DistDNAS: Search Efficient Feature Interactions within 2 Hours

Tunhou Zhang
Duke University
Durham, USA
tunhou.zhang@duke.edu

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

Igor Fedorov
Meta AI
Menlo Park, USA
ifedorov@meta.com

Xi Liu
Meta AI
Menlo Park, USA
xliu1@meta.com

Buyun Zhang
Meta AI
Menlo Park, USA
buyunz@meta.com

Fangqiu Han
Meta AI
Menlo Park, USA
fhan@meta.com

Wen-Yen Chen
Meta AI
Menlo Park, USA
wychen@meta.com

Yiping Han
Meta AI
Menlo Park, USA
yipinghan@meta.com

Feng Yan
University of Houston
Houston, USA
fyan5@central.uh.edu

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

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

Abstract—Search efficiency and serving efficiency are two major axes in building feature interactions and expediting the model development process in recommender systems. Searching for the optimal feature interaction design on large-scale benchmarks requires extensive cost due to the sequential workflow on the large volume of data. In addition, fusing interactions of various sources, orders, and mathematical operations introduces potential conflicts and additional redundancy toward recommender models, leading to sub-optimal trade-offs in performance and serving cost. This paper presents DistDNAS as a neat solution to brew swift and efficient feature interaction design. DistDNAS proposes a supernet incorporating interaction modules of varying orders and types as a search space. To optimize search efficiency, DistDNAS distributes the search and aggregates the choice of optimal interaction modules on varying data dates, achieving a speed-up of over 25× and reducing the search cost from 2 days to 2 hours. To optimize serving efficiency, DistDNAS introduces a differentiable cost-aware loss to penalize the selection of redundant interaction modules, enhancing the efficiency of discovered feature interactions in serving. We extensively evaluate the best models crafted by DistDNAS on a 1TB Criteo Terabyte dataset. Experimental evaluations demonstrate 0.001 AUC improvement and 60% FLOPs saving over current state-of-the-art CTR models.

Index Terms—Recommender Systems, Neural Architecture Search, AutoML, Click-Through Rate Prediction

I. INTRODUCTION

Recommender models vary in depth, width, interaction types and selection of dense/sparse features. These versatile feature interactions exhibit different levels of performance in recommender systems. The design of interaction between dense/sparse features is the key driver to optimizing the recommender models. The advancement of feature interactions incorporates improved prior knowledge into the recommender systems, enhancing the underlying user-item relationships and benefits personalization benchmarks such as Click-Through Rate (CTR) prediction. In past practices, there have been several advancements in feature interactions, such as collaborative filtering [1]–[4]. With the rise of Deep Learning (DL), factorization machines [5], [6], DotProduct [7], [8], deep crossing [9], and self-attention [10]. The stack (e.g., DHEN [11])
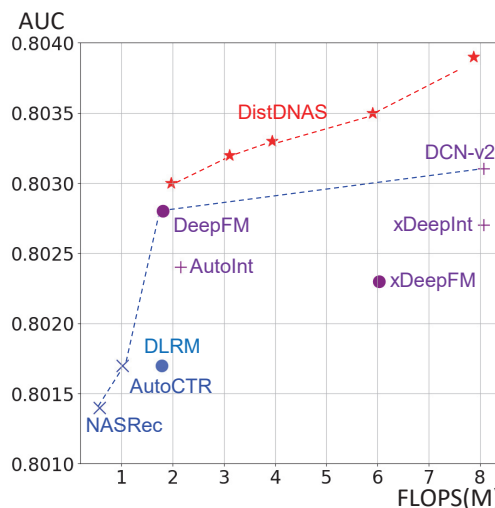

Fig. 1. Model AUC versus FLOPs on Criteo Terabyte. DistDNAS unlocks 0.001 AUC compared to state-of-the-art recommender models.

and the combination (e.g., Mixture of Experts [12], [13]) create versatile interaction types with varying orders, types, and dense/sparse input sources.

The recent thriving of Automated Machine Learning (AutoML) democratized the design of feature interactions and exceeded human performance in various domains, such as feature selection [14] and Neural Architecture Search [15]–[17]. Remarkably, NASRec [16] employs a supernet to represent the search space for recommender models and achieves state-of-the-art results on small-scale CTR benchmarks. However, optimizing feature interactions through manual design/search for large-scale CTR prediction has two challenges. First, designing/searching for the optimal feature interactions requires extensive wall-clock time, as the design/search sequentially iterates high data volume in production to obtain a good solution in feature interaction. This raises obstacles to ensuring the freshness of feature interaction on the latest data, thus potentially harming production quality and causing model staleness. Thus, an efficient search methodology is needed to explore the selection of the feature interaction and scale

with the data volume growth. Second, fusing versatile feature interactions introduces potential conflicts and redundancy in recommender models. For example, we observe that loss divergence issues can be directly attributed to a complex interaction module (e.g., xDeepInt) with alternative feature interaction modules (e.g., CrossNet, DLRM) when training on a large dataset. This creates a sub-optimality of performance-efficiency trade-offs in service. As the relationship between model size and model performance in recommender systems has not been exploited yet, it is uncertain whether fusing multiple orders and types of feature interactions into a single architecture can benefit recommender models or whether it is possible to harvest the improvement from different feature interactions fully.

In this paper, we address the above challenges and present an efficient AutoML system, **Dist**ributed **D**ifferentiable **N**eural **A**rchitecture **S**earch (DistDNAS), to craft efficient feature interactions in a few hours. DistDNAS follows the setting of the supernet-based approach in NASRec [16] and applies Differentiable Neural Architecture Search [18] (DNAS) to learn the structure of a feature interaction. In DistDNAS, a feature interaction is built with multiple choice blocks. Each choice block represents a linear combination of feature interaction modules (e.g., Linear, CrossNet [9], etc.). DistDNAS presents two robust techniques to improve design and model efficiency during feature interaction search. For search efficiency, DistDNAS distributes DNAS on each training day and averages the learned weighting to derive the best combinatorial choice of interaction modules. Without ad-hoc optimizations such as embedding table sharding and communication between multiple devices, DistDNAS exhibits better scalability with a large volume of training data, achieving $\sim 25\times$ speedup on 1TB Criteo Terabyte benchmark and reducing the search cost from 2 days to 2 hours. For serving efficiency, DistDNAS calculates the cost importance of each interaction module and incorporates a differentiable cost-aware regularization loss to penalize cost-expensive interaction modules. As cost-aware regularization loss removes unnecessary interaction modules within the supernet, DistDNAS alleviates potential conflict and harvests performance improvement.

