Practice on Pruning CTR Models for Real-world Systems

Rihan Chen*
rihan.crh@alibaba-inc.com
Alibaba Group

Yuchao Zheng*
yuchao.zyc@bupt.edu.cn
Beijing University of Posts and Telecommunications

Guorui Zhou
guorui.xgr@alibaba-inc.com
Alibaba Group

Xinchen Luo
lxc263790@alibaba-inc.com
Alibaba Group

Jingwei Zhuo
zjw169463@alibaba-inc.com
Alibaba Group

Xianjie Qiao
xianjie.qxj@alibaba-inc.com
Alibaba Group

Yunlong Xu
yunlong.xyl@alibaba-inc.com
Alibaba Group

Xiaoqiang Zhu
xiaoqiang.zxq@alibaba-inc.com
Alibaba Group

ABSTRACT

Deep models have been pervasive for click-through rate (CTR) prediction in modern recommender systems. However, their stellar performance is at the cost of tremendous resource consumption, which has received little attention so far. The core of this problem is to trade off model complexity and accuracy. Motivated by the successes of model pruning in computer vision, we propose an Auto Pruning-based Architecture Search (APAS) pipeline, which leverages reinforcement learning to learn the optimization policy without hand-crafted heuristics. Our method takes a first step towards filling the void in recommender system, which can directly and efficiently optimize arbitrary performance metrics, e.g. throughput and latency, in real-world systems.

Experimental results over two large-scale real-world datasets show that our proposed method significantly outperforms traditional methods for different CTR model structures. The proposed method has been widely deployed in Taobao Display Advertising system. In production environments, APAS brings at least 10% throughput improvement on various model structures in several scenarios without significant negative effect for prediction accuracy. We hope our experiences in developing such an automatic pruning pipeline will be helpful for people interested in applying pruning techniques to industrial systems.

CCS CONCEPTS

• **Information systems** → **Recommender systems**; • **Computing methodologies** → **Reinforcement learning**.

KEYWORDS

Recommender System, Model Pruning, AutoML, Arbitrary Metrics

*Both authors contributed equally to this research.

DLP-KDD 2021, August 15, 2021, Singapore

ACM ISBN 978-x-xxxx-xxxx-x/YY/MM…$15.00
https://doi.org/10.1145/nnnnnnn.nnnnnnn

1 INTRODUCTION

Large-scale recommender systems and online advertising have been pervasive and essential to build bridges between users and content providers. In recent years, deep learning has gained considerable interests in the field of personalized recommendation and proven to be effective in internet enterprises, e.g, YouTube [3] and Amazon [16].

However, the ever-growing model complexity has gradually hindered online deployment in recommender system because of the limit of computational resource. Under such a circumstance, trading off model complexity and its prediction accuracy becomes a crucial problem. Hence, our main goal is to design an end-to-end AutoML pipeline for real-world recommender systems. And the AutoML pipeline could automatically produce highly efficient model structure without sacrificing its accuracy. Model pruning, a crucial technique for model acceleration and compression, arouse our attention greatly owing to its convincing performance in computer vision.

In order to handle extremely large scale of users and items under computational and storage constraints, industrial recommender systems usually follow a multi-stage cascade architecture. There are two typical parts that need to be highlighted, i.e., candidate generation and ranking. The former is aimed at retrieving thousands of items from an extremely large candidate set [3, 13, 29–31], while the latter is aimed at sorting the candidate set based on more accurate predicted scores, e.g., click-through-rate (CTR) [27, 28]. Usually, the models at these two stages have more complicated model structures than others, which give great challenges for online inference engine. Hence, the trade-off between complexity and accuracy is more crucial for these stages, especially for ranking stage. In this paper, we mainly focus on CTR models for ranking, though our methods can be extended to candidate generation models as well.

Besides, performance metrics, such as throughput and latency, should be optimized directly in real-world recommender systems. In computer vision, models typically take Convolutional Neural Network (CNN) as their backbone architectures, which are computation intensive and graphical processing unit (GPU) friendly. In these fields, it is usually enough to use FLOPs as an strong indicator for both model compression and acceleration. However, the foundation of network architecture in recommendation systems

differs from what is mainly used in computer vision. For example, researchers usually resort to recurrent neural network for modelling behaviour sequence and attention mechanism for relevance between user and target item. In these cases, FLOPs is not necessarily equivalent with latency or throughput, thus invalid for model acceleration in usual. As illustrated in Figure 1, the relationship between latency and FLOPs for these selected model configurations is not significant, especially for the interval from 1,000,000 FLOPs to 3,000,000 FLOPs. Motivated by these observations, it is necessary to optimize arbitrary metrics directly in real-world systems.

