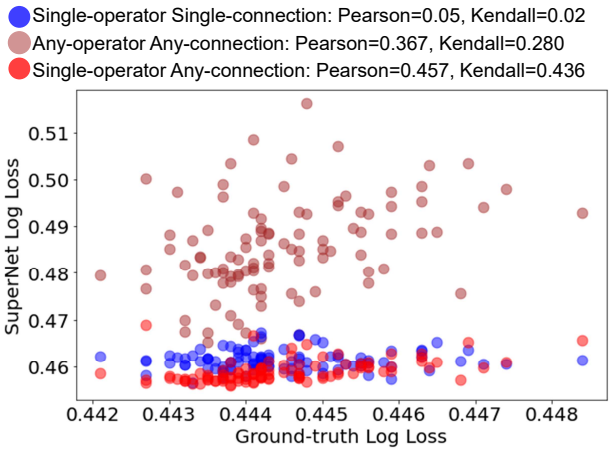

Figure 3: Ranking evaluation of various path sampling strategies on *NASRec-Full* supernet. We evaluate all ranking coefficients over 100 randomly sampled subnets on Criteo.

In more details, compared to Any-operator Any-connection sampling, Single-operator Any-connection sampling achieves higher training efficiency: the reduced number of sampled operators reduces the training cost by up to 1.5×. In addition, Single-operator Any-connection samples medium-sized networks more frequently. These medium-sized networks achieve the best trade-off between model size and performance as we will show in Table 5.

We evaluate the ranking of subnets by WS-NAS on Criteo and by 100 randomly sampled networks in Figure 3. Here, we adopt the design of operator-balancing interaction modules in Section 4.2 to maximize the potential of each path sampling strategy. In the figure, the y-axis is the Log Loss of subnets, whose weights are copied from corresponding architectures in the trained supernet. Single-operator Any-connection achieves at least 0.16 higher Kendall Tau and 0.11 higher Pearson rho compared to other path sampling strategies. In addition, we observe that Single-operator Any-connection sampling allows better convergence of the NASRec supernet and subnets that inherit weights from supernet achieve lower log loss during validation, leading to a better exploitation of their ground-truth performance for a better ranking quality.

4.2 Operator-Balancing Interaction Modules

Recommender systems involve multi-modality data with an indefinite number of inputs, for example, a large number of sparse inputs. We define operator imbalance as the imbalance of the numbers of weights between operators within a block. In weight-sharing NAS, operator imbalance may cause the issue that supernet training may favor operators with more weights. This will offset the gains due to poor ranking correlations of subnets: the subnet performance in supernet may deviate from its ground-truth performance when trained from scratch. We identify that, in our NASRec, such an issue is strongly related to the Dot-Product operator, and provide mitigation to address such operator imbalance.

Given N_s sparse embeddings, a Dot-Product block produces $N_s^2/2$ pairwise interactions as a quadratic function on the number of sparse embeddings. As detailed in Section 3.1, the supernet requires a linear projection layer (i.e., FC) to match the output dimensions of operators within each choice block. Typically for Dot-Product, this leads to an extra $(N_s^2 \cdot dim_d/2)$ trainable weights.

However, the weight consumption of such projection layer is large given a large number of sparse embeddings. For example, given $N_s = 448$ and $dim_d = 512$ in a 7-block NASRec supernet, the projection layer induces over $50M$ parameters in the NASRec supernet, which has a similar scale of parameter consumption with sparse embedding layers. Moreover, such tremendous weight parameterization is a quadratic function of the number of sparse inputs N_s, yet other building operators have much fewer weights, such as, the number of trainable weights in EmbedFC is a linear function of the number of sparse inputs N_s. As a result, the over-parameterization in Dot-Product leads to an increased convergence rate for the Dot-Product operator and consequently favor parameter-consuming subnets with a high concentration of Dot-Product operations as we observed. In addition, the ignorance of other heterogeneous operators other than Dot-Product provides a poor ranking of subnets, leading to sub-optimal performance on recommender systems.