We evaluate DistDNAS on the 1TB Criteo Terabyte dataset using AUC, LogLoss, and Normalized Entropy (NE) [19] as evaluation metrics. DistDNAS removes redundant interaction modules without human intervention and discovers efficient feature interactions, unlocking 0.001 higher AUC and/or 60% fewer FLOPs in the discovered models. The optimization in AUC and FLOPs brings state-of-the-art models, see Figure 1. We summarize our contributions as follows.

- We analyze the search and serving efficiency challenges when designing feature interactions on large-scale CTR recommender benchmarks.
- We propose DistDNAS, an AutoML system, to tackle the efficiency challenge in feature interaction design. DistD-NAS distributes the search over multiple data splits and averages the learned architecture on each data split for search efficiency. In addition, DistDNAS incorporates a

cost-aware regularization into the search to enhance the serving efficiency of discovered feature interactions.
- Our empirical evaluations demonstrate that DistDNAS significantly pushes the Pareto frontier of state-of-the-art CTR models.

II. Related Work

Feature Interactions in Recommender Systems. The feature interactions within recommender systems such as CTR prediction have been thoroughly investigated in various approaches, such as Logistic Regression [20], and Gradient-Boosting Decision Trees [19]. Recent approaches apply deep learning based interaction [21] to enhance end-to-end modeling experience by innovating Wide & Deep Neural Networks [7], Deep Crossing [22], Factorization Machines [5], [6], DotProduct [8] and gating mechanism [9], [22], ensemble of feature interactions [11], feature-wise multiplications [23], and sparsifications [24]. In addition, these works do not fully consider the impact of fusing different types of feature interactions, such as the potential redundancy, conflict, and performance enhancement induced by a variety of feature interactions. DistDNAS constructs a supernet to explore different orders and types of interaction modules and distributes differentiable search to advance search efficiency.

Cost-aware Neural Architecture Search. Neural Architecture Search (NAS) automates the design of Deep Neural Network (DNN) in various applications: the popularity of NAS is consistently growing in brewing Computer Vision [18], [25]–[27], Natural Language Processing [28], [29], and Recommender Systems [15]–[17], [30]. Tremendous efforts are made to advance the performance of discovered architectures to brew a state-of-the-art model. Despite the improvement in search/evaluation algorithms, existing NAS algorithms overlook the opportunity to harvest performance improvements by addressing potential conflict and redundancy in feature interaction modules. DistDNAS regularizes the cost of the searched feature interactions and prunes unnecessary interaction modules as building blocks, yielding better FLOPs-LogLoss trade-offs on CTR benchmarks.

III. Differentiable Feature Interaction Space

Supernet is a natural fusion that incorporates feature interaction modules. DistDNAS emphasizes search on feature interaction modules and simplifies the search space from NAS-Rec [16]. A supernet in DistDNAS contains multiple choice blocks, with a fixed connection between them and a fully enabled dense-to-sparse/sparse-to-dense merger within all choice blocks. Unlike NASRec [16], which solely selects the optimal interaction module within each choice block, DistDNAS can choose an arbitrary number of interaction modules within each choice block and use differentiable bi-level optimization [18] to determine the best selection. This allows flexibility in fusing varying feature interactions and obtaining the best combination with enhanced search efficiency. We present the details of feature interaction modules as follows.
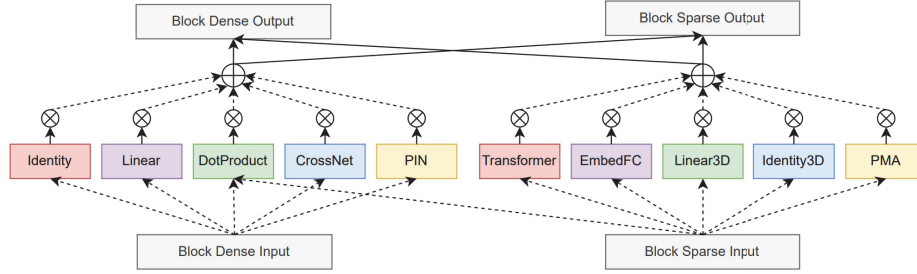
Fig. 2. Feature interaction search space for each choice block in DistDNAS. Here, a dashed line denotes a searchable feature interaction in DistDNAS, and $\otimes$ denotes the mixing of different feature interaction modules.

A. Feature Interaction Modules

Feature interaction modules connect dense 2D inputs with sparse 3D inputs to learn valuable representations on user modeling. In a recommender system, a dense input is a 2D tensor generated from either raw dense features or dense interaction modules, such as a fully connected layer. A sparse input is a 3D tensor of sparse embeddings generated either by raw sparse/categorical features or by sparse interaction modules, such as a self-attention layer. We define a dense input as $X_d \in \mathbb{R}^{B \times dim_d}$ and a sparse input $X_s \in \mathbb{R}^{B \times N_s \times dim_s}$. Here, B denotes the batch size, dim_d/dim_s denotes the dimension of the dense/sparse input, and N_s denotes the number of inputs in the sparse input.

We collect a set of simple feature interaction modules from the existing literature, as demonstrated in Figure 2. A dense interaction module produces a dense output given input features, and a sparse interaction module produces a sparse output given input features. These interaction modules can cover a set of state-of-the-art CTR models, such as DLRM [8], DeepFM [22], xDeepInt [31], DCN-v2 [9], and AutoInt [10].