Moreover, we hope that the AutoML pruning pipeline doesn't increase either the complexity or the risk for the whole system. For example, online learning and batch learning have dominated in modern recommender systems for modeling the constantly changing data distribution. Consequently, real-world systems are always in the states of releasing and deploying. Thus, pruning methods based on weight inheritance cannot be easily deployed in real-world system because they inevitably extend the online deployment process. In contrast, other methods, e.g. Neural Architecture Search (NAS), could be decoupled from the online deployment process. NAS-like methods only require that the model structures can be updated regularly, because they doesn't focus on model weights but structures by nature[19].

To address the challenges above, we propose Auto Pruning-based Architecture Search (APAS), which tailored for models in real-world recommender systems. APAS can automatically trade off model speed and accuracy without introducing complexity and risk for online deployment. Model pruning has been studied extensively in computer vision and achieved great success for model deployment on mobile devices. However, researchers seldom set foot in the context of real-world recommender systems because of the highly complicated and dynamic production environment. Currently, it is challenging for online inference engine to satisfy the requirements of throughput and latency set by the system because of the explosion of model complexity. Hence, the importance of model compression and acceleration cannot be overstated in modern recommender systems. The main contributions of our paper are summarized as follows:

- We propose APAS pipeline, a industrial level solution to model pruning. Arbitrary performance metrics (e.g., latency and throughput) could be automatically optimized by APAS. Moreover, the decoupling with model deployment process makes APAS non-intrusive and lightweight. To the best of our knowledge, it is the first attempt to systematically solve the challenges of model pruning in real-world recommender systems.
- We also propose APAS-light that can solve the pruning task with much fewer iterations. The light version aims to further address the scalability issue in industry, which greatly reduce the number of trial-and-error explorations for model structures.
- Through analysis on the by-product of APAS pipeline, we reveal that APAS does not only achieve the balance between computational resources and accuracy, but also provides a useful tool for analyzing the performance of CTR models.

We conduct extensive experiments on two large-scale real-world datasets, which show that our APAS pipeline outperforms existing methods significantly for different CTR model structures. Online A/B tests in Taobao Display Advertising platform also demonstrate the effectiveness of APAS in production environment.

2 RELATED WORKS

Deep CTR Model. In recent years, deep learning has been widely acknowledged in many areas, such as computer vision, natural language processing. Inspired by these successes, influence of deep learning is also pervasive in the fields of information retrieval, recommender systems and online advertising. Especially, deep learning demonstrates its effectiveness when applied to click-through rate (CTR) prediction [3, 13, 22, 29–31]. Compared with traditional methods, deep CTR model can not only learn feature representations [1] from scratch but also model the user's ever-changing interests through its sophisticated model structure. These modules, which are not GPU-friendly in usual, give great challenges to model pruning.

Unstructured and Structured Pruning. Many works have been done on model acceleration through network pruning, especially in computer vision [7, 9, 11, 14, 18, 21, 24–26]. One major branch of network pruning methods is unstructured pruning. Specifically, unstructured pruning removes individual weight connections from a network. Unstructured pruning has a long history, which has been quite active for years in the setting of statistics for variable selection [23]. More recently, Han et al. [7] propose to prune redundant connections based on weight magnitude, which is further integrated into their Deep compression pipeline [6]. In most recent works, the lottery tickets theory [4] gains notable attention, which claimed that model weights and sparsity are coupled with each other for achieving competitive model with high sparsity. However, even though unstructured pruning can introduce astonishing sparsity, it cannot lead to model inference speedup without dedicated hardware/libraries. In contrast, structured pruning is more practical by achieving model compression and acceleration at the same time. Hu et al [10] propose to use channel pruning based on their predefined metrics that evaluate the importance of each neuron after activation. Thinet [20], on the other hand, greedily searches the effective sub-network within the whole network by reconstruction errors.