We insert a simple EmbedFC as a projection layer before the Dot-Product to mitigate such over-parameterization, see Figure 4. Our intuition is projecting the number of sparse embeddings in Dot-Product to $[\sqrt{2dim_d}]$, such that the following Dot-Product operator produces approximately dim_d outputs that later requires a minimal projection layer to match the dimension. As such, the Dot-Product operator consumes at most $(dim_d^2 + N_s[\sqrt{2dim_d}])$ trainable weights and ensures a linear growth of parameter consumption with the number of sparse EmbedFC N_s. Thus, we balance interaction operator to allow a more similar convergence rate of all building operators. Table 2 reflects a significant enhancement on the training efficiency and ranking quality of the *NASRec-full* supernet with Single-operator Any-connection path sampling strategy.

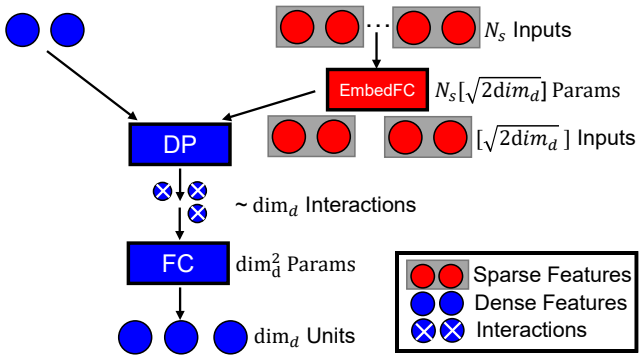

Figure 4: Operator-balancing interaction. A simple projection layer before Dot-Product ensures linear parameter consumption and balances building operators.

Table 2: Operator-Balancing Interactions reduces supernet training cost and improves ranking of subnets.

Interaction Type	Training Cost	Pearson ρ	Kendall τ
Imbalanced DP	5 hours	0.31	0.32
Balanced DP	2 hours	0.46	0.43

4.3 Post-training Fine-tuning

Although dropout-like subnet training provide a great way to reduce the adaptation of weights for a specific subnet, the subnet performance prediction by supernet can fail when weights should not share across some subnets. After the supernet training and during a stand alone subnet evaluation, we carry a post-training fine-tuning that re-adapt its weights back to the specific subnet. This can re-calibrate the weights which are corrupted when training other subnets during the supernet training. In practice, we find that fine-tuning the last FC on the target dataset for a few training steps (e.g., 0.5K) is good enough. With only marginal extra search cost, this novel post-training fine-tuning technique boosts the ranking of subnets by addressing the underlying weight adaptation issue, and thus provides a better chance to discover better models for recommender systems.

Table 3 demonstrates the improvement of post-training fine-tuning on different path sampling strategies. Surprisingly, post-training fine-tuning achieves decent ranking quality improvement under Single-operator Single-connection and Any-operator Any-connection path sampling strategy. This is because subnets under these strategies do not usually converge well in supernet: they either suffer from poor supernet coverage, or poor convergence induced by co-adaptation. The fine-tuning process releases their potential and approaches their real performance on the target dataset. Remarkably, Single-operator Any-connection path sampling strategy cooperates well with post-training fine-tuning, and achieves the global optimal Pearson ρ and Kendall τ ranking correlation among different approaches, with at least 0.15 Kendall τ improvement and 0.14 Pearson ρ improvement on *NASRec-Full* search space over Single-operator Single-connection sampling without fine-tuning.

Table 3: Effects of post-training fine-tuning on different path sampling strategies on *NASRec-Full*. We demonstrate Pearson ρ and Kendall τ over 100 random subnets on Criteo.