- **Identity/Identity3D** is a dense/sparse interaction module that carries an identity transformation on dense/sparse input.
- **Linear/Linear3D** is a dense/sparse interaction module that applies on 2D/3D dense inputs.
- **DotProduct** [7], [8] is a dense interaction module that computes pairwise inner products of dense and sparse inputs.
- **CrossNet** [9]/**PIN** [31] is a dense interaction module with gate inputs from various sources.
- **Transformer** [32] is a sparse interaction module that utilizes the multihead attention mechanism to learn the weighting of different sparse inputs. The queries, keys, and values of a Transformer layer are identical.
- **Embedded Fully-Connected (EmbedFC)** is a sparse interaction module that applies a linear operation along the sparse (middle) dimension.
- **Pooling by Multihead Attention (PMA)** [33] is a sparse interaction module that forms attention between seed vectors and sparse features.

If the dimensions of inputs do not match, a proper linear projection (e.g., linear) will be applied within all dense/sparse interaction modules. The feature interaction search space contains versatile dense/sparse interaction modules.

B. Differentiable Supernet

In DistDNAS, a differentiable supernet contains N choice blocks with a rich collection of feature interactions. Each choice block employs a set of dense/sparse interactions $op^{(i)} = \{op_d^{(i)}, op_s^{(i)}\}$ to take dense/sparse inputs $X_d^{(i)}/X_s^{(i)}$ and learn useful representations. Each choice block contains $|op_d| = 5$ dense feature interactions and $|op_s| = 5$ sparse feature interactions. Each choice block receives input from previous choice blocks and produces a dense output $Y_d^{(i)}$ and a sparse output $Y_s^{(i)}$. In block-wise feature aggregation, each choice block concatenates the dense/sparse output from the previous 2/1 choice blocks as dense/sparse block input. For dense inputs in previous blocks, the concatenation occurs in the last feature dimension. For sparse inputs in previous blocks, concatenation occurs in the middle dimension to aggregate different sparse inputs. We present the mixing of different feature interaction modules in the following context.

Continuous Relaxation of Feature Interactions. Within a single choice block in the differentiable supernet, we depict the mixing of candidate feature interaction modules in Figure 2. We parameterize the weighting of each dense/sparse/dense-sparse interaction using architecture weights. In choice block i, we use $\alpha^{(i)} = \{\alpha_1^{(i)}, \alpha_2^{(i)}, ..., \alpha_{|op_d|}^{(i)}\}$ to represent the weighting of a dense interaction module and parameterize the selection of dense interaction modules with architecture weight $A = \{\alpha^{(1)}, \alpha^{(2)}, ..., \alpha^{(N)}\}$. Similarly, we parameterize the selection of sparse interaction modules with architecture weight $B = \{\beta^{(1)}, \beta^{(2)}, ..., \beta^{(N)}\}$. We employ the Gumbel Softmax [34] trick to allow a smoother sampling from categorical distribution on dense/sparse inputs as follows:

$$Y_d^{(i)} = \sum_{j=1}^{|op_d|} \frac{\exp(\frac{\log \alpha_j^{(i)} + g_j}{\lambda})}{\sum_k^{|op_d|} \exp(\frac{\log \alpha_k^{(i)} + g_k}{\lambda})} op_j(X_d^i, X_d^{i-1}), \quad (1)$$

$$Y_s^{(i)} = \sum_{j=1}^{|op_s|} \frac{\exp(\frac{\log \beta_j^{(i)} + g_j}{\lambda})}{\sum_k^{|op_s|} \exp(\frac{\log \beta_k^{(i)} + g_k}{\lambda})} op_j(X_s^i, X_s^{i-1}). \quad (2)$$

Here, g_j and g_k are sampled from the Gumbel distribution, and λ is the temperature. As a result, a candidate architecture $\mathcal{C}$ can be represented as a tuple of dense/sparse feature interaction: $\mathcal{C} = (A, B)$. Within DistDNAS, our goal is to perform a differentiable search and obtain the optimal architecture $\mathcal{C}^*$ that contains the weighting of dense/sparse interaction modules.
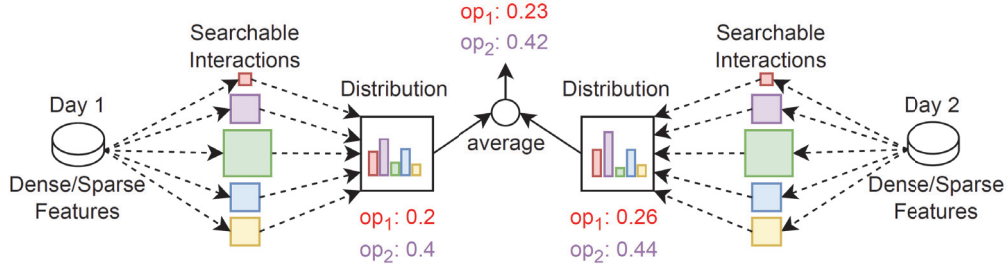
Fig. 3. Overview of DistDNAS methodology. Here, dashed lines denote searchable interaction modules, and the size of interaction modules indicates the cost penalty applied to each interaction module for serving efficiency.

Discretization. Discretization converts the weighting in the optimal architecture to standalone models to serve CTR applications. Past DNAS practices [18] typically select the top k modules for each choice block, brewing a building cell containing a fixed number of modules in all parts of the network. In recommender models, we discretize each choice block by a fixed threshold θ to determine useful interaction modules. For example, in the choice block i, we discretize the weight $\alpha^{(i)}$ to obtain the discretized dense interaction module $\hat{\alpha}^{(i)}$ as follows:

$$\hat{\alpha}_j^{(i)} = \begin{cases} 1, if \hat{\alpha}_j^{(i)} \geq \theta, \\ 0, otherwise. \end{cases}$$

We typically set threshold $\theta = 1/|op|$ (i.e., 0.2 for dense/sparse module search) for each choice block, or use a slightly larger value (e.g., 0.25) to remove more redundancy. There are a few advantages of adopting threshold-based discretization in recommender models. First, using a threshold θ is a clearer criterion to distinguish important/unimportant interaction modules within each choice block. Second, since a recommender model contains multiple choice blocks with different hierarchies, levels, and dense/sparse input sources, there is a need for varying numbers of dense/sparse interactions to maximize the representation capacity within each module.