Neural Architecture Search. Neural Architecture Search (NAS) [32] has been widely adopted in the real world. It doesn't focus on weight inheritance but model structure exploration. Liu et al [19] have shown that model structure plays a central role in determining the model prediction under some mild assumptions, rather than delicate weight initialization. These observations prove the effectiveness of NAS to a certain extent, especially in computer vision. Hence, as what we stated earlier, NAS-like methods have the advantage of integration with recommender system. In general, NAS can be divided into two categories - reinforcement learning-based method and differentiable method. For reinforcement learning-based method, researchers intend to reduce the search space for model structures by reinforcement learning. AMC [8] uses reinforcement learning to generate compression ratio for each layer. It speeds up the exploration by using the validation accuracy without

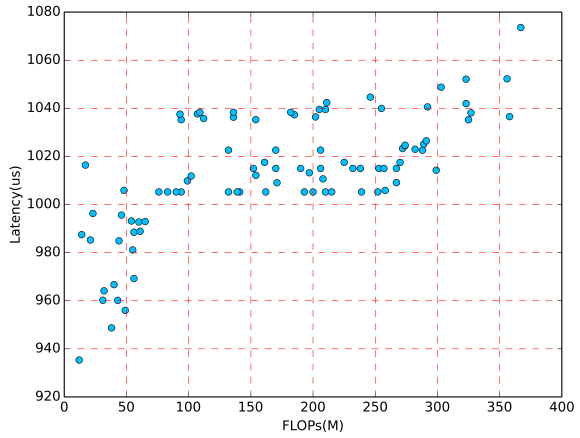

Figure 1: Illustration of relationship between FLOPs and latency. In deep CTR model, these metrics are usually not interchangeable. We take Deep Interests Network(DIN) as an example. We select some different model configurations and display the models with FLOPs in ascending order.

model fine-tuning as a delegate reward. Even though AMC can accelerate the architecture exploration, its delegate rewards inevitably introduce bias to the agent. Moreover, it is also questionable that AMC prune the model based on weight magnitude. To address the scalability issue, many works attempt to formulate the task in a differentiable manner. DARTS [17] searches the space of pre-defined operations through gradient descent by continuous relaxation of the architecture representation. DMCP [5] models the channel pruning as a Markov process and samples the best candidate by the Markov process with learned transition probabilities. These methods usually use a differentiable surrogates for budget regularization. However, we cannot simply assume that budget regularization is continuous in real-world systems, let alone differentiable. In this paper, we borrow ideas from one-shot pruning, searching for optimal subnet structures within supernet. Compared with previous works, we devise an pruning-based architecture search approach for real-world recommender systems which is easy-to-deploy, automatic and scalable.

3 METHODOLOGY

The APAS mainly follows one-shot pruning paradigm and applies structured pruning for both model compression and acceleration. NAS-like method has been adopted by APAS by considering the online system's complexity. Arbitrary metrics, e.g, latency and throughput, can be incorporated into our APAS pipeline and optimized directly based on reinforcement learning. Besides, APAS uses a layerwise penalization strategy because of each layer's different contribution to these metrics and prediction. Moreover, we propose APAS-light that remains almost the same performance as APAS but could greatly decrease the number of trial-and-error model structure searches.

3.1 Preliminaries

Deep CTR Model. In real-world systems, the data for CTR model are usually formulated as a set of tuples $\{(u, v, y)\}$, where u and v stand for user $u \in U$ and item $v \in V$ respectively. y is usually binary which indicates whether u interacts with v, e.g. click in CTR prediction.

$$y = \begin{cases} 1, & \text{if } u \text{ interacts with } v \\ 0, & \text{o.w.} \end{cases} \tag{1}$$

In CTR prediction, the CTR model aims to predict the likelihood that user u clicks on v with a learned function $f_\Theta(\cdot)$ parameterized by Θ. Usually, there are three main parts of raw features $(\mathbf{u}, \mathbf{v}, \mathbf{c})$. Here $\mathbf{v}$ denotes the feature vector for the target item and $\mathbf{c}$ is the context feature from prediction requests. $\mathbf{u}$ is user side feature which in usual contains some side information and a sequence of items with which user $\mathbf{u}$ interacts in history. For deep CTR model, $f_\Theta(\cdot)$ is based on deep neural network which learn feature representations of raw features $(\mathbf{u}, \mathbf{v}, \mathbf{c})$ from scratch and extract the structural and temporal information within features through its architecture.

