RESEARCH-ARTICLE

Towards Automated Model Design on Recommender Systems

TUNHOU ZHANG, Duke University, Durham, NC, United States

DEHUA CHENG, Meta Platforms, Inc., Menlo Park, CA, United States

YUCHEN HE, Meta Platforms, Inc., Menlo Park, CA, United States

ZHENGXING CHEN, Meta Platforms, Inc., Menlo Park, CA, United States

XIAOLIANG DAI, Meta Platforms, Inc., Menlo Park, CA, United States

LIANG XIONG, Meta Platforms, Inc., Menlo Park, CA, United States

View all

Towards Automated Model Design on Recommender Systems

TUNHOU ZHANG, Duke University, Durham, United States
DEHUA CHENG, Meta Platforms Inc, Menlo Park, United States
YUCHEN HE, Meta Platforms Inc, Menlo Park, United States
ZHENGXING CHEN, Meta Platforms Inc, Menlo Park, United States
XIAOLIANG DAI, Meta Platforms Inc, Menlo Park, United States
LIANG XIONG, Meta Platforms Inc, Menlo Park, United States
YUDONG LIU, Duke University, Durham, United States
FENG CHENG, Duke University, Durham, United States
YUFAN CAO, Duke University, Durham, United States
FENG YAN, University of Houston, Houston, United States
HAI LI, Duke University, Durham, United States
YIRAN CHEN, Duke University, Durham, United States
WEI WEN, Meta Platforms Inc, Menlo Park, United States

The increasing popularity of deep learning models has created new opportunities for developing artificial intelligence–based recommender systems. Designing recommender systems using deep neural networks (DNNs) requires careful architecture design, and further optimization demands extensive co-design efforts on jointly optimizing model architecture and hardware. Design automation, such as Automated Machine Learning (AutoML), is necessary to fully exploit the potential of recommender model design, including model choices and model–hardware co-design strategies. We introduce a novel paradigm that utilizes weight sharing to explore abundant solution spaces. Our paradigm creates a large supernet to search for optimal architectures and co-design strategies to address the challenges of data multimodality and heterogeneity in the recommendation domain. From a model perspective, the supernet includes a variety of operators, dense connectivity, and dimension search options. From a co-design perspective, it encompasses versatile Processing-In-Memory (PIM) configurations to produce hardware-efficient models. Our solution space's scale, heterogeneity, and

complexity pose several challenges, which we address by proposing various techniques for training and evaluating the supernet. Our crafted models show promising results on three Click-Through Rate (CTR) prediction benchmarks, outperforming both manually designed and AutoML-crafted models with state-of-the-art performance when focusing solely on architecture search. From a co-design perspective, we achieve 2× floating-point operations efficiency, 1.8× energy efficiency, and 1.5× performance improvements in recommender models.

CCS Concepts: • **Information systems → Recommender systems**; • **Computing methodologies →** **Discrete space search**; **Neural networks**;

Additional Key Words and Phrases: Recommendation Systems, AutoML, Neural Architecture Search, Evolutionary Search

1 Introduction

Recommender systems, which are widely used in search engines and social media platforms [6, 19] to optimize **Click-Through Rates (CTR)**, rely on deep learning-based models [9, 15, 26] that incorporate multimodality features. However, these models present challenges in feature interaction modeling and neural network optimization due to the heterogeneity of the features. Finding a good backbone model with appropriate priors on multimodality features is standard practice, but it requires significant manual effort [8, 16, 17, 21, 26, 28, 30, 31] and is limited by available resources.

Automated Machine Learning (AutoML) techniques, such as **Weight-Sharing Neural** **Architecture Search (WS-NAS)** [5, 23, 46], have shown promise in optimizing the design of efficient models for recommender systems without human intervention. NAS seeks the best architecture choice within an abundant solution space, with versatile search strategies [20, 25, 50] and evaluation strategies of models. However, these techniques face unique challenges owing to the multimodality and heterogeneity of data and architecture in recommendation systems compared with vision models. One challenge is that the inputs to building blocks in recommendation systems are multimodal and generate **two-dimensional (2D)** and **three-dimensional** **(3D)** tensors, whereas vision models have homogeneous 3D tensors [5, 37] as inputs. Additionally, while state-of-the-art NAS in vision converges to searching size configurations, recommendation models are heterogeneous, with each stage using a different building block. Furthermore, recommendation models use a variety of heterogenous operators, such as Fully-Connected layer [8], Dot-Product [26], Multi-Head Attention [34], and Factorization Machine [16, 21]. In contrast, vision models mainly use homogenous convolutional operators and backbone search on design motifs [5, 46], such as layer width, layer depth, and kernel size. These challenges are further complicated by co-design tasks such as mixed-precision quantization search, which can worsen the quality of the product driven by AutoML. Overall, there is a need for more effective AutoML techniques that can handle the unique challenges of data multimodality and architecture heterogeneity in recommendation systems.

Owing to the challenges above, the study of AutoML in recommender systems is limited. For example, search spaces in AutoCTR [33] and DNAS [20] follow the design principle of human-crafted **deep learning recommendation models (DLRMs)** [26], and they only include **Fully**

Table 1. Comparison of Our Approach Versus Existing AutoML Methods for Recommender Systems from a **Model** Perspective

Method	Building Operators?	Dense Connectivity?	Full arch Search?	Co-design Support?	Criteo Log Loss
DNAS [20]	FC, Dot-Product	✓			0.4442
PROFIT [13]	FC, FM				0.4427
AutoCTR [33]	FC, Dot-Product, FM, EFC	✓	✓		0.4413
Ours	FC, Gating, Sum, Attention, Dot-Product, FM, EFC	✓	✓	✓	**0.4399**

Connected (FL) layers and Dot-Products as searchable operators. They also heavily rely on manually crafted operators, such as Factorization Machines [33] or feature interaction modules [13] in the search space to increase architecture heterogeneity. Moreover, existing works suffer from huge computation costs [33] or challenging bilevel optimization [20]. Thus, they only employ narrow design spaces (sometimes with strong human priors [13]) to craft architectures, discouraging diversified feature interactions and harming the quality of crafted models.

We propose a full-stack solution paradigm to fully enable Neural Architecture and Parameter Search via WS-NAS under data modality and architecture heterogeneity. We offer end-to-end solutions on model design and hardware co-design strategies. From a model perspective, we summarize the advancement of our proposed paradigm over other NAS approaches in Table 1. We achieve this by building a supernet incorporating much more heterogeneous operators than previous works, including an FC layer, Gate, Sum, Dot-Product, Self-Attention, and **Embedded Fully Connected (EFC)** layer. In the supernet, we densely connect a cascade of blocks, each including all operators as options. As dense connectivity allows any block to take in any raw feature embeddings and intermediate tensors, the supernet is not limited by any particular data modality. This supernet design minimizes the encoding of human priors [47], supporting the nature of data modality and architecture heterogeneity in recommenders and covering models beyond popular recommendation models such as Wide & Deep [8], DeepFM [16], DLRM [26], AutoCTR [33], DNAS [20], and PROFIT [13]. Our approach also supports the co-design of model and hardware (e.g., Processing-In-Memory architecture [45]), providing headroom for improvement when deploying recommender models in reality. The ad-hoc analysis of structured pruning on our crafted models further expands the opportunity to improve the efficiency of recommender models.

The supernet essentially 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. All subnets share weights from the same supernet, called **Weight Sharing NAS (WS-NAS)**. To efficiently search models/subnets in the search space, we advance one-shot approaches [5, 46] to the recommendation domain. We propose *Single-operator Any-connection sampling* to decouple operator selections and increase connection coverage, *operator-balancing interaction* blocks to train subnets in the supernet fairly, and *post-training fine-tuning* to reduce weight co-adaptation. These approaches enable a better training efficiency and ranking of subnet models in the supernet, resulting in ~0.001 log loss reduction of searched models on full NASRec search space. We further apply the search on model-hardware co-design and study ad hoc structured pruning, unlocking extra benefits in the efficiency of recommender models.