IV. Towards Efficiency in DNAS

Search efficiency and serving efficiency are two major considerations in deploying DNAS algorithms in large-scale CTR datasets. In this section, we first revisit DNAS and address the efficiency bottleneck via a distributed search mechanism. Then, we propose our solution to reduce the service cost of feature interaction via a cost-aware regularization approach. Figure 3 provides the core methodology of DistDNAS.

A. Revisiting DNAS on Recommender Systems

A Click-Through Rate (CTR) prediction task usually contains multiple days of training data. In recommender systems, we typically use a few days of data (i.e., day 1 to day T) as the training source and evaluate the trained model based on its CTR prediction over subsequent days. DNAS carries bilevel optimization to find the optimal candidate architecture $C^* = (A^*, B^*)$ as follows:

$$(A^*, B^*) = \arg\min_{A,B} LogLoss_{x \sim D}[x; w^*(A, B), A, B]. \quad (3)$$

Here, BCE denotes binary cross-entropy, $D = (D_1, ..., D_T)$ indicates the training data from day 1 to day T, w indicates the weight parameters within the DNN architecture, and t indicates a certain day of data. The previous DNAS workflow must iterate over T days of data, with a significant search cost. More specifically, the large search cost originates from the following considerations in search efficiency and scalability:

- Sequentially iterating over T days of data requires T times the search cost of DNAS on a single day of data. This creates challenges for model freshness in production environments where T can be extremely large.
- Deploying the search over multiple devices may suffer from poor scalability due to communication. For example, the forward/backward process must shift from model to data parallelism when offloading tensors from embedding table shards to feature interaction modules.
- Within our implementation on Criteo Terabyte, the throughput on multiple NVIDIA A5000 GPUs is lower than the throughput on a single GPU during the search, as demonstrated in the Queries-Per-Second (QPS) analysis in Figure 4. Thus, realizing good scalability in the growth of computing devices in DNAS is challenging.

The above considerations point to a distributed version of DNAS, where we partition the training data, launch a DNAS procedure on each day of training data, and average the results to derive the final architecture. We hereby propose a DistDNAS search with the following bilevel optimization.

$$(A^*, B^*)^{(t)} = \arg\min_{A,B} LogLoss_{x \sim D_t}[x; w^*(A, B), A, B], \quad (4)$$

such that

$$w^*(A, B)^{(t)} = \arg\min_{w} LogLoss_{x \sim D_t}[x; w(A, B), A, B]. \quad (5)$$

Here, $t \in \{1, 2, ..., T\}$ indicates a certain day of training data. DistDNAS aggregates the learned weights on each day to retrieve the final architecture coefficients with a simple averaging aggregator as follows:

$$(A^*, B^*)^{(*)} = \sum_{t=1}^{T} \frac{1}{T} (A^*, B^*)^{(t)} \quad (6)$$

The simple averaging scheme incorporates the statistics from each day of data to obtain the learned architecture weights. In addition, DistDNAS can be asynchronously paralleled on different computing devices, accelerating search scalability
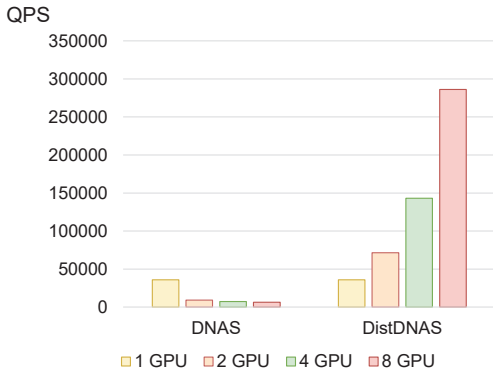
QPS

Fig. 4. QPS comparison between DistDNAS and DNAS.

and reducing the total wall-clock run-time in recommender systems. Figure 4 compares DistDNAS versus DNAS on 1-8 NVIDIA A5000 GPUs, with 4K batch size. Due to communication savings with 1-GPU training, DistDNAS benefits from significantly lower search costs.

B. Cost-aware Regularization

Serving the cost of feature interactions, e.g., FLoating-point OPerations (FLOPs), is critical in recommender systems. A lower servicing cost indicates a shorter response time to process a user query request. As a result, optimizing the cost of recommender models is as important as optimizing their performance in the production environment.

In DNAS, we measure the cost by combining training FLOPs and inference latency. The cost of a feature interaction in discovery is dependent on the weights of the learned architecture $C^* = (A^*, B^*)$. An intuition to optimize the feature interaction module is rewarding cost-efficient operators (e.g., Linear, Identity) while penalizing cost-inefficient operators (e.g., Transformer, CrossNet) during differentiable search. Motivated by this, we introduce a differentiable cost regularizer to penalize large models in discovery. The cost regularizer adds an additional regularization term R to the loss function during DNAS to induce cost-effective feature interactions in discovery.