Reinforcement learning. Reinforcement learning intends to solve sequential decision-making problems. At each timestep t, an agent observes an observation x_t, take action a_t and receives a reward r_t. An agent's behaviors are defined by a policy π, which is a mapping from states to a probability distribution over actions $\pi : S \rightarrow P(A)$. In reinforcement learning, we usually model an problem as a Markov decision process (MDP) with a state space S, action space A, transition dynamics $p(s_{t+1}|s_t, a_t)$, and a reward $r(s_t, a_t)$. The final goal of reinforcement learning is to take actions in an environment in order to maximize the cumulative reward $R_t = \sum_{i=1}^{T} \gamma^{(i-t)} r(s_i, a_i)$ where $\gamma \in [0, 1]$ is a discount factor that determines how much the reinforcement learning agents concern rewards in the distant future relative to those in the immediate future.

3.2 Problem Definition

In general, we work on building a AutoML pipeline which could automatically trade off model speedup and accuracy. Mathematically, we formulate the pruning problem as a constrained optimization problem, where the objective is to maximize the model prediction performance under the computation budget constraint.

$$\begin{aligned} &\arg\max_{\zeta} SCORE_\zeta(x_{test}, y_{test}) \\ &\text{s.t. } m_\zeta \leq C \end{aligned} \tag{2}$$

where $SCORE_\zeta(x_{test}, y_{test})$ is any prediction score on test data sets e.g. AUC or GAUC. m_ζ is any performance metric m for the model with structure ζ and C is the corresponding constraint required by real-world systems. Without loss of generality, we assume that we maximize $SCORE_\zeta(x_{test}, y_{test})$ with m_ζ less than or equal to C. ζ is the solution for the constrained optimization problem, which denotes model structure in our specific problem. In our scenario, we endeavor to search for the best candidate (subnet) within unpruned model (supernet) by following the one-shot pruning paradigm. It is worth mentioning that arbitrary metrics are optimized directly through our formulation, which is quite important for model pruning in real-world systems.

3.3 Overall Pipeline

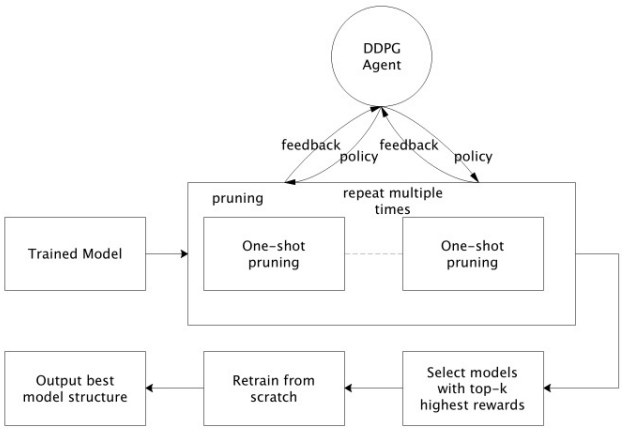

Figure 2: Illustration of APAS pipeline. In APAS pipeline, one-shot pruning repeat several times in order to interact with DDPG agent. DDPG agent learns optimization policy automatically during the interactions. Finally, APAS selects best model structure from the model candidates with top-k highest rewards