From a model perspective, we evaluate our AutoML-crafted models on three popular CTR benchmarks and demonstrate significant improvements compared with hand-crafted and NAS-crafted models. Remarkably, our approach advances the state-of-the-art with log loss reduction of ~0.001 and ~0.003 on Criteo and KDD Cup 2012, respectively. On Avazu, our approach advances

the state-of-the-art PROFIT [13] with **area under the curve (AUC)** improvement of ∼0.002 and on-par log loss while outperforming PROFIT [13] on Criteo by ∼0.003 log loss reduction. Thanks to the efficient weight-sharing mechanism, our approach only needs to train a single supernet, greatly reducing the search cost. From a co-design perspective, we offer a detailed analysis to exploit the potential of our crafted models and uncover 1.5× theoretical speedup on discovered models. The ad-hoc structured pruning achieves ∼2× **floating point operations per second (FLOPs)** savings without harming log loss and AUC on recommender benchmarks.

The rest of the article is organized as follows. Section 2 introduces the related work in recommender systems. Section 3 elaborates on the search space from both the model and co-design perspectives. In Section 4, we propose the search methodology and demonstrate the main technologies. In Section 5, we evaluate our crafted models on 3 CTR benchmarks, demonstrating state-of-the-art performance and uncovering theoretical efficiency gain from model–hardware co-design. In Section 6, we provide ablation studies and discussions to better understand our system and methodologies, including structured pruning to advance the model efficiency. In Section 7, we present our conclusion. We summarize our major contributions below.

— We propose a new paradigm to scale up the automated design of recommender systems from both the model and co-design perspectives. NASRec establishes a flexible supernet (search space) with minimal human priors, overcoming data modality and architecture heterogeneity challenges in the recommendation domain.
— We advance weight-sharing NAS to the recommendation domain by introducing single-operator any-connection sampling, operator-balancing interaction modules, and post-training fine-tuning.
— From a model perspective, our crafted models outperform hand-crafted and AutoML-crafted models with a smaller search cost.
— From a co-design perspective, we explore various choices to co-design model architecture with Processing-In-Memory hardware, demonstrating significant speed-up headroom.

2 Related Work

Deep learning–based recommender systems. Machine-based recommender systems, such as those predicting CTRs, have been extensively studied using various approaches, such as Logistic Regression [30], Gradient-Boosting Decision Trees [17], Wide & Deep Neural Networks [8], crossing networks [31], Factorization Machines [16, 21], Dot-Product [26], and gating mechanisms [38, 39, 44]. Additionally, researchers have explored efficient feature interactions via feature-wise multiplications [42] and sparsifications [10] to develop lightweight recommender systems. However, these methods require significant manual effort and may result in suboptimal performance owing to limited resource availability and constrained design choices. Our work introduces a novel paradigm for learning effective recommender models, including novel model architecture search space and effective model hardware co-design via Processing-In-Memory hardware and mixed-precision quantization.

AutoML and NAS. AutoML has gained significant popularity in automating the design of deep neural networks across various applications such as Computer Vision [5, 23, 40, 43, 50], Natural Language Processing [32, 37], and Recommendation Systems [13, 20, 33]. NAS, especially WS-NAS [5, 37], has recently garnered attention owing to its ability to train a supernet representing the entire search space directly on target tasks and efficiently evaluate subnets with shared supernet weights. However, applying WS-NAS to recommender systems is challenging because these systems involve heterogeneous architectures dedicated to interacting with multimodality data, requiring more flexible search spaces and effective supernet training algorithms. These

challenges lead to co-adaptation [3] and operator-imbalance problems [22] in WS-NAS, resulting in lower rank correlation for distinguishing models. To address these issues, our work introduces a series of technical solutions: single-operator any-connection sampling, operator-balancing interaction modules, and post-training fine-tuning. In addition, our work considers both joint architecture–hardware search and ad-hoc mixed-precision exploration to enhance discovered models, providing novel perspectives and insights on recommender system model designs.

Software–hardware Co-design. Classic software–hardware co-design works mainly focus on the joint optimization of the model architecture and hardware execution throughput [5, 37]. In this work, we explore **processing-in-memory (PIM)** architectures and discuss the co-design of recommender models with PIM architectures for real-world applications. We also connect PIM optimization with model compression techniques such as pruning and quantization. PIM uses crossbar-based structures in advanced memory technologies such as **Resistive Random-Access Memory (ReRAM)** [45]. Prior research in this field [41] has explored using PIM's inherent parallel processing capabilities to enhance the performance of recommender systems. However, many of these studies have not fully addressed the unique challenges posed by PIM-based recommender systems, such as uneven cache access patterns, inefficient mapping strategies, and suboptimal heuristic-based design methodologies. Further exploration in this field [35] reveals that varying configurations of PIM—such as the size of crossbars, the precision of **Analog-to-Digital Converters (ADCs)** and **Digital-to-Analog Converters (DACs)**, and the resolution of crossbars—can significantly influence key performance metrics such as accuracy and energy consumption. Our work incorporates the optimization of PIM design into model architecture search and demonstrates the initiative to craft hardware-friendly models for recommendation and user personalization.

3 Hierarchical Search Space for Model and Co-design

We first introduce the hierarchical search space design dedicated to architecture search from a model perspective. We propose our resolutions to craft more hardware-friendly models from a co-design perspective. This complements our vision to start AutoML research and realize an end-to-end recommender model design paradigm for social good.

3.1 Model Architecture Search Space

Thus, we first demonstrate the model architecture space design by revisiting NASRec [47]. The flexibility of search space is the key to supporting data modality and architecture heterogeneity in recommender systems. The major manual process in designing the search space is simply collecting common operators used in existing approaches [16, 21, 26, 33, 38, 39]. Beyond that, we further incorporate the prevailing Transformer Encoder [36] into the search space for better flexibility and higher potential in searched architectures thanks to its dominance in applications such as ViT [11] for image recognition, Transformer [36] for natural language processing, and its emerging exploration in recommender systems [7, 14].

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 EFC 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 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 one that produces a dense (sparse) output. In NASRec, dense operators include FC, Gating,
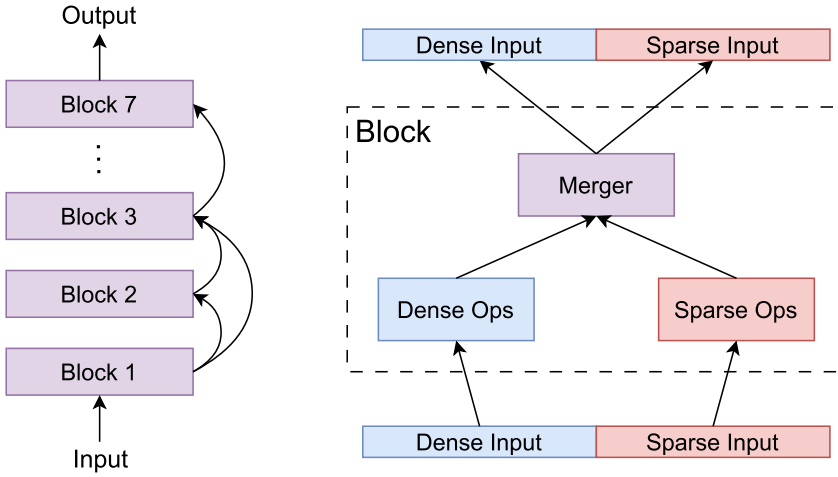
Fig. 1. NASRec search space enables a full architecture search on dense connectivity of blocks, dense/sparse operators, and mergers that fuse dense/sparse representations.

Sum, and Dot-Product, which form the "dense branch"; sparse operators include EFC and self-attention, which form the "sparse branch".

A candidate architecture in NASRec search space is a stack of N choice blocks, followed by a final FC layer to compute the final logit. Each choice block admits an arbitrary number of multimodality inputs, each of which is $X = (X_d, X_s)$ from a previous block or raw inputs, and produces a multimodality 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 representing the NASRec search space (see Figure 1). The supernet subsumes all possible candidate models/subnets and performs weight sharing among subnets to train them simultaneously. 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 operator $O = [O^{(1)}, \ldots, O^{(N)}]$ enumerates the set of building operators from choice block 1 to N. The connection $C = [C^{(1)}, \ldots, C^{(N)}]$ contains the connectivity $< i, j >$ between choice block i and choice block j. The dimension $\mathcal{D} = [D^{(1)}, \ldots, D^{(N)}]$ 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 zero masking to mask 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 $\{Y^{(1)}, \ldots, Y^{(N-1)}\}$ and the sampled connections $C_{sample}^{(N)}$:

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

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

Here, $\mathbf{1}_b$ is 1 when b is true, otherwise 0.