We use j to represent an index of a feature interaction module in $\mathcal{C}$, for example, the index of a dense interaction module. We first sample a few pairs of architecture and cost
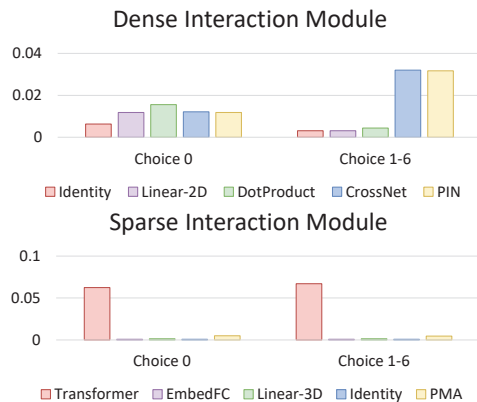

Fig. 5. Normalized cost importance in a 7-block supernet.

metrics from the DistDNAS search space and create a cost mapping $cost : \mathrm{C} \rightarrow \mathrm{R}$ to model the relationship between feature interactions and FLOPs. Then, we use the permutation importance [35] to obtain the importance of offline cost s_j in the cost mapping $cost$, illustrating the offline FLOPs importance of an interaction module i. Finally, we formulate a cost-aware loss and incorporate it to regularize all interaction modules: $R(\mathrm{A}, \mathrm{B}) = \gamma \sum_{op^j \in (\mathrm{A}, \mathrm{B})} s_j$. Here, γ is an adjustable coefficient to control the strength of cost-aware regularization. With cost-aware regularization, the final architecture of DNAS with cost-aware loss searched on a single day t can be formulated as follows.

$$(\mathrm{A}^*, \mathrm{B}^*)^{(t)} = \arg\min_{\mathrm{A}, \mathrm{B}} LogLoss_{x \sim D_t}[x; w^*(\mathrm{A}, \mathrm{B}), \mathrm{A}, \mathrm{B}] + R(\mathrm{A}, \mathrm{B}),$$
(7)

such that

$$w^*(\mathrm{A}, \mathrm{B})^{(t)} = \arg\min_{w} LogLoss_{x \sim D_t}[x; w(\mathrm{A}, \mathrm{B}), \mathrm{A}, \mathrm{B}] + R(\mathrm{A}, \mathrm{B}).$$
(8)

As the feature interaction search space adopts a fixed connectivity and dimension configuration during the search, the cost importance of different interaction modules is unique in the first choice block, and identical across all other choice blocks. We demonstrate the normalized cost importance of each interaction module in Figure 5. Among all interaction modules, the DotProduct contributes to a significant amount of FLOPs consumption by integrating dense and/or sparse features. Except for Transformer, sparse interaction modules contribute significantly fewer serving costs compared to their dense counterparts. Thus, despite the strong empirical performance of Transformer models, recommender models choose Transformer sparingly to build an efficient feature interaction.

V. EXPERIMENTS

We thoroughly examine DistDNAS on Criteo Terabyte. First, we introduce the experiment settings of DistDNAS, which produce efficient feature interactions. Then, we compare the performance of models crafted by DistDNAS versus a series of metrics with strong hand-crafted/AutoML baselines.

A. Experiment Setup

We illustrate the key components of our experiment setup and elaborate on the detailed configuration.

Training Dataset. Criteo Terabyte contains 4B training data on 24 different days. Each data item contains 13 integer features and 26 categorical features. Each day of data on Criteo Terabyte contains ~ 0.2B data. During the DistDNAS search, we use data from day 1 to day 22 to learn architecture the optimal architecture weights: $C^* = (A^*, B^*)$. During the evaluation, we use the data from day 1 to day 23 as a training dataset and use **day 24** as a holdout testing dataset. We perform inter-day data shuffling during training yet iterate over data from day 1 to day 23 sequentially.

Data Preprocessing. We do not apply special preprocessing to dense features except for normalization. We cap the maximum embedding table size to 5M for sparse embedding tables and use an embedding dimension of 16 to obtain each sparse

TABLE I
PERFORMANCE OF THE BEST DISCOVERED DISTDNAS MODEL ON 1TB CRITEO TERABYTE.

Model	FLOPS(M)	Params(M)	NE (%) ↓	Relative NE (%) ↓	AUC ↑	LogLoss ↓
DLRM [8]	1.79	453.60	0.8462	0.0	0.8017	0.12432
DeepFM [5]	1.81	453.64	0.845	-0.14	0.8028	0.12413
xDeepFM [6]	6.03	454.14	0.846	-0.02	0.8023	0.12429
AutoInt [10]	2.16	453.80	0.8455	-0.08	0.8024	0.12421
DCN-v2 [9]	8.08	459.91	0.845	-0.14	0.8031	0.12413
xDeepInt [31]	8.08	459.91	0.8455	-0.08	0.8027	0.12421
NASRec-tiny [16]	0.57	452.47	0.8463	0.01	0.8014	0.12437
AutoCTR-tiny [15]	1.02	452.78	0.8460	-0.02	0.8017	0.12429
DistDNAS	1.97	453.62	**0.8448**	**-0.17**	0.8030	**0.12410**
	3.11*	454.70	**0.8444**	**-0.21**	**0.8032**	**0.12405**
DistDNAS (M=2)	3.94	455.48	**0.8448**	**-0.17**	**0.8033**	**0.12410**
DistDNAS (M=3)	5.90	457.31	**0.8440**	**-0.26**	**0.8035**	**0.12399**
DistDNAS (M=4)	7.87	459.14	**0.8438**	**-0.29**	**0.8039**	**0.12395**

feature. Thus, each model contains ∼450M parameters in the embedding table.

Optimization. We train all models from scratch without inheriting knowledge from other sources, such as pre-trained models or knowledge distillation. For sparse parameters, we utilize Adagrad with a learning rate of 0.04. For dense parameters, we use Adam with a learning rate of 0.001. No weight decay is performed. During training, we use a fixed batch size of 8192 and a fixed learning rate after the initial warm-up.

Architecture Search. Our supernet contains $N = 7$ choice blocks during the search. We choose γ=0.004 for cost-aware regularization. During the search, we linearly warm up the learning rate from 0 to maximum with 10K warm-up steps and use a batch size of 8K to learn the architecture weights while optimizing the DNAS supernet. During discretization, we use θ=0.20 as the discretization threshold for DistDNAS marked with *, and use $\theta = 0.25$ to other feature interactions created by DistDNAS, in Table I, As most baseline models are larger, we naively stack M copies of feature interactions in parallel to match the FLOPs of large models, such as DCN-v2 [9] and xDeepInt [31].