As shown in the Figure 2, APAS mainly follow one-shot pruning paradigm which takes a trained model, prune it once and extract an subnet from an supernet one. As the overall pipeline of APAS described in Algorithm 1, the pipeline takes a trained model $M_{\zeta_{full}}$ as an input, which is model M with model structure ζ_{full}. Then the trained model $M_{\zeta_{full}}$ enters the pruning stage, where DDPG agent interacts with this module several times to learn the optimization policy and automatically control the pruning process. We make it clean that the pruning stage (line 6 in Algorithm 1) is a pluggable module. In Section 3.6, Algorithm 1 will be substituted by APAS-light pruning described in Algorithm 3 for exploring the model structure solution space more efficiently. We describe the one-shot pruning of APAS pipeline in Algorithm 2. The one-shot pruning repeats several times for interactions with DDPG agent. For each one-shot pruning, it will start from a trained model and add scaling factor η (initialized with 1) to each neuron's output. The DDPG agent will automatically generate layerwise soft-thresholding λ's in order to control the model sparsity at coarse-grained neuron level. The penalization will terminate till the performance metric m_{ζ_e} meets the constraint at episode e. Then, the training will be continued for a while without penalizing the scaling factors. By doing that, APAS can mitigate the prediction bias incurred by the penalties added on neurons, which distinguish APAS from AMC. Finally, we evaluate the pruned model with x_{test} and y_{test} and get the prediction score $\hat{SCORE}_{\zeta}(x_{test}, y_{test})$, e.g. AUC. To be noticed, $\hat{SCORE}_{\zeta}(x_{test}, y_{test})$ is a delegate for $SCORE_{\zeta}(x_{test}, y_{test})$ due to the penalization from DDPG agent. After several one-shot pruning processes, APAS will select models with top-k highest rewards to retrain from scratch, and then output the optimal model structure with highest validation score $SCORE_{\zeta}(x_{test}, y_{test})$.

Algorithm 1 APAS overall pipeline

1: **Input:** Trained Model $M_{\zeta_{full}}$
2: **Output:** Best Model structure ζ
3: Initialize replay buffer R
4: Initialize model queue Q
5: **for** e **in 1** ... N:
6: Run Algorithm 2
7: Update DDPG agent
8: **end for**
9: Select models with top-k highest r_e from model queue Q
10: Retrain these models from scratch
11: **Return** model structure ζ with highest $SCORE_{\zeta}(x_{test}, y_{test})$

Algorithm 2 APAS one-shot pruning

1: **Input:** $M_{\zeta_{full}}$, e, R, Q
2: Add scaling factor η_j for the jth neuron n_j
3: Agent generates layerwise soft-thresholding $\lambda_{l,e}$ upwardly.
4: **while** ($m_{\zeta_e} \geq C$):
5: Continue pruning by penalizing scaling factors.
6: **end while**
7: Get model M_{ζ_e} with m_{ζ_e} which satisfies the constraint
8: Fine-tune M_{ζ_e} for several iterations
9: Evaluate M_{ζ_e} and get $\hat{SCORE}_{\zeta}(x_{test}, y_{test})$
10: Get $r_e = \hat{SCORE}_{\zeta_e}(x_{test}, y_{test}) - \beta \times \max(0, \mathrm{m}_{\zeta_e} - \mathrm{C})$
11: Append $(s_{l,e}, a_{l,e}, r_e, s_{l+1,e})$ for each layer to replay buffer R
12: Append model structure ζ_e to model queue Q

3.4 Structure Pruning with Scaling Factor

Unbiased reward is one of the keys to make the DDPG agent learn optimization policy correctly. However, the cost of retraining pruned model with structure ζ from scratch and getting unbiased validation accuracy is almost prohibitive. Therefore, we decide to use pruning-while-training paradigm to get a surrogate score efficiently. Moreover, we desire that the pruning mechanism can be end-to-end and easy-to-use, which can be included into the model training without many modifications. Motivated by these considerations, we resort to scaling factors [11] to put structured pruning into effect. Scaling factor could scale the outputs of certain structures, such as neurons, groups or residual blocks, as illustrated in Figure 3. The structure selection is achieved by the sparse penalties added on these scaling factors. The APAS pipeline penalizes some of the scaling factors into zeros through the sparsity regularizers and prunes unimportant parts of models automatically. Formally, we have:

$$\min_{\mathrm{w},\lambda} \frac{1}{N} \sum_{i=1}^{N} L(y_i, f(x_i, W, \eta)) + R_s(\eta) \tag{3}$$

where $R_s(\eta)$ is the sparsity regularizer for scaling factors, which is usually the lasso penalty. We summarized our main reasons for using scaling factors as follow:

- The scaling factor method is data-driven, easy-to-use and compatible with one-shot pruning.