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

$$Y_d^{(N)} = \sum_{o \in O} \mathbf{1}_{o \in O_{sample}^{(N)}} \cdot Mask\left(o\left(X_d^{(N)}\right), D_{sample,o}^{(N)}\right), \tag{3}$$

where

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

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

— **Fully Connected (FC) layer.** The connected layer is the backbone of DNN models for recommender systems [8] that extracts dense representations. The FC layer is applied on 2D dense inputs and followed by rectified linear unit (ReLU) activation.

— **Sigmoid Gating (SG) layer.** We follow the intuition in [7, 39] and employ a dense building operator, Sigmoid Gating, to enhance the potential of the search space. Given two dense inputs $X_{d1} \in \mathbb{R}^{B \times dim_{d1}}$ and $X_{d2} \in \mathbb{R}^{B \times dim_{d2}}$, Sigmoid Gating interacts these two inputs as follows: $SG(X_{d1}, X_{d2}) = sigmoid(FC(X_{d1})) * X_{d2}$. If the dimensions of two dense inputs do not match, zero padding is applied to the input with the lower dimension.

— **Sum layer.** This dense building operator adds two dense inputs—$X_{d1} \in \mathbb{R}^{B \times dim_{d1}}$, $X_{d2} \in \mathbb{R}^{B \times dim_{d2}}$—and merges two features from different levels of the recommender system models by simply performing $Sum(X_{d1}, X_{d2}) = X_{d1} + X_{d2}$. Like Sigmoid Gating, zero padding is applied on the input with a lower dimension.

— **Dot-Product (DP) layer.** We leverage Dot-Product to grasp the interactions among multimodality inputs via a pairwise inner product. Dot-Product can take dense and/or sparse inputs and produce a dense output. After being sent to the "dense branch," these sparse inputs can later use 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 concatenates them as $X = Concat[X_d, X_s]$ and then performs pairwise inner products: $DP(X_d, X_s) = Triu(XX^T)$. dim_d is first projected to dim_s if they do not match.

— **Embedded Fully Connected (EFC) layer**. An EFC layer is a sparse building operator that applies an FC layer along the middle dimension. Specifically, an EFC layer 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 $Y_s \in \mathbb{R}^{B \times N_{out} \times dim_s}$.

— **Attention (Attn) layer.** Attention layer is a sparse building operator that utilizes the **Multi-Head Attention (MHA)** mechanism to learn the weighting of sparse inputs and better exploit their interaction in recommendation systems. Here, we apply Transformer Encoder on a given sparse input $X_s \in \mathbb{R}^{B \times N_s \times dim_s}$, with identical queries, keys, and values.

We observe that the aforementioned set of building operators provides 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 introduce the "**dense-sparse merger**" to allow dense/sparse outputs to merge into the "sparse/dense branch"optionally. The dense-sparse merger contains two major components.

— "Dense-to-sparse" merger. This merger first projects the dense outputs X_d using an FC layer, then uses a reshape layer to reshape the projection into a 3D sparse tensor. The reshaped 3D tensor is merged into the sparse output via concatenation.
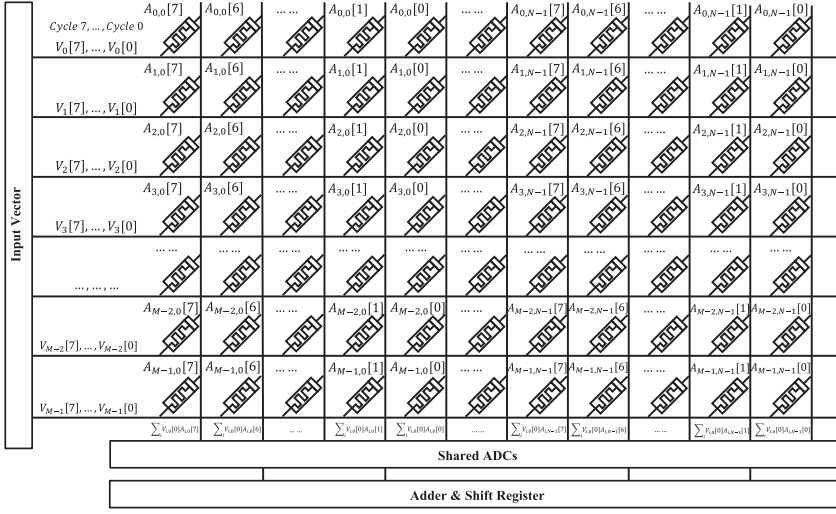
Fig. 2. Operation principle of ReRAM-based in-memory computing.

— "Sparse-to-dense" merger. This merger employs a **Factorization Machine (FM)** [16] to convert the sparse output into a dense representation and then adds the dense representation to the dense output.

Beyond the rich choices of building operators and mergers, each choice block can receive inputs from any preceding choice blocks and raw input features. This involves exploring any connectivity among choice blocks and raw inputs, extending the wiring heterogeneity for search.

3.2 Co-design Model Architecture and Hardware

Next, we consider further enhancing the co-design of model architecture and hardware. Figure 2 illustrates an overview of PIM hardware design, representing a significant innovation in computing. These designs apply analog voltages to each **Word Line (WL)**, initiating a process in which these voltages are multiplied by the conductance present in each row. This multiplication adheres to the principles of Ohm's Law. The currents produced from this multiplication are then combined along each column according to Kirchhoff's Current Law. At the end of each **Bit Line (BL)**, specialized circuitry interprets these aggregated currents to facilitate complex **Matrix–Vector Multiplication (MVM)** functions within the memory array.

As such, we propose integrating ReRAM-related parameters into the search space from a co-design perspective. This enables simultaneous and efficient co-exploration of the recommender system model architecture and the PIM architecture. We map the building above operators onto the crossbars with minimal effort for straightforward evaluation. Operators like EmbedFC, FC, and the dense-to-sparse merger are intrinsically MVM and follow the mapping protocol outlined in the background section. Dot-Product and sparse-to-dense-merger (i.e., FM [16]), not ideal for PIM, are assigned to the digital functional unit instead.

Our vision adopts mixed-precision search as a preliminary study for PIM hardware co-design, such as crossbar optimization and ReRAM optimization. This is because quantization provides headroom analysis in ReRAM design towards hardware metrics such as DAC resolution, memristor precision, aADC resolution, etc. We provide theoretical analysis and demonstrate the simulation results on hardware, providing concrete guidance on the theoretical headroom of co-designing software and hardware for recommender models.

3.3 Search Components

In the NASRec search space dedicated to model architectures, we search for each choice block's connectivity, operator dimensions, and building operators. We illustrate the three key search components as follows:

- **Connection.** We place no restrictions on the number of connections 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 EFC to dim_d and N_s, respectively, and other operator outputs in the dense (sparse) branch are linearly projected to dim_d (N_s). This ensures that operator outputs in each branch have the same dimension and can be added 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 Equation (4) zeros out the extra dimensions, allowing a flexible selection of building operators' dimensions.
- **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 validated candidate architecture. In addition, we independently sample dense-sparse mergers to allow optional dense-to-sparse interaction.

We showcase two model architecture search spaces as examples.

- *NASRec-Small.* We limit the choice of operators within each block to FC, EFC, and Dot-Product and allow any connectivity between blocks. This provides a similar scale of search space as AutoCTR [33].
- *NASRec-Full.* We enable all building operators, mergers, 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 branches, 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 tests the capability of NASRec to the extreme.

The combination of full dense connectivity search and independent dense/sparse dimension configuration gives the model architecture search space a large cardinality. *NASRec-Full* has $N = 7$ blocks, containing up to 5×10^{33} 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.