Training. To ensure a fair comparison and better demonstrate the strength of the discovered models, we employ no hyper-parameter tuning for all models. We linearly warm up the learning rate from 0 to maximum using the first 2 days of training data. We use single-pass training to prevent overfitting and iterate the whole training dataset only once.

Baselines. We select the popular hand-crafted design choice of CTR models from the existing literature to serve as baselines, i.e., DLRM [8], DeepFM [5], xDeepFM [6], AutoInt [10], DCN-v2 [9] and xDeepInt [31]. We also incorporate the best models from the NAS literature: AutoCTR [15] and NASRec [16] to serve as baselines and use the best model discovered for Criteo Kaggle. Without further specification, all hand-crafted or AutoML baselines use $dim_s = 16$ as the embedding dimension. All hand-crafted or AutoML baselines use 512 or $dim_d = 256$ units in the MLP layer, including 1 MLP layer in dense feature processing and 7 MLP layer in aggregating high-level dense/sparse features. All AutoML models use $N_s = 16$ for sparse interaction modules. This ensures a fair comparison between hand-crafted and AutoML models, as the widest part in hand-crafted/AutoML models

does not exceed 512. All hand-crafted feature interactions (e.g., CrossNet) are stacked 7 times to match $N = 7$ blocks in the AutoML supernet, as NASRec, AutoCTR, and proposed DistDNAS employ $N = 7$ blocks for feature interaction. We name the derived NAS baselines **NASRec (tiny)** and **AutoCTR (tiny)**. We implement all baseline feature interactions based on open-source code and/or paper demonstration.

B. Evaluation on Criteo Terabyte

We use DistDNAS to represent the performance of the best models discovered by DistDNAS and compare performance against a series of cost metrics such as FLOPs and parameters. We use AUC, Normalized Entropy (NE) [19], and LogLoss as evaluation metrics to measure model performance. We also calculate the testing NE of each model relative to DLRM and demonstrate relative performance. Note that relative NE is equivalent to relative LogLoss on the same testing day of data. Table I summarizes our evaluation of DistDNAS. Here, M indicates the number of parallel stackings we apply on DistDNAS to match the FLOPs of baseline models.

Upon transferring to large datasets, previous AutoML models [15], [16] searched on smaller datasets are less competitive when applied to large-scale Criteo Terabyte. This is due to sub-optimal architecture transferability from the source dataset (i.e., Criteo Kaggle) to the target dataset (i.e., Criteo Terabyte). Among all baseline models, DCN-v2 achieves state-of-the-art performance on Criteo Terabyte with the lowest LogLoss/NE and highest AUC. DistDNAS shows remarkable model efficiency by establishing a new Pareto frontier on AUC/NE versus FLOPs. With a discretization threshold of 0.25, DistDNAS outperforms tiny baseline models such as DLRM and DeepFM and unlocks at least 0.02% AUC/NE with on-par FLOPs complexity. With a discretization threshold of 0.2, DistDNAS achieves better AUC/NE as state-of-the-art DCN-v2 models, yet with a reduction of over 60% FLOPS. By naively stacking more blocks in parallel, DistDNAS outperforms DCN-v2 by 0.001 AUC and achieves state-of-the-art.

VI. DISCUSSION

In this section, we conduct ablation studies and analyze various confounding factors within DistDNAS.
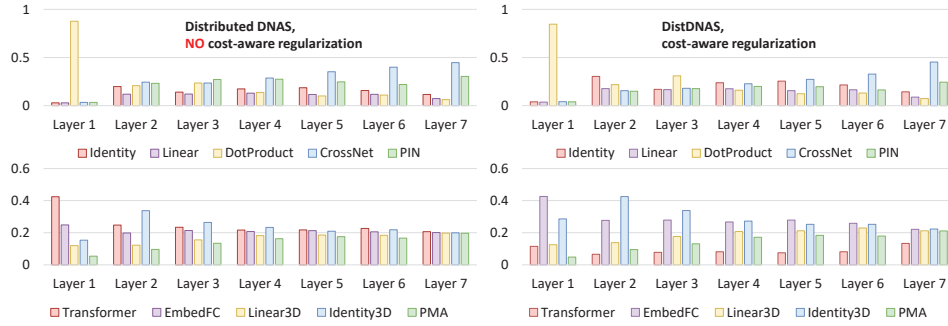
Fig. 6. Comparison of learned architecture weights under distributed DNAS versus DistDNAS.

A. DistDNAS Search Strategy

We discuss the alternative choices to DistDNAS as follows. **SuperNet** indicates a direct use of the DistDNAS supernet as a feature interaction module. No search is performed.

Distributed DNAS applies the DistDNAS search process in a distributed manner but does not involve cost-aware regularization. Both distributed DNAS and DistDNAS take 2 hours to complete on NVIDIA A5000 GPUs.

One-shot DNAS kicks off DNAS and iteratively over the entire search data set (that is, 22 days in Criteo Terabyte) to obtain the best architecture. Running a one-shot DNAS takes ~ 50 GPU hours on an NVIDIA A5000 GPU.

Fresh DNAS only uses the most recent data in the training data set (that is, day 22 on Criteo Terabyte) and performs a search to learn the best architecture. This serves as a strong baseline due to the correlation between testing data set and the most recent data.

DistDNAS applies all the techniques proposed in this paper, including distributed search and cost-aware regularization. Figure 6 compares the learned architecture weights in Distributed DNAS versus DistDNAS. DistDNAS is more likely to preserve cost-efficient interaction modules like EmbedFC/Identity than distributed DNAS without cost-aware regularization.

We perform each of the searches above and evaluate different search strategies based on the following questions:

- **(Search Convergence)** Whether the search converges and produces effective feature interactions?
- **(Training Convergence)** Does the discovered feature interaction converge on a large-scale Criteo Terabyte benchmark with 24 days of training data?
- **(Testing Quality)** What is the quality of the interactions of the features discovered?

Table II summarizes a study of different search strategies for these questions. We have a few findings regarding the use of distributed search and cost-aware regularization.