Path Sampling Strategy	No Fine-tuning		Fine-tuning	
	Pearson ρ	Kendall τ	Pearson ρ	Kendall τ
Any-operator Any-connection	0.37	0.28	0.46	0.43
Single-operator Single-connection	0.05	0.02	0.43	0.29
Single-operator Any-connection	0.46	0.43	0.57	0.43

4.4 Evolutionary Search on Best Models

We utilize regularized evolution [18] to obtain the best child subnet in NASRec search space, including *NASRec Small* and *NASRec-Full*. Here, we first introduce a single mutation of a hierarchical genotype with the following sequence of actions in one of the choice blocks:

- Re-sample the dimension of one dense building operator.
- Re-sample the dimension of one sparse building operator.
- Re-sample one dense building operator.
- Re-sample one sparse building operator.
- Re-sample its connection to other choice blocks.
- Re-sample the choice of Project-Concat block that enables the interaction between dense and sparse outputs.

The mutation space covers both the search components of operators, connections, and dimensions in NASRec search spaces. We use **Adaptive Mutation Strategy** to explore subnets in NASRec search space. The original regularized evolution performs a single mutation to generate child architectures from a given parent architecture. Unfortunately, directly applying regularized evolution may suffer from locally optimal solution, mostly caused by a bad initialization point (i.e., initial population) during the initial phase of regularized evolution. To mitigate this issue, we envision that, at the initial phase of regularized evolution, we can perform multiple mutations to generate a child architecture from a parent architecture. Because initial parent architectures are unlikely to be optimal, performing more mutations at the early stage allow the evolutionary search to explore a wider region of the NASRec search space, thus leads to potentially better solutions. In addition, the number of mutations to generate a child architecture will gradually decay to 1 as the search progresses.

5 EXPERIMENTS

We first show the detailed configuration that NASRec employs during architecture search, model selection and final evaluation. Then, we demonstrate empirical evaluations on three popular recommender system benchmarks for Click-Through Rates (CTR) prediction: Criteo[2], Avazu[3] and KDD Cup 2012[4]. All three datasets are pre-processed in the same fashion as AutoCTR [23].

5.1 Search Configuration

We first demonstrate the detailed configuration of *NASRec-Full* search space as follows:

- **Connection Search Components.** We utilize $N = 7$ blocks in our NASRec search space to keep a fair comparison with existing NAS methods, such as AutoCTR [23]. All choice blocks can arbitrarily connect to previous choice blocks or raw features.
- **Operator Search Components.** In each choice block, our search space contains 6 distinct building operators, including 4 dense building operators: FC, Gating, Sum, Dot-Product and 2 distinct sparse building operators: Self-Attention and EmbedFC.
- **Dimension Search Components.** For each dense building operator, the dense output dimension can choose from {32, 64, 128, 256, 512}. For each sparse building operator, the sparse output dimension can be chosen from {16, 32, 64}.

[2]http://labs.criteo.com/2014/02/kaggle-display-advertising-challenge-dataset
[3]https://www.kaggle.com/c/avazu-ctr-prediction/data
[4]https://www.kaggle.com/c/kddcup2012-track2/data

In *NASRec-Small*, we employ the same settings except that we use only 2 dense building operators: FC, Dot-Product and 1 sparse building operator: EmbedFC. Then, we illustrate some techniques on brewing the NASRec supernet, including the configuration of embedding, supernet warm-up, and supernet training settings.

- **Capped Embedding Table.** We cap the maximum embedding table size to 0.5M during supernet training for search efficiency. This gives 2~3 × speedup compared to using the full embedding table. During the final evaluation, we maintain the full embedding table to retrieve the best performance, i.e., a total of 540M parameters in DLRM [17] on Criteo to ensure a fair comparison.
- **Supernet Warm-up.** We observe that the supernet may collapse at initial training phases due to the varying sampled paths and uninitialized embedding layers. To mitigate the initial collapsing of supernet, we randomly sample the full supernet at the initial 1/4 of the training steps, with a probability p that linearly decays from 1 to 0. This provides dimension warm-up, operator warm-up [2] and connection warm-up for the supernet with minimal impact on the quality of sampled paths.
- **Supernet Training Settings.** We insert layer normalization [1] into each building operator to normalize varying subnet statistics during supernet training. Our choice of hyperparameters is robust over different NASRec search spaces and recommender system benchmarks. We train the supernet for only 1 epoch with Adagrad optimizer, an initial learning rate of 0.04, a cosine learning rate schedule [16], and a batch size 256 to perform sufficient path sampling on target recommender system benchmarks.