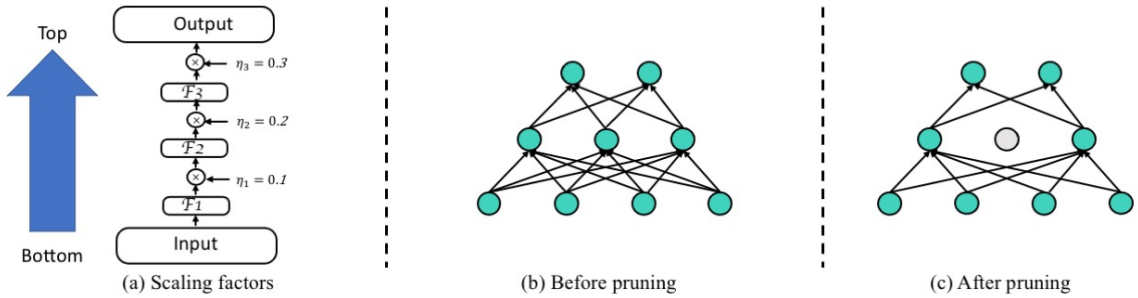

Figure 3: Illustration of scaling factor method. Scaling factor method scale the outputs of some specific structures,e.g.neurons, by introducing scaling factors into the network. The model structures could be pruned at coarse-grained level, once some scaling factors scale their corresponding outputs to zeros. As shown in Figure 3c, the scaling factor method falls into the category of structured pruning. Hence, model compression and acceleration could be accomplished simultaneously by scaling factor method.

- Scaling factor method is end-to-end in one training pass, which allows us to use pruning-while-training paradigm.
- The sparsity of scaling factors can be directly influenced by their corresponding soft-thresholding λ's, which can be controlled by DDPG agent.

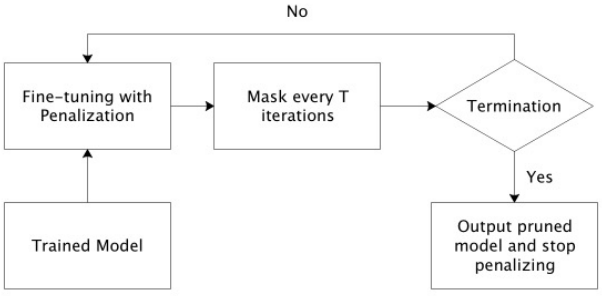

Figure 4: Illustration of iterative pruning in APAS. We implement iterative pruning in each one-shot pruning, where we mask parts of neurons every T iterations.

Additionally, it is also desirable that the process of model pruning can be stable in real-world systems, which means the sparsity of model is controllable and without sudden change or spike. Therefore, APAS also integrates conventional scaling factor method with iterative pruning [2], which refers to model pruning in an iterative way. Once every T iterations, APAS masks parts of model structure which have scaling factors penalized to zeros. The iterative pruning is quite practical in real world due to the following reasons:

- Iterative pruning can stabilize the pruning process, i.e. APAS can guarantee that model sparsity monotonically increase during one-shot pruning by iterative masking.
- As an important component of APAS light, iterative pruning also provides a way to solve the scalability issue of model pruning in industrial systems. In Section 3.6, it will be discussed in detail.
- Some off-the-shelf hardware or library has special requirements for model structures in order to get model acceleration, e.g. the Nvidia Turing Core acceleration requires the weight matrix dimension should be multiple of eight.

3.5 APAS with Reinforcement Learning

We leverage reinforcement learning to automatically control the sparsity of scaling factors. Reinforcement learning empower the APAS pipeline to optimize arbitrary metrics directly, e.g. latency and throughput. In contrast, differentiable methods, such as DARTS and DMCP, usually need delicate model design and differentiable surrogates for budget constraints. In APAS, the sparsity of scaling factors can be directly controlled by the magnitudes of their corresponding soft-thresholding λ's in the pruning-while-training process. Specifically, we adopt deep deterministic policy gradient (DDPG) [15] method to search over continuous action space of soft-thresholding λ's. By controlling over continuous actions, we avoid an explosion of the number of discrete actions which usually existed in conventional reinforcement learning-based Neural architecture search (NAS) method.