TABLE II
STUDY OF DIFFERENT DISTDNAS SEARCH STRATEGIES.

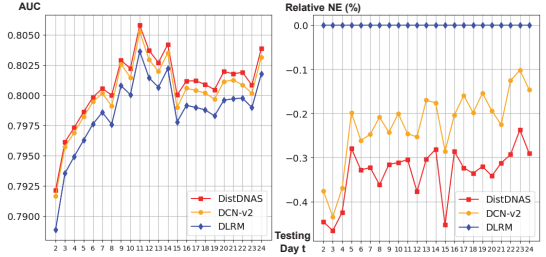| Strategy	Searching Converge?	Training Converge?	Testing FLOPs/NE
SuperNet	N/A	No	N/A
One-shot DNAS	No	No	N/A
Freshness DNAS	Yes	No	N/A
Distributed DNAS	Yes	Yes	3.56M/0.8460
DistDNAS	Yes	Yes	3.11M/0.8444

Fig. 7. Comparison of AUC and Relative NE under recurring training.

A standalone supernet cannot converge when trained on Criteo Terabyte dataset, indicating that different feature interactions may have conflicts with each other.

On a large-scale dataset such as Criteo Terabyte, distributing DNAS over multiple-day splits and aggregating the learned weight architectures are critical to the convergence of search and training. This is because in recommender systems, there might be an abrupt change in different user behaviors across/intra-days; thus, a standalone architecture learned on a single day may not be suitable to capture the knowledge and fit all user-item representations. Additionally, as NAS may overfit the target dataset, a standalone feature interaction searched on day X may not be able to learn day Y well and is likely to collapse due to changes in user behavior.

We also compare distributed DNAS with DistDNAS to demonstrate the importance of cost-aware regularization. Experimental evaluation demonstrates that FLOPS-regularization enhances the performance of searched feature interaction, removing the redundancy contained in the supernet. This observation provides another potential direction for recommender models to compress unnecessary characteristics and derive better recommender models, such as the usage of pruning.

B. Performance Analysis under Recurring Training Scenario

Recurring training [19] is a common practice in recommender system applications. In recurring training, practitioners must regularly update the model on the latest data to gain fresh knowledge. Here, we simulate the scenario in recurring training to evaluate top-performing feature interactions (i.e., DistDNAS and DCN-v2) on different training/evaluation splits of Criteo Terabyte. We use day 1 to day t as training dates and day $t+1$ as testing dates to report AUC and relative NE.

We demonstrate the evaluation of recurring training on DistDNAS, DCN-v2, and DLRM (baseline) in Figure 7. DistDNAS consistently outperforms the previous state-of-the-

art DCN-v2 on all testing splits under recurring training. This indicates that DistDNAS successfully injects the implicit patterns contained within the large-scale dataset into the searched feature interaction.

VII. Conclusion

In this article, we emphasize search efficiency and serving efficiency in the design of feature interactions through a differentiable supernet. We propose DistDNAS to explore the differentiable supernet containing various dense and sparse interaction modules. We distribute the search on different days of training data to advance search scalability, reducing end-to-end search cost from 2 days to 2 hours with a $25\times$ speed-up in scalability. In addition, DistDNAS incorporates cost-aware regularization to remove potential conflicts and redundancies within feature interaction modules, yielding better performance and efficiency in searched architectures.

References

[1] O. Barkan and N. Koenigstein, "Item2vec: neural item embedding for collaborative filtering," in *2016 IEEE 26th International Workshop on Machine Learning for Signal Processing (MLSP)*, pp. 1–6, IEEE, 2016.

[2] D. Wang, P. Cui, and W. Zhu, "Structural deep network embedding," in *Proceedings of the 22nd ACM SIGKDD international conference on Knowledge discovery and data mining*, pp. 1225–1234, 2016.

[3] X. He, L. Liao, H. Zhang, L. Nie, X. Hu, and T.-S. Chua, "Neural collaborative filtering," in *Proceedings of the 26th international conference on world wide web*, pp. 173–182, 2017.

[4] C. Zhang, L. Yu, Y. Wang, C. Shah, and X. Zhang, "Collaborative user network embedding for social recommender systems," in *17th SIAM International Conference on Data Mining, SDM 2017*, pp. 381–389, Society for Industrial and Applied Mathematics Publications, 2017.

[5] H. Guo, R. Tang, Y. Ye, Z. Li, and X. He, "Deepfm: a factorization-machine based neural network for ctr prediction," *arXiv preprint arXiv:1703.04247*, 2017.

[6] J. Lian, X. Zhou, F. Zhang, Z. Chen, X. Xie, and G. Sun, "xdeepfm: Combining explicit and implicit feature interactions for recommender systems," in *Proceedings of the 24th ACM SIGKDD international conference on knowledge discovery & data mining*, pp. 1754–1763, 2018.

[7] H.-T. Cheng, L. Koc, J. Harmsen, T. Shaked, T. Chandra, H. Aradhye, G. Anderson, G. Corrado, W. Chai, M. Ispir, *et al.*, "Wide & deep learning for recommender systems," in *Proceedings of the 1st workshop on deep learning for recommender systems*, pp. 7–10, 2016.

[8] M. Naumov, D. Mudigere, H.-J. M. Shi, J. Huang, N. Sundaraman, J. Park, X. Wang, U. Gupta, C.-J. Wu, A. G. Azzolini, *et al.*, "Deep learning recommendation model for personalization and recommendation systems," *arXiv preprint arXiv:1906.00091*, 2019.

[9] R. Wang, R. Shivanna, D. Cheng, S. Jain, D. Lin, L. Hong, and E. Chi, "Dcn v2: Improved deep & cross network and practical lessons for web-scale learning to rank systems," in *Proceedings of the web conference 2021*, pp. 1785–1797, 2021.

[10] W. Song, C. Shi, Z. Xiao, Z. Duan, Y. Xu, M. Zhang, and J. Tang, "Autoint: Automatic feature interaction learning via self-attentive neural networks," in *Proceedings of the 28th ACM International Conference on Information and Knowledge Management*, pp. 1161–1170, 2019.