Finally, we present the details of regularized evolution and model selection strategies over NASRec search spaces.

- **Regularized Evolution.** Despite the large size of *NASRec-Full* and *NASRec-small*, we employ an efficient configuration of regularized evolution to seek the optimal subnets from supernet. Specifically, we maintain a population of 64 architectures and run regularized evolution for 100 iterations. In each iteration, we first pick up the best architecture from 32 sampled architectures from the population as the parent architecture, and generate 16 child architectures to update the population. Following the aforementioned "Adaptive Mutation Strategy", each child architecture is derived from the parent architecture with 5 mutations. We decay the number of mutations by 1 every 20 iterations.
- **Model Selection.** We follow the evaluation protocols in AutoCTR [23] and split each target dataset into 3 sets: training (80%), validation (10%) and testing (10%). During weight-sharing neural architecture search, we train the supernet on the training set and select the top-15 subnets that has the best performance on the validation set. To obtain the best subnet, we train the top-15 models from scratch with reduced embedding table size, and select the subnet with best validation performance. Finally, we evaluate the best subnet with full embedding table by training it from scratch, and computing the performance metrics (e.g., Log Loss, AUC) on test split.

5.2 Recommender System Benchmark Results

We select baselines from both hand-crafted state-of-the-arts (SOTA) and automatically designed models for comparison. We follow the

Table 4: Performance of NASRec on General CTR Predictions Tasks.

	Method	Criteo		Avazu		KDD Cup 2012		Search Cost
		Log Loss	AUC	Log Loss	AUC	Log Loss	AUC	(GPU days)
Hand-crafted Arts	DLRM [17]	0.4436	0.8085	0.3814	0.7766	0.1523	0.8004	-
	xDeepFM [14]	0.4418	0.8052	-	-	-	-	-
	AutoInt+ [24]	0.4427	0.8090	0.3813	0.7772	0.1523	0.8002	-
	DeepFM [10]	0.4432	0.8086	0.3816	0.7767	0.1529	0.7974	-
NAS-crafted Arts	DNAS [13]	0.4442	-	-	-	-	-	-
	PROFIT [8]	0.4427	0.8095	**0.3735**	0.7883	-	-	~0.5
	AutoCTR [23]	0.4413	0.8104	0.3800	0.7791	0.1520	0.8011	~0.75
	Random Search @ *NASRec-Small*	0.4411	0.8105	0.3748	0.7885	0.1500	0.8123	1.0
	Random Search @ *NASRec-Full*	0.4418	0.8098	0.3767	0.7853	0.1509	0.8071	1.0
	NASRec @ *NASRec-Small*	**0.4407**	**0.8109**	0.3757	0.7872	**0.1494**	**0.8141**	~0.25
	NASRec @ *NASRec-Full*	**0.4409**	**0.8107**	**0.3739**	**0.7900**	0.1498	0.8117	~0.3

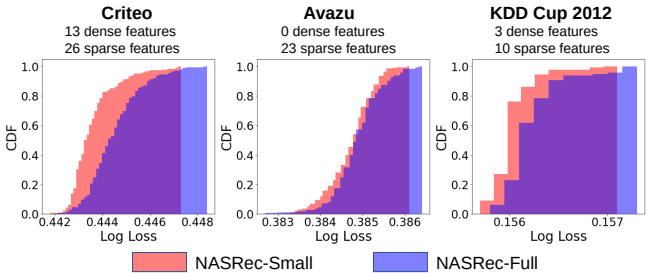

Figure 5: Cumulative Distribution Function (CDF) comparison of randomly sampled architectures from different NASRec search spaces.

data preprocessing protocol in AutoCTR [23] and report their reproduced baseline results for fair comparison. We train the best model discovered by NASRec from scratch on three classic recommender system benchmarks, and compare the performance (i.e., Log Loss, AUC) of models that are crafted by NASRec on three general recommender system benchmarks. In Table 4, we report the evaluation results of our end to end NASRec-crafted models and a random search baseline which randomly samples and trains models in our NASRec search space.