In addition, we construct the co-design search space as follows:

- **DNN Design Space.** The DNN design space follows the *NASRec-Small* search space dependent on the compatibility of building operators on PIM hardware. This includes dense operators such as FC and DP, with feature dimensions ranging from 64 to 1024. We also incorporate sparse operators with dimensions from 16 to 64 and dense-sparse interaction operators, including FC and FM.
- **Quantization Design Space.** We allow mapping onto all previously mentioned operators, including FC and EFC layers and FC and EFC projections inside DP and FM, but excluding DP and FM, as they are not a natural fit for ReRAM. The quantization of weights ranges from 4 to 8 bits.
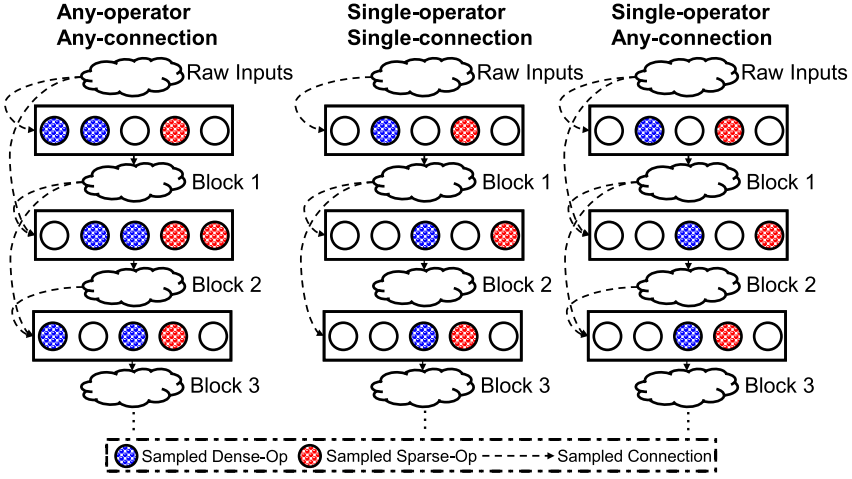
Fig. 3. Single-operator any-connection path sampling combines the advantages of the first two sampling strategies.

4 Weight-sharing Neural Architecture Search for Recommender Systems

A NASRec supernet simultaneously breeds different subnet models in the aforementioned model and co-design search space, yet its large cardinality challenges training efficiency and ranking quality. This section proposes a novel path sampling strategy, *single-operator any-connection* sampling, which combines operator sampling with a good connection sampling coverage. We further observe the operator imbalance phenomenon induced by some over-parameterized operators and tackle this issue by *operator-balancing interaction* to improve supernet ranking. Finally, we employ *post-training fine-tuning* to alleviate weight co-adaptation and utilize regularized evolution to obtain the best subnet. We also provide insights that effectively explore the best recommender models.

4.1 Single-Operator Any-Connection Sampling

During supernet training, a drop-out-like approach is adopted in which, at each mini-batch, a subnet is sampled and trained. The goal is to train subnets that can predict well the performance of models under weight sharing. The sampling strategy used is critical to achieve this goal. The sampling strategy used is critical to achieve this goal. Three path sampling strategies have been explored; single-operator any-connection sampling is the most effective among them, see Figure 3.

Single-Operator Single-Connection Strategy: This path sampling strategy, which has its roots in Computer Vision, uniformly samples a single dense and sparse operator in each choice block and a single connection as input to a block. While this strategy is efficient because it trains only a small subnet at a time, it encourages only chain-like formulations of models without extra connectivity patterns, leading to slower convergence, poor performance, and inaccurate ranking of models.

Any-Operator Any-Connection Strategy: This sampling strategy increases the coverage of sub-architectures of the supernet during subnet training by uniformly sampling an arbitrary number of dense and sparse operators in each choice block and an arbitrary number of connections to aggregate different block outputs. However, training efficiency is poor when training large subnets sampled in this way. Moreover, the co-adaptation of multiple operators within a choice block may affect the independent evaluation of subnets and lead to poor ranking quality.

Single-Operator Any-Connection: This path sampling strategy combines the strengths of the first two strategies. It samples a single dense and a single sparse operator in each choice block while allowing the sampling of an arbitrary number of connections to aggregate outputs from different choice blocks. The key insight behind this strategy is to separate the sampling of parametric operators to avoid weight co-adaptation while allowing the sampling of non-parametric connections to gain good coverage of the search space.

Here, dashed connections and operators denote a sampled path in the supernet. Compared with 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 shown in Table 5.

4.2 Operator-Balancing Interaction Modules

Recommender systems involve multimodality data with an indefinite number of inputs, such as many sparse inputs. We define operator imbalance as the imbalance of the number of weights between operators within a block. In weight-sharing NAS, operator imbalance may cause supernet training to favor operators with more weights. This will offset the gains due to poor ranking correlations of subnets: the subnet performance in the supernet may deviate from its ground-truth performance when trained from scratch. Within the NASRec search space, we identify that 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, this leads to extra $(N_s^2 \cdot dim_d/2)$ trainable weights for Dot-Products.

However, the weight consumption of such a projection layer is large given many 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. Such tremendous weight parameterization is a quadratic function of the number of sparse inputs N_s, yet other building operators have much fewer weights. For example, the number of trainable weights in EFC is a linear function of the 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 favors parameter-consuming subnets with a high concentration of Dot-Product operations, as we observed. In addition, the ignorance of heterogeneous operators other than Dot-Product provides a poor ranking of subnets, leading to suboptimal performance on recommender systems.

We insert a simple EFC as a projection layer before the Dot-Product to mitigate such over-parameterization demonstrated in 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 require 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 EFC N_s. Thus, we balance the interaction operators to allow a similar convergence rate for all building operators. We evaluate the training efficiency and ranking quality for supernets trained with/without operator-balancing interaction. Results demonstrate that operator-balancing interaction achieves 0.11 Kendall's τ improvement while reducing the search cost from 4 **graphics processing unit (GPU)** hours to only 1.5 GPU hours.
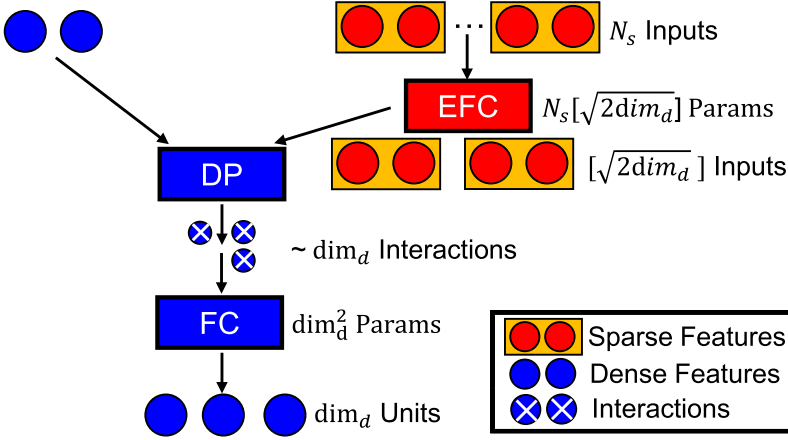
Fig. 4. Operator-balancing interaction ensures linear parameter consumption and balance building operators.

4.3 Post-training Fine-tuning

Although dropout-like subnet training can effectively reduce the adaptation of weights for a specific subnet during supernet training, it may fail when weights should not be shared across certain subnets, leading to inaccurate subnet performance predictions by the supernet. To address this issue, we propose a post-training fine-tuning technique that re-adapts the weights of each standalone subnet back to its specific configuration after supernet training. This helps to re-calibrate the corrupted weights during supernet training while training other subnets. In practice, fine-tuning only the last FC layer on the target dataset for a few training steps (e.g., 0.5K) is sufficient. This novel post-training fine-tuning technique comes with only marginal additional search cost and significantly boosts the ranking of subnets by addressing the underlying weight adaptation issue. As a result, this technique provides a better chance to discover better models for recommender systems.

Table 2 demonstrates the improvement of post-training fine-tuning on different path sampling strategies. Surprisingly, post-training fine-tuning achieves decent ranking quality improvement under the single-operator single-connection and any-operator any-connection path sampling strategies. This is because subnets under these strategies do not usually converge well in the 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, the single-operator any-connection path sampling strategy cooperates well with post-training fine-tuning and achieves the global optimal Pearson's ρ and Kendall's τ ranking correlation among different approaches, with at least 0.14 Pearson's ρ and Kendall's τ improvement on *NASRec-Full* search space over single-operator single-connection sampling with fine-tuning.

4.4 Evolutionary Search on Best Models

We utilize regularized evolution [27] 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.