[11] B. Zhang, L. Luo, X. Liu, J. Li, Z. Chen, W. Zhang, X. Wei, Y. Hao, M. Tsang, W. Wang, *et al.*, "Dhen: A deep and hierarchical ensemble network for large-scale click-through rate prediction," *arXiv preprint arXiv:2203.11014*, 2022.

[12] S. Masoudnia and R. Ebrahimpour, "Mixture of experts: a literature survey," *Artificial Intelligence Review*, vol. 42, pp. 275–293, 2014.

[13] K. Balog, L. Azzopardi, and M. De Rijke, "Formal models for expert finding in enterprise corpora," in *Proceedings of the 29th annual international ACM SIGIR conference on Research and development in information retrieval*, pp. 43–50, 2006.

[14] B. Liu, C. Zhu, G. Li, W. Zhang, J. Lai, R. Tang, X. He, Z. Li, and Y. Yu, "Autofis: Automatic feature interaction selection in factorization models for click-through rate prediction," in *Proceedings of the 26th ACM SIGKDD International Conference on Knowledge Discovery & Data Mining*, pp. 2636–2645, 2020.

[15] Q. Song, D. Cheng, H. Zhou, J. Yang, Y. Tian, and X. Hu, "Towards automated neural interaction discovery for click-through rate prediction," in *Proceedings of the 26th ACM SIGKDD International Conference on Knowledge Discovery & Data Mining*, pp. 945–955, 2020.

[16] T. Zhang, D. Cheng, Y. He, Z. Chen, X. Dai, L. Xiong, F. Yan, H. Li, Y. Chen, and W. Wen, "Nasrec: Weight sharing neural architecture search for recommender systems," *arXiv preprint arXiv:2207.07187*, 2022.

[17] R. Krishna, A. Kalaiah, B. Wu, M. Naumov, D. Mudigere, M. Smelyanskiy, and K. Keutzer, "Differentiable nas framework and application to ads ctr prediction," *arXiv preprint arXiv:2110.14812*, 2021.

[18] H. Liu, K. Simonyan, and Y. Yang, "Darts: Differentiable architecture search," *arXiv preprint arXiv:1806.09055*, 2018.

[19] X. He, J. Pan, O. Jin, T. Xu, B. Liu, T. Xu, Y. Shi, A. Atallah, R. Herbrich, S. Bowers, *et al.*, "Practical lessons from predicting clicks on ads at facebook," in *Proceedings of the eighth international workshop on data mining for online advertising*, pp. 1–9, 2014.

[20] M. Richardson, E. Dominowska, and R. Ragno, "Predicting clicks: estimating the click-through rate for new ads," in *Proceedings of the 16th international conference on World Wide Web*, pp. 521–530, 2007.

[21] S. Zhang, L. Yao, A. Sun, and Y. Tay, "Deep learning based recommender system: A survey and new perspectives," *ACM computing surveys (CSUR)*, vol. 52, no. 1, pp. 1–38, 2019.

[22] R. Wang, B. Fu, G. Fu, and M. Wang, "Deep & cross network for ad click predictions," in *Proceedings of the ADKDD'17*, pp. 1–7, 2017.

[23] Z. Wang, Q. She, and J. Zhang, "Masknet: Introducing feature-wise multiplication to ctr ranking models by instance-guided mask," *arXiv preprint arXiv:2102.07619*, 2021.

[24] W. Deng, J. Pan, T. Zhou, D. Kong, A. Flores, and G. Lin, "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*, pp. 922–930, 2021.

[25] B. Zoph, V. Vasudevan, J. Shlens, and Q. V. Le, "Learning transferable architectures for scalable image recognition," in *Proceedings of the IEEE conference on computer vision and pattern recognition*, pp. 8697–8710, 2018.

[26] W. Wen, H. Liu, Y. Chen, H. Li, G. Bender, and P.-J. Kindermans, "Neural predictor for neural architecture search," in *Computer Vision–ECCV 2020: 16th European Conference, Glasgow, UK, August 23–28, 2020, Proceedings, Part XXIX*, pp. 660–676, Springer, 2020.

[27] H. Cai, C. Gan, T. Wang, Z. Zhang, and S. Han, "Once-for-all: Train one network and specialize it for efficient deployment," *arXiv preprint arXiv:1908.09791*, 2019.

[28] D. So, Q. Le, and C. Liang, "The evolved transformer," in *International conference on machine learning*, pp. 5877–5886, PMLR, 2019.

[29] H. Wang, Z. Wu, Z. Liu, H. Cai, L. Zhu, C. Gan, and S. Han, "Hat: Hardware-aware transformers for efficient natural language processing," *arXiv preprint arXiv:2005.14187*, 2020.

[30] C. Gao, Y. Li, Q. Yao, D. Jin, and Y. Li, "Progressive feature interaction search for deep sparse network," *Advances in Neural Information Processing Systems*, vol. 34, pp. 392–403, 2021.

[31] Y. Yan and L. Li, "xdeepint: a hybrid architecture for modeling the vector-wise and bit-wise feature interactions," *arXiv preprint arXiv:2301.01089*, 2023.

[32] A. Vaswani, N. Shazeer, N. Parmar, J. Uszkoreit, L. Jones, A. N. Gomez, Ł. Kaiser, and I. Polosukhin, "Attention is all you need," *Advances in neural information processing systems*, vol. 30, 2017.

[33] J. Lee, Y. Lee, J. Kim, A. Kosiorek, S. Choi, and Y. W. Teh, "Set transformer: A framework for attention-based permutation-invariant neural networks," in *International conference on machine learning*, pp. 3744–3753, PMLR, 2019.

[34] E. Jang, S. Gu, and B. Poole, "Categorical reparameterization with gumbel-softmax," *arXiv preprint arXiv:1611.01144*, 2016.

[35] L. Breiman, "Random forests," *Machine learning*, vol. 45, pp. 5–32, 2001.