NASRec Revolutionizes Hand-crafted Recommender System Models. Even within an aggressively large *NASRec-Full* search space, NASRec achieves record-breaking performance over hand-crafted CTR models [10, 14, 17] with minimal human priors as shown in Table 4. Compared with AutoInt [24], the hand-crafted SOTA that fabricates feature interactions with delicate engineering efforts, NASRec achieves ~ 0.002 Log Loss reduction on Criteo, ~ 0.007 Log Loss reduction on Avazu, and ~ 0.003 Log Loss reduction on KDD Cup 2012. Thus, NASRec revolutionizes the design of CTR models: it shows promising prospects on improving full architecture search on recommender systems, with minimal human expertise and interventions.

NASRec is the SOTA NAS method. Even compared to the most recent NAS-crafted models [23], NASRec achieves the state-of-the-art (SOTA) Log Loss and AUC on all three recommender system benchmarks. With the same scale of search space as AutoCTR (i.e., *NASRec-Small* search space), NASRec yields better performance on all three benchmarks: it achieves 0.0006 Log Loss reduction on Criteo, 0.004 Log Loss reduction on Avazu, and 0.003 Log Loss reduction on KDD Cup 2012. This demonstrates the generic superiority of NASRec towards different recommendation system applications. Compared to DNAS [13] and PROFIT [8] which only focuses on configuring part of the architectures, such as dense connectivity and raw feature interactions, NASRec achieves at least ~ 0.002 Log Loss reduction on Criteo and justifies the significance of full architecture search on recommender system benchmarks. By extending NASRec to an extremely large *NASRec-full* search space, the best discovered NAS models further improves its result on Avazu and out-performs PROFIT by 0.002 AUC improvement with on-par Log Loss, justifying the design of *NASRec-full* with aggressively large cardinality and minimal human priors. On Criteo and KDD Cup 2012, NASRec maintains the edge in discovering state-of-the-art CTR models compared to existing NAS methods [8, 13, 23].

Efficient Search within a Versatile Search Space. Despite a larger NASRec search space that presents more challenges to fully explore, NASRec achieves at least 1.7× searching efficiency compared to state-of-the-art efficient NAS methods [8, 23] with significant Log Loss improvement on all three benchmarks. This is greatly attributed to the efficiency of Weight-Sharing NAS applied on heterogeneous operators and multi-modality data.

We also establish a random search baseline on both *NASRec-Full* and *NASRec-Small*. We first plot the Cumulative Distribution Function in *NASRec-small*. A higher density of top-performing architectures exists compared to *NASRec-full*, making random search relatively easier to obtain top-performing architectures within a limited budget. However, as the cardinality of the search space increases, random search obtains worse performance due to a lower density of top-performing architectures, see Figure 5.

Despite a lower random search baseline, the scalable Weight-Sharing NAS tackles the exploration of full *NASRec-Full* search space thanks to the broad coverage of the supernet. With an effective Single-Operator Any-connection path sampling strategy, Weight-Sharing NAS narrows the gap of random search on Criteo and KDD Cup 2012, yet surprisingly discovers a significantly better model on Avazu. This indicates that a larger NASRec search space has the potential to explore more architecture possibilities and obtain better models for recommendation system designs, and justify our usage of the flexible and hierarchical *NASRec-Full* search space.

Table 5: Model Complexity Analysis.

Method	Log Loss			FLOPS(M)		
	Criteo	Avazu	KDD	Criteo	Avazu	KDD
DLRM	0.4436	0.3814	0.1523	26.92	18.29	25.84
DeepFM	0.4432	0.3816	0.1529	22.74	22.50	21.66
AutoInt+	0.4427	0.3813	0.1523	18.33	17.49	14.88
AutoCTR	0.4413	0.3800	0.1520	12.31	7.12	3.02
NASRec @ *NASRec-Small*	**0.4407**	0.3757	**0.1494**	1.64	2.13	**0.76**
NASRec @ *NASRec-Full*	0.4409	**0.3739**	0.1498	**1.50**	**1.70**	0.96

5.3 Complexity Analysis of Crafted Models.

We compare the model complexity of NASRec with SOTA handcrafted models and SOTA NAS models. We collect all baselines from AutoCTR [23], and list performance versus the number of theoretical Floating-point Operations (FLOPs) in Table 5. We profile all FLOPS of NASRec models using FvCore[5].