Table 2. Effects of Post-training Fine-tuning on Different Path Sampling Strategies on *NASRec-Full*

Path Sampling Strategy	No Fine-tuning		Fine-tuning	
	Pearson's ρ	Kendall's τ	Pearson's ρ	Kendall's 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

We demonstrate Pearson's ρ and Kendall's τ over 100 random subnets on Criteo.

Table 3. Statistics of Different CTR Benchmarks

Benchmark	# Dense	# Sparse	# Samples (M)
Criteo	13	26	45.84
Avazu	0	23	40.42
KDD	3	10	149.64

— 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 dense-to-sparse/sparse-to-dense merger that enables the communication between dense/sparse outputs.

5 Experiments

We first show the detailed configuration that NASRec employs during the architecture search, model selection, and final evaluation. Then, we demonstrate empirical evaluations on three popular recommender system benchmarks for CTR prediction: Criteo,[1] Avazu[2] and KDD Cup 2012.[3] All three datasets are preprocessed in the same fashion as AutoCTR [33]. We release our implementation framework in NASRec. On Criteo/Avazu/KDD Cup, we observe +/− 0.0002 as the standard deviation between each run and treat 0.001 as the level of significant improvement.

We show the statistics of each CTR benchmark in Table 3.

Here, we observe that Criteo has the most dense (sparse) features and thus is the most complex and challenging benchmark. Avazu contains only dense features, thus requiring fewer interactions between dense outputs in each choice block. KDD has the least number of features and the most data, making it a relatively easier benchmark to train and evaluate.

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. This allows a fair comparison with recent NAS methods [33]. 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, and Dot-Product, and 2 distinct sparse building operators, EFC and Attention. The dense-sparse merger option is fully explored.

---

[1] https://www.kaggle.com/c/criteo-display-ad-challenge
[2] https://www.kaggle.com/c/avazu-ctr-prediction/data
[3] https://www.kaggle.com/c/kddcup2012-track2/data

— **Dimension Search Components.** For each dense building operator, the dense output dimension can be chosen from {16, 32, 64, 128, 256, 512, 768, 1024}. The sparse output dimension can be chosen from {16, 32, 48, 64} for each sparse building operator.

— **Quantization Search Components.** For each dense/sparse building operator, we perform weight/activation quantization of 4/8 bits for each building operator. This provides 16,384 extra search complexity for a $N = 7$ block search space.

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, EFC. Then, we illustrate some techniques for 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. During the final evaluation, we maintain the full embedding table to retrieve the best performance, i.e., 540 M parameters in the DLRM [26] on Criteo to ensure a fair comparison.

— **Supernet Warm-up.** The supernet may collapse at initial training phases due to the varying sampled paths and uninitialized embedding layers. To mitigate the supernet's initial collapse, we randomly sample the full supernet at the initial 1/5 of the training steps, with a probability p that linearly decays from 1 to 0. This provides dimension warm-up, operator warm-up [4], 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 stabilize supernet training. Our choice of hyperparameters is robust over different NASRec search spaces and recommender system benchmarks. We train the supernet for only one epoch with an Adagrad optimizer, an initial learning rate of 0.12, and a cosine learning rate schedule [24] on target recommender system benchmarks.

Finally, we present the details of regularized evolution and model selection strategies over NAS-Rec 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 the supernet. Specifically, we maintain a population of 128 architectures and run regularized evolution for 240 iterations. In each iteration, we first pick up the best architecture from 64 sampled architectures from the population as the parent architecture and generate 8 child architectures to update the population.

— **Model Selection.** We follow the evaluation protocols in AutoCTR [33] and split each target dataset into 3 sets: training (80%), validation (10%), and testing (10%). During the weight-sharing neural architecture search, we train the supernet on the training set and select the top 15 subnets on the validation set. We train the top 15 models from scratch and select the best subnet, NASRecNet, as the final architecture. We perform light tuning on the learning rate of the best subnet within range (0.1, 0.2) and demonstrate the best learning rate setting on the open-source repository.[4]

5.2 Recommender System Benchmark Results

We train our AutoML-crafted models from scratch on three classic recommender system benchmarks and compare the performance of models that NASRec crafts on three general recommender

---

[4]https://github.com/facebookresearch/NasRec

Table 4. Performance of NASRec on General CTR Prediction Tasks

	Method	Criteo		Avazu		KDD Cup 2012		Search Cost
		Log Loss	AUC	Log Loss	AUC	Log Loss	AUC	(GPU days)
Handcrafted Arts	DLRM [26]	0.4436	0.8085	0.3814	0.7766	0.1523	0.8004	–
	xDeepFM [21]	0.4418	0.8052	–	–	–	–	–
	AutoInt+ [34]	0.4427	0.8090	0.3813	0.7772	0.1523	0.8002	–
	DeepFM [16]	0.4432	0.8086	0.3816	0.7767	0.1529	0.7974	–
NAS-crafted Arts	DNAS [20]	0.4442	–	–	–	–	–	–
	PROFIT [13]	0.4427	0.8095	**0.3735**	0.7883	–	–	∼0.5
	AutoCTR [33]	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
	AutoML @ *NASRec-Small*	**0.4399**	**0.8118**	0.3747	0.7887	**0.1495**	**0.8135**	∼0.25
	AutoML @ *NASRec-Full*	**0.4408**	**0.8107**	**0.3737**	**0.7903**	**0.1491**	**0.8154**	∼0.3

system benchmarks. In Table 4, we report the evaluation results of our end-to-end crafted models and a random search baseline, which randomly samples and trains models in our NASRec search space.

State-of-the-art Performance. Even within an aggressively large *NASRec-Full* search space, our crafted models achieve record-breaking performance over handcrafted CTR models [16, 21, 26] with minimal human priors, as shown in Table 4. Compared with AutoInt [34], the handcrafted model that fabricates feature interactions with delicate engineering efforts, our crafted model achieves ∼0.003 Log Loss reduction on Criteo, ∼0.007 Log Loss reduction on Avazu, and ∼0.003 Log Loss reduction on KDD Cup 2012, with minimal human expertise and interventions.

Next, we compare our crafted models to the more recent NAS-crafted models. Compared with AutoCTR [33], NASRecNet achieves the state-of-the-art 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), our crafted model yields 0.001 Log Loss reduction on Criteo, 0.005 Log Loss reduction on Avazu, and 0.003 Log Loss reduction on KDD Cup 2012. Compared with DNAS [20] and PROFIT [13], which only focuses on configuring part of the architectures, such as dense connectivity, our crafted model achieves at least ∼0.002 Log Loss reduction on Criteo, justifying the significance of full architecture search on recommender systems.

By extending NASRec to an extremely large *NASRec-Full* search space, our crafted model further improves its result on Avazu and outperforms 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 with existing NAS methods [13, 20, 33].

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 with state-of-the-art efficient NAS methods [13, 33], with significant Log Loss improvement on all three benchmarks. This is greatly attributed to the efficiency of WS-NAS on heterogeneous operators and multimodality data.

We observe that a compact *NASRec-Small* search space produces strong random search baselines, whereas a larger *NASRec-Full* search space has a weaker baseline. A limited search budget makes it more challenging to discover promising models within a large search space. Yet, the scalable WS-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, WS-NAS improves the quality of discovered models on Criteo and discovers a better model on Avazu and KDD Cup 2012 than the *NASRec-Small* 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
AutoML @ *NASRec-Small*	**0.4399**	0.3747	0.1495	2.20	3.08	3.48
AutoML @ *NASRec-Full*	0.4408	**0.3737**	**0.1491**	**1.45**	**1.87**	**1.09**

Co-design Evaluation. Following the aforementioned search procedures on NASRec search space, we further enable quantization design space and inherit the same configurations, including dense/sparse building operator choices, hyperparameters, and regularized evolution configurations. Instead of searching on *NASRec-Full* search space, we use *NASRec-Small* search space as all of the included building operators are PIM compatible. We model buffers using CACTI [2] at 32 nm. We use the same ReRAM parameters as modeled in MNSIM 2.0 [49] to obtain the area, latency, and power consumption parameters of the ReRAM crossbars. We build an in-house simulator to simulate the performance. The co-exploration process and performance simulation are performed in Intel Xeon Gold 6254 processors. We use NVIDIA A5000 devices to speed up the co-exploration. We perform quantitative simulation using Criteo dataset features, i.e., 13 integer-dense features and 26 categorical-sparse features. Our simulation results demonstrate a 1.5× speedup for the empirically handcrafted ReRAM design and a 1.1× speedup for RecNMP [18] under the NASRec-Small search space. Additionally, the searched design shows 1.8× and 5.2× energy efficiency compared with the empirical design and RecNMP. We will use the investigation and discovery of the *NASRec-Full* search space in future work.