Even without any FLOPs constraints, the best crafted NASRec models surprisingly out-perform existing arts in efficiency. Despite achieving lower Log Loss, our NAS-crafted model achieves 8.2×, 4.2×, and 3.1× FLOPS reduction on Criteo, Avazu, and KDD Cup 2012 benchmarks. One reason is our Single-operator Any-connection sampling strategies favor medium-sized models to trade off model cost and performance as introduced in Section 4.1. One possible reason lies in the use of operator-balancing interaction modules projects the large number of sparse inputs to a smaller dimension before carrying cross-term feature interaction. This leads to significantly lower computation costs. As a result, our Weight-Sharing NAS enables discovering of compact yet high-performing models, and greatly contributes efficient models towards recommender systems.

6 CONCLUSION

In this paper, we propose NASRec, a new paradigm to fully enable **NAS** for **rec**ommender systems via Weight Sharing Neural Architecture Search (WS-NAS) under data modality and architecture heterogeneity. NASRec establishes a large supernet to represent the full architecture space, and incorporates versatile building operators and dense block connections to minimize human priors in automated architecture design for recommender systems. NASRec identifies the scale and heterogeneity challenges of large-scale NASRec search space that compromises supernet, and proposes a series of techniques to improve training efficiency and mitigate ranking disorder. NASRec achieves state-of-the-art performance on 3 popular recommender system benchmarks, shows promising prospects on full architecture search space and directs motivating research towards fully automated architecture fabrication with minimal human priors.

7 ACKNOWLEDGEMENTS

Yiran Chen's work is partially supported by the following grants: NSF-2120333, NSF-2112562 and NSF-1937435. Feng's work is partially supported by the following grants: NSF CAREER-2048044 and IIS-1838024. We also thank Maxim Naumov, Jeff Hwang and Colin Taylor in Meta Platforms, Inc. for their kind help on this project.

[5]https://github.com/facebookresearch/fvcore

REFERENCES

[1] Jimmy Lei Ba, Jamie Ryan Kiros, and Geoffrey E Hinton. 2016. Layer normalization. *arXiv preprint arXiv:1607.06450* (2016).
[2] Gabriel Bender, Hanxiao Liu, Bo Chen, Grace Chu, Shuyang Cheng, Pieter-Jan Kindermans, and Quoc V Le. 2020. Can weight sharing outperform random architecture search? an investigation with tunas. In *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition*. 14323–14332.
[3] Han Cai, Chuang Gan, Tianzhe Wang, Zhekai Zhang, and Song Han. 2019. Once-for-all: Train one network and specialize it for efficient deployment. *arXiv preprint arXiv:1908.09791* (2019).
[4] 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.
[5] 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.
[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] Alexey Dosovitskiy, Lucas Beyer, Alexander Kolesnikov, Dirk Weissenborn, Xiaohua Zhai, Thomas Unterthiner, Mostafa Dehghani, Matthias Minderer, Georg Heigold, Sylvain Gelly, et al. 2020. An image is worth 16x16 words: Transformers for image recognition at scale. *arXiv preprint arXiv:2010.11929* (2020).
[8] Chen Gao, Yinfeng Li, Quanming Yao, Depeng Jin, and Yong Li. 2021. Progressive Feature Interaction Search for Deep Sparse Network. *Advances in Neural Information Processing Systems* 34 (2021).
[9] Luyu Gao, Zhuyun Dai, and Jamie Callan. 2020. Modularized transfomer-based ranking framework. *arXiv preprint arXiv:2004.13313* (2020).
[10] 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).
[11] 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.
[12] Xinran He, Junfeng Pan, Ou Jin, Tianbing Xu, Bo Liu, Tao Xu, Yanxin Shi, Antoine Atallah, Ralf Herbrich, Stuart Bowers, et al. 2014. Practical lessons from predicting clicks on ads at facebook. In *Proceedings of the eighth international workshop on data mining for online advertising*. 1–9.
[13] Ravi Krishna, Aravind Kalaiah, Bichen Wu, Maxim Naumov, Dheevatsa Mudigere, Misha Smelyanskiy, and Kurt Keutzer. 2021. Differentiable NAS Framework and Application to Ads CTR Prediction. *arXiv preprint arXiv:2110.14812* (2021).
[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] Hanxiao Liu, Karen Simonyan, and Yiming Yang. 2018. Darts: Differentiable architecture search. *arXiv preprint arXiv:1806.09055* (2018).
[16] Ilya Loshchilov and Frank Hutter. 2016. Sgdr: Stochastic gradient descent with warm restarts. *arXiv preprint arXiv:1608.03983* (2016).
[17] Maxim Naumov, Dheevatsa Mudigere, Hao-Jun Michael Shi, Jianyu Huang, Narayanan Sundaraman, Jongsoo Park, Xiaodong Wang, Udit Gupta, Carole-Jean Wu, Alisson G Azzolini, et al. 2019. Deep learning recommendation model for personalization and recommendation systems. *arXiv preprint arXiv:1906.00091* (2019).
[18] 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.
[19] Steffen Rendle, Zeno Gantner, Christoph Freudenthaler, and Lars Schmidt-Thieme. 2011. Fast context-aware recommendations with factorization machines. In *Proceedings of the 34th international ACM SIGIR conference on Research and development in Information Retrieval*. 635–644.
[20] 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.
[21] Ying Shan, T Ryan Hoens, Jian Jiao, Haijing Wang, Dong Yu, and JC Mao. 2016. Deep crossing: Web-scale modeling without manually crafted combinatorial features. In *Proceedings of the 22nd ACM SIGKDD international conference on knowledge discovery and data mining*. 255–262.
[22] David So, Quoc Le, and Chen Liang. 2019. The evolved transformer. In *International Conference on Machine Learning*. PMLR, 5877–5886.
[23] 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.

[24] 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.

[25] Ashish Vaswani, Noam Shazeer, Niki Parmar, Jakob Uszkoreit, Llion Jones, Aidan N Gomez, Łukasz Kaiser, and Illia Polosukhin. 2017. Attention is all you need. *Advances in neural information processing systems* 30 (2017).

[26] Hanrui Wang, Zhanghao Wu, Zhijian Liu, Han Cai, Ligeng Zhu, Chuang Gan, and Song Han. 2020. Hat: Hardware-aware transformers for efficient natural language processing. *arXiv preprint arXiv:2005.14187* (2020).

[27] 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.

[28] Ruoxi Wang, Rakesh Shivanna, Derek Cheng, Sagar Jain, Dong Lin, Lichan Hong, and Ed Chi. 2021. DCN V2: Improved deep & cross network and practical lessons for web-scale learning to rank systems. In *Proceedings of the Web Conference 2021*. 1785–1797.

[29] Wei Wen, Hanxiao Liu, Yiran Chen, Hai Li, Gabriel Bender, and Pieter-Jan Kindermans. 2020. Neural predictor for neural architecture search. In *European Conference on Computer Vision*. Springer, 660–676.

[30] Jiahui Yu, Pengchong Jin, Hanxiao Liu, Gabriel Bender, Pieter-Jan Kindermans, Mingxing Tan, Thomas Huang, Xiaodan Song, Ruoming Pang, and Quoc Le. 2020. Bignas: Scaling up neural architecture search with big single-stage models. In *European Conference on Computer Vision*. Springer, 702–717.

[31] Barret Zoph, Vijay Vasudevan, Jonathon Shlens, and Quoc V Le. 2018. Learning transferable architectures for scalable image recognition. In *Proceedings of the IEEE conference on computer vision and pattern recognition*. 8697–8710.