5.3 Discussion

In this section, we analyze the complexity of our crafted models and demonstrate the impact of our proposed techniques for mitigating ranking disorders and improving the quality of searched models.

Model Complexity Analysis. We compare the complexity of our crafted models with that of state-of-the-art hand-crafted and NAS models. We collect all baselines from AutoCTR [33] and compare performance versus the number of FLOPs in Table 5.

We profile all FLOPs of our crafted models using FvCore [29]. Even without any FLOPs constraints, our crafted models outperform existing models efficiently. Despite achieving lower Log Loss, our crafted models reduce FLOPs by 8.5*times*, 3.8*times*, and 2.8 *times* on Criteo, Avazu, and KDD Cup 2012 benchmarks, respectively. One possible reason is using operator-balancing interaction modules, which project the sparse inputs to a smaller dimension before carrying out cross-term feature interaction. This leads to significantly lower computation costs, contributing to compact yet high-performing recommender models.

Effects of Path Sampling and Fine-tuning. We discuss the path sampling and fine-tuning techniques in Section 4.3 and demonstrate the empirical evaluation of these techniques on the quality of searched models in Table 6. The results show that (1) the importance of path sampling far outweighs the importance of fine-tuning in deciding the quality of searched models, and (2) a higher Kendall's τ that correctly ranks subnets in NASRec search space (i.e., Table 6) indicates a consistent improvement on searched models.

Table 6. Effects of Different Training Techniques on NASRecNet, Evaluated on Criteo

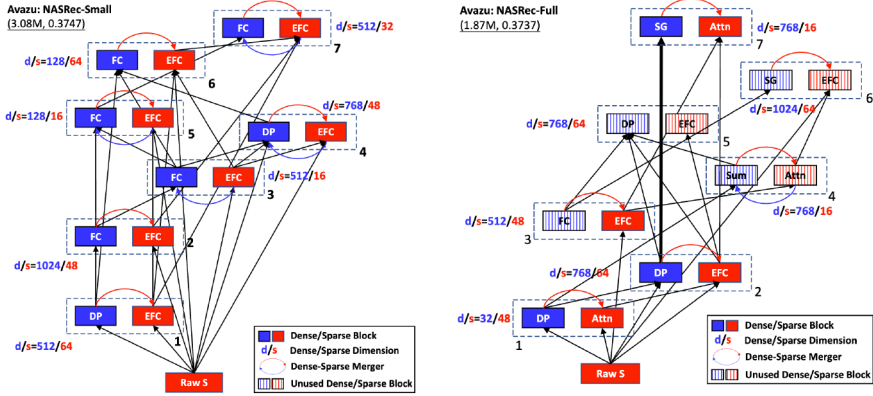| Method	Log Loss	FLOPs (M)
Baseline (single-operator any-connection + fine-tuning)	0.4408	1.45
Single-operator single-connection + fine-tuning	0.4417	1.78
Any-operator any-connection + fine-tuning	0.4413	2.04
Single-operator any-connection, NO fine-tuning	0.4410	3.62

Fig. 5. Best NASRec models discovered on Avazu.

6 Ablation Studies

In this section, we provide more details regarding NASRec, including (1) the visualization and insight of searched architectures, (2) an ad-hoc structured pruning of AutoML-crafted models for enhanced model efficiency on Criteo/Avazu, and (3) the details on subnet sampling and ranking.

6.1 Model Visualization

We visualize the models searched within the *NASRec-Small/NASRec-Full* search space on three CTR benchmarks: Criteo, Avazu, and KDD.

Avazu. Figure 5 depicts the detailed structures of the best architecture within the *NASRec-Small/NASRec-Full* search space. Here, a striped blue (red) block indicates an unused dense (sparse) block in the final architecture, and a bold connection indicates the same source input for a dense operator with two inputs (i.e., Sigmoid Gating and Sum).

As the Avazu benchmark only contains sparse features, the interaction and extraction of dense representations are less important. For example, the best model within *NASRec-Full* search space only contains one operator (i.e., Sigmoid Gating) that solely processes dense representations, yet with more Dot-Product (DP) and Attention (Attn) blocks that interact with the sparse representations. Within the *NASRec-Small* search space, FC layers process dense representations more frequently after interacting with the sparse representations in the Dot-Product block. Yet, processing dense features requires slightly more fully connected blocks than the self-attention mechanism adopted in the *NASRec-Full* search space.

Criteo. Figure 6 depicts the detailed structures of the best architecture within the *NASRec-Small/NASRec-Full* search space. Here, a striped blue (red) block indicates an unused dense (sparse) block in the final architecture, and a bold connection indicates the same source input for a dense operator with two inputs (i.e., Sigmoid Gating and Sum).
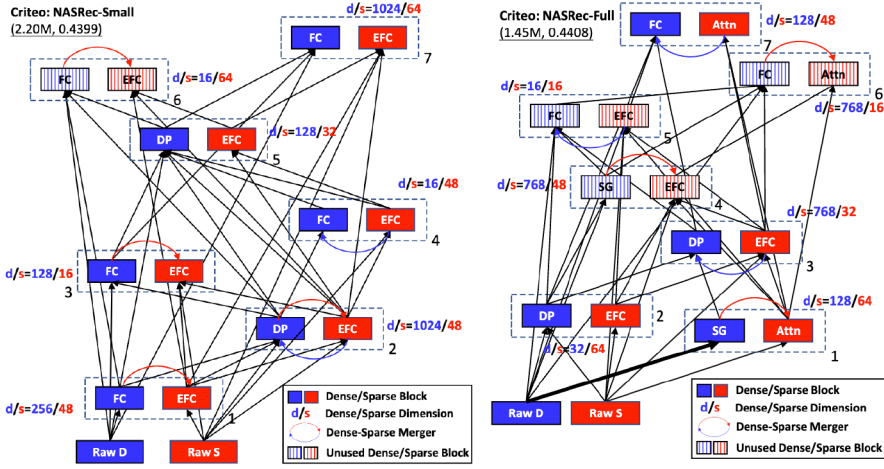
Fig. 6. Best NASRec models discovered on Criteo.

Criteo contains the richest set of dense (sparse) features and, thus, is the most complex in architectural fabrication. We observe that dense connectivity is highly appreciated within both *NASRec-Small* and *NASRec-Full* search space, indicating that feature fusion significantly impacts the log loss on complex benchmarks. In addition, self-gating on raw, dense features (i.e., block one @ *NASRec-Full*) is considered an important motif in interacting features. Similar patterns can also be observed in the best architecture searched on KDD benchmarks.

Owing to the complexity of Criteo and *NASRec-Full* search blocks, the best-searched architecture does not use all seven blocks in the search space. Some of the blocks are not utilized in the final architecture. For example, the best architecture searched within *NASRec-Full* contains only four valid blocks. We leave this as a future work to improve supernet training so that deeper architectures can be discovered in a more scalable fashion.

KDD. Figure 7 depicts the detailed structures of the best architecture within the *NASRec-Small*/*NASRec-Full* search space. Here, a striped blue (red) block indicates an unused dense (sparse) block in the final architecture, and a bold connection indicates the same source input for a dense operator with two inputs (i.e., Sigmoid Gating and Sum). Similar to what we found on Criteo, the searched architecture within *NASRec-Full* has more building operators yet less dense connectivity.

As KDD is a simpler benchmark with fewer dense (sparse) features, the architecture searched is simpler, especially within the NASRec search space. Similar self-gating on dense inputs is still important in designing a better architecture.

We summarize our observations on three unique benchmarks as follows:

— **Benchmark Complexity Decides Architecture Complexity.** The choice of a benchmark decides the complexity of the final architecture. The more complex a benchmark is, the more complicated a searched model is in dense connectivity and operator heterogeneity.
— **Search Space Decides Connectivity.** The best architecture searched within *NASRec-Full* on all three CTR benchmarks contains more operator heterogeneity and less dense connectivity. Yet, the reduced dense connectivity between different choice blocks helps reduce FLOPs consumption of searched models, leading to less complexity and better model efficiency. This also shows that the search for building operators may outweigh the importance of the search for dense connectivity when crafting an efficient CTR model.
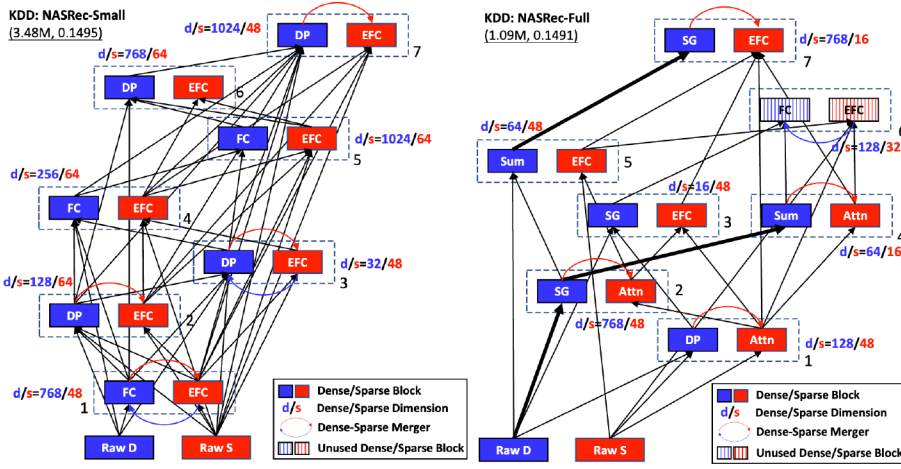
Fig. 7. Best NASRec models discovered on KDD.

— **Attention Has a Huge Impact.** Attention blocks are rarely studied in the existing literature on recommender systems. The architectures searched on *NASRec-Full* search space justify the effectiveness of the attention mechanism on aggregating dense (sparse) features. For example, the first block in the best architecture always adopts an attention layer to interact *raw, sparse inputs*. The stacking of attention blocks is also observed in searched architectures to demonstrate high-order interaction between dense (sparse) features.

— **Self-Gating Is a Useful Motif.** Self-gating indicates a pairwise gating operator with identical dense inputs. On both Criteo/KDD benchmarks, self-gating is discovered to process *raw, dense inputs* and provide higher-quality dense projections. On Avazu, with no dense input features, self-gating is discovered to combine a higher-level dense representation for better prediction results.

6.2 Pruning NASRec via Lottery Ticket

Recommender systems face unique challenges owing to the heterogeneity, uncertainty, and multimodality of data. It is challenging to apply existing pruning techniques [6, 15, 19] and maintain performance on compressed recommender models. For example, existing pruning methods require the training of recommender models for several passes, leading to severe performance degradation and instability [48] due to overfitting. Our methodology is inspired by the Lottery Ticket Hypothesis [12] that learns a smaller sub-architecture (i.e., winning tickets) without involving multi-pass training. Our methodology includes **mask generation** and **structured pruning**.

Mask Generation. We design a mask generation process to mask out zero weights in the original weight matrix W_{orig}. We generate a mask matrix M using a 2-layer **multi-layer perceptron (MLP)** for each weight matrix. The 2-layer MLP inputs the original weight matrix W_{orig}. The first MLP layer employs a ReLU activation function, and the second MLP layer uses a sigmoid activation. The formulation is as follows:

$$M = \sigma(W_2 \cdot \text{ReLU}(W_1 \cdot W_{\text{orig}})), \tag{5}$$

where W_{orig} is the original weight matrix, W_1/W_2 denotes the weights of the first/second layers of the MLP, and $\cdot$ denotes matrix multiplication. The first layer projects the weight matrix to a higher-dimensional space to extract a rich representation. The second layer projects this high

Table 7. Pruning "Model" on General CTR Prediction Tasks

	Model	Mask-Based Pruning		Magnitude-Based Pruning	
		Log Loss ↓	MFLOPs (Percentage)	Log Loss ↓	MFLOPs (Percentage)
Criteo	Model	0.4408	1.45 (100%)	0.4408	1.45 (100%)
	Model+Pruning (T=5)	0.4402	0.78 (54%)	0.4405	0.78 (54%)
	Model+Pruning (T=3)	0.4403	1.01 (69%)	0.4406	1.01(69%)
	Model+Pruning (T=1)	0.4402	1.37 (94%)	0.4406	1.37 (94%)
Avazu	Model	0.3737	1.87 (100%)	0.3737	1.87 (100%)
	Model+Pruning (T=5)	0.3742	0.88 (47%)	0.3748	0.88 (47%)
	Model+Pruning (T=3)	0.3741	1.23 (66%)	0.3746	1.23 (66%)
	Model+Pruning (T=1)	0.3741	1.58 (84%)	0.3744	1.58 (84%)

latent dimension back to the original dimensionality of the weight matrix. The final mask M is obtained through element-wise multiplication with the weight matrix:

$$W_{\text{masked}} = M \odot W_{\text{orig}}. \tag{6}$$

Here, $\odot$ denotes element-wise multiplication.

Structured Pruning. We conduct iterative structured pruning by applying lottery tickets on recommender models to generate masks M and zero out unmasked weights. We initialize the original weight mask as $M^{(0)}$, with all mask values set to 1. The overall iterative structured pruning takes T iterations. Within each iteration t, we train a backbone model with the learned lottery ticket $M^{(t-1)}$ from scratch and zero out 20% of the lowest values in $M^{(t-1)}$ to derive a new mask $M^{(t)}$.

Experimental Evaluation. We apply the pruning methodology structure above to the NAS-Rec model searched within the NASRec search space, which contains various building operators. We apply the pruning methodology on all building blocks covering FC/EFC/DP modules on dense/sparse building operators and dense-to-sparse/sparse-to-dense mergers. We use "Model" to represent AutoML models crafted under *NASRec-Full* search space, with baseline results presented in Table 4. We showcase our evaluation of the Criteo/Avazu dataset in Table 7, demonstrating that our pruning method effectively reduces FLOPs without significant degradation in loss. Our approach reduces 53%/46% FLOPs on NASRec models on the Criteo/Avazu benchmark without incurring noticeable log loss. In some cases, combining lottery tickets with recommender models shows some gains (e.g., "Model" on Criteo), indicating potential model redundancy in existing searched models and possible headroom for improvement.

6.3 Subnet Sampling Details

In Section 4, we sample 100 subnets within *NASRec-Full* search space on the Criteo benchmark, with a more balanced and efficient setting on dimension search components: the dense output dimension can choose from {32, 64, 128, 256, 512} and the sparse output dimension can choose from {16, 32, 64}. All subnets are trained on the Criteo benchmark with a batch size of 1024 and a learning rate of 0.12.

We plot the **Cumulative Distribution Function (CDF)** of sampled subnets on all three benchmarks in Figure 8. For the top 50% architectures evaluated on *NASRec-Full* supernet, we report a Kendall's τ of 0.24 for the Criteo benchmark, showing a clear improvement in ranking top-performing architectures over the random search (0.0). In future work, we propose establishing a CTR benchmark for NAS to increase the statistical significance of evaluated ranking coefficients and better facilitate the research in accurately ranking different architectures.
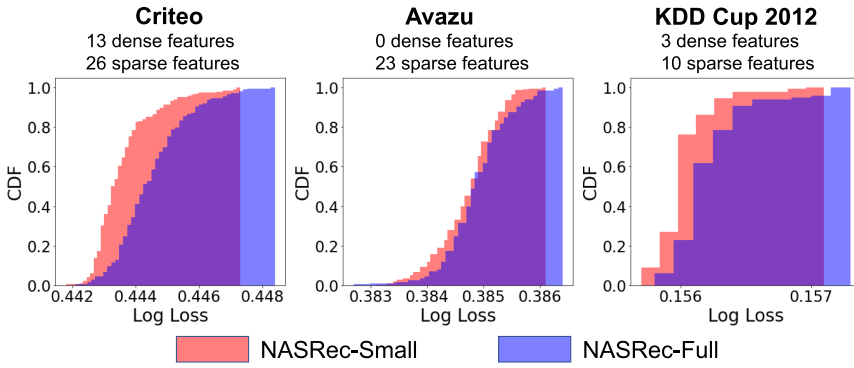
Fig. 8. CDF of Log Loss on CTR benchmarks.

7 Conclusion

In this article, we introduce a novel paradigm for fully enabling AutoML in full-stack recommender model design, leveraging WS-NAS under diverse data modalities and architectures. We construct a large supernet that encompasses the entire architecture search space, incorporating versatile building blocks and dense connection operators to minimize human intervention in automated architecture design for recommender systems. To address the scalability and heterogeneity challenges inherent in large-scale NASRec search spaces, we propose a series of techniques to enhance training efficiency and mitigate ranking disorders. We achieve state-of-the-art performance on three prominent recommender system benchmarks, showcasing promising prospects for a full architecture search and motivating further research towards fully automated architecture fabrication with minimal human priors. Moreover, we suggest opportunities for co-designing models and inference hardware and unlock the potential to perform ad-hoc structure pruning on AutoML-crafted models to achieve improved performance.

Acknowledgment

References

[1] Jimmy Lei Ba, Jamie Ryan Kiros, and Geoffrey E. Hinton. 2016. Layer normalization. *arXiv preprint arXiv:1607.06450* (2016).

[2] Rajeev Balasubramonian, Andrew B. Kahng, Naveen Muralimanohar, Ali Shafiee, and Vaishnav Srinivas. 2017. CACTI 7: New tools for interconnect exploration in innovative off-chip memories. *ACM Trans. Archit. Code Optim.* 14, 2 (2017), 14:1–14:25.

[3] Gabriel Bender, Pieter-Jan Kindermans, Barret Zoph, Vijay Vasudevan, and Quoc Le. 2018. Understanding and simplifying one-shot architecture search. In *International Conference on Machine Learning*. PMLR, 550–559.

[4] 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.

[5] 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).

[6] Ben Carterette and Rosie Jones. 2007. Evaluating search engines by modeling the relationship between relevance and clicks. *Advances in Neural Information Processing Systems* 20 (2007).

[7] 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.

[8] 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.

[9] 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.

[10] Wei Deng, Junwei Pan, Tian Zhou, Deguang Kong, Aaron Flores, and Guang Lin. 2021. DeepLight: Deep lightweight feature interactions for accelerating CTR predictions in ad serving. In *Proceedings of the 14th ACM International Conference on Web Search and Data Mining*. 922–930.

[11] 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).

[12] Jonathan Frankle and Michael Carbin. 2018. The lottery ticket hypothesis: Finding sparse, trainable neural networks. *arXiv preprint arXiv:1803.03635* (2018).

[13] 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).

[14] Luyu Gao, Zhuyun Dai, and Jamie Callan. 2020. Modularized transformer-based ranking framework. *arXiv preprint arXiv:2004.13313* (2020).

[15] Guibing Guo, Jie Zhang, and Neil Yorke-Smith. 2015. TrustSVD: Collaborative filtering with both the explicit and implicit influence of user trust and of item ratings. In *Proceedings of the AAAI Conference on Artificial Intelligence*, Vol. 29.

[16] 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).

[17] 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 8th International Workshop on Data Mining for Online Advertising*. 1–9.

[18] Liu Ke, Udit Gupta, Carole-Jean Wu, Benjamin Youngjae Cho, Mark Hempstead, Brandon Reagen, Xuan Zhang, David Brooks, Vikas Chandra, Utku Diril, Amin Firoozshahian, Kim Hazelwood, Bill Jia, Hsien-Hsin S. Lee, Meng Li, Bert Maher, Dheevatsa Mudigere, Maxim Naumov, Martin Schatz, Mikhail Smelyanskiy, and Xiaodong Wang. 2019. RecNMP: Accelerating Personalized Recommendation with Near-Memory Processing. arXiv:1912.12953 [cs.DC]

[19] Dominik Kowald, Subhash Chandra Pujari, and Elisabeth Lex. 2017. Temporal effects on hashtag reuse in Twitter: A cognitive-inspired hashtag recommendation approach. In *Proceedings of the 26th International Conference on World Wide Web*. 1401–1410.

[20] 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).

[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] Hanwen Liang, Shifeng Zhang, Jiacheng Sun, Xingqiu He, Weiran Huang, Kechen Zhuang, and Zhenguo Li. 2019. DARTS+: Improved differentiable architecture search with early stopping. *arXiv preprint arXiv:1909.06035* (2019).

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

[24] Ilya Loshchilov and Frank Hutter. 2016. SGDR: Stochastic gradient descent with warm restarts. *arXiv preprint arXiv:1608.03983* (2016).

[25] Joe Mellor, Jack Turner, Amos Storkey, and Elliot J. Crowley. 2021. Neural architecture search without training. In *International Conference on Machine Learning*. PMLR, 7588–7598.

[26] 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).

[27] 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.

[28] 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.

[29] Facebook Research. 2022. fvcore. Retrieved from https://github.com/facebookresearch/fvcore

[30] 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.

[31] Ying Shan, T. Ryan Hoens, Jian Jiao, Haijing Wang, Dong Yu, and J. C. 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.

[32] David So, Quoc Le, and Chen Liang. 2019. The evolved transformer. In *International Conference on Machine Learning.* PMLR, 5877–5886.

[33] 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.

[34] 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.

[35] Hanbo Sun, Zhenhua Zhu, Chenyu Wang, Xuefei Ning, Guohao Dai, Huazhong Yang, and Yu Wang. 2023. Gibbon: An efficient co-exploration framework of NN model and processing-in-memory architecture. *IEEE Trans. Comput. Aided Des. Integr. Circuits Syst.* 42, 11 (2023), 4075–4089. https://doi.org/10.1109/TCAD.2023.3262201

[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. *Advances in Neural Information Processing Systems* 30 (2017).

[37] 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).

[38] 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.

[39] 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.

[40] Tianzhe Wang, Kuan Wang, Han Cai, Ji Lin, Zhijian Liu, and Song Han. 2020. APQ: Joint search for network architecture, pruning and quantization policy. In *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition.*

[41] Yitu Wang, Zhenhua Zhu, Fan Chen, Mingyuan Ma, Guohao Dai, Yu Wang, Hai Li, and Yiran Chen. 2021. Rerec: In-ReRAM acceleration with access-aware mapping for personalized recommendation. In *2021 IEEE/ACM International Conference on Computer Aided Design (ICCAD'21).* IEEE, 1–9.

[42] Zhiqiang Wang, Qingyun She, and Junlin Zhang. 2021. MaskNet: Introducing feature-wise multiplication to CTR ranking models by instance-guided mask. *arXiv preprint arXiv:2102.07619* (2021).

[43] 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.

[44] Yachen Yan and Liubo Li. 2023. xDeepInt: A hybrid architecture for modeling the vector-wise and bit-wise feature interactions. *arXiv preprint arXiv:2301.01089* (2023).

[45] Xiaoxuan Yang, et al. 2022. Research progress on memristor: From synapses to computing systems. *IEEE Transactions on Circuits and Systems I: Regular Papers* 69, 5 (2022), 1845–1857.

[46] 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.

[47] Tunhou Zhang, Dehua Cheng, Yuchen He, Zhengxing Chen, Xiaoliang Dai, Liang Xiong, Feng Yan, Hai Li, Yiran Chen, and Wei Wen. 2023. NASRec: Weight sharing neural architecture search for recommender systems. In *Proceedings of the ACM Web Conference 2023.* 1199–1207.

[48] 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.

[49] Zhenhua Zhu, Hanbo Sun, Kaizhong Qiu, Lixue Xia, Gokul Krishnan, Guohao Dai, Dimin Niu, Xiaoming Chen, Xiaobo Sharon Hu, Yu Cao, Yuan Xie, Yu Wang, and Huazhong Yang. 2020. MNSIM 2.0: A behavior-level modeling tool for memristor-based neuromorphic computing systems. In *GLSVLSI '20: Great Lakes Symposium on VLSI 2020, Virtual Event, China, September 7–9, 2020*, Tinoosh Mohsenin, Weisheng Zhao, Yiran Chen, and Onur Mutlu (Eds.). ACM, 83–88.

[50] 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.