The State Space For each layer l, we describe its state with 6 features which can reflect its distinctness and relevance to other layers.

$$(\mathrm{l}, \mathrm{op\_type}[\mathrm{l}], \mathrm{FLOPs}[\mathrm{l}], \mathrm{FLOPs}_{\mathrm{pre}}[\mathrm{l}], \mathrm{FLOPs}_{\mathrm{rest}}[\mathrm{l}], \mathrm{action}[\mathrm{l}-1]) \tag{4}$$

where l is the layer l's index that indicates its position in the whole model structure. op_type[l] represents the type of layer l, e.g. fully connected layer and GRU. FLOPs[l] is the FLOPs of computation involved in layer l. $\mathrm{FLOPs}_{\mathrm{pre}}[\mathrm{l}]$ and $\mathrm{FLOPs}_{\mathrm{rest}}[\mathrm{l}]$ are sums of feed-forward FLOPs before and after layer l. action[l − 1] is the corresponding action for layer $l-1$ from DDPG agent. Notably, we also include features that can reflect the layer l's state in global,e.g. $\mathrm{FLOPs}_{\mathrm{pre}}[\mathrm{l}]$ and $\mathrm{FLOPs}_{\mathrm{rest}}[\mathrm{l}]$. The reason is that each layer's contribution to the performance metrics is mutually influenced.

The Action Space Conventionally, reinforcement learning based NAS methods search over a discrete action space. The pruning task, especially for large and complicated models, usually suffers from

an explosion of the number of discrete actions. Thus, these works usually need lots of trial and error to explore such a large action space.

In this work, we take each layer's soft-thresholding λ as our action in order to solve the scalability issue introduced by discrete action space. The soft-thresholding λ has a continuous action space that can be explored more efficiently with limited data sets.

DDPG Agent For each episode, the agent takes layerwise actions in a bottom-up manner. For example, at layer l, the agent receives the corresponding state s_l and outputs the continuous action a_l which is the soft-thresholding λ. Then, the agent will take the successive action at layer $l+1$ of which state includes the action from layer l. After the agent generates actions for all layers, APAS will add the scaling factors to the unpruned model and start the one-shot pruning process. The pruning process will follow iterative pruning paradigm. Once every T iterations, APAS will evaluate the performance metrics during the masking pause. The process will terminate once the model satisfies the performance metrics. After pruning, APAS pipeline will continue training for several iterations and evaluate the model with validation datasets to get score $\hat{SCORE}_{\zeta}(x_{test}, y_{test})$, e.g. AUC and GAUC. This score combined with latency or throughput constitutes the final reward for this episode. All layers share one reward in a episode, which means each transition is (s_l, a_l, R, s_{l+1}) where R is the reward obtained after APAS one-shot pruning. APAS share one reward among all layers, because both $\hat{SCORE}_{\zeta}(x_{test}, y_{test})$ and m_{ζ_e} are decided by the model as a whole not by any individual part alone.

In this work, we use DDPG, a model-free, off-policy actor-critic algorithm, to control over the soft thresholding λ. We construct our exploration policy as follow:

$$\mu'(s_l) \sim \mu(s_l|\theta^{\mu}) + \mathcal{N} \tag{5}$$

where $\mu(s_l|\theta^{\mu})$ is the actor policy generated by the actor network based on layer l's state. $\mathcal{N}$ is the noise process that help the agent to explore in continuous action spaces. Following the way of DDPG, we formulate empirical loss function for action-value function as:

$$L(\theta^{Q}) = \frac{1}{N}\sum_{i}(y_i - Q(s_i, a_i|\theta^{Q}))^2 \tag{6}$$

$$y_i = r_i + \eta Q(s_{i+1}, \mu(s_{i+1})|\theta^{Q}) \tag{7}$$

where we sample a random minibatch of N transitions (s_i, a_i, r_i, s_{i+1}) each time from replay buffer. We can simply define the reward as any metric that evaluate the model prediction, e.g. AUC and GAUC. In this way, we could keep training till the performance metric is satisfied, and then, only use prediction scores as the reward. In this manner, the agent will learn each layer's importance towards the final prediction. Alternatively, we could also apply a reward shaping strategy [12] to incorporate the computation budget constraint. To be specific, the action would incurs a penalty if the constraint is violated defined in formulation (2) :

$$r = score - \beta \times \max(0, \mathrm{m}_{\zeta} - \mathrm{C}) \tag{8}$$

Even thought this strategy is naive and simple, it enables the DDPG agent to encourage higher prediction score, and meanwhile, penalize the model to arrive at the constraint.

3.6 APAS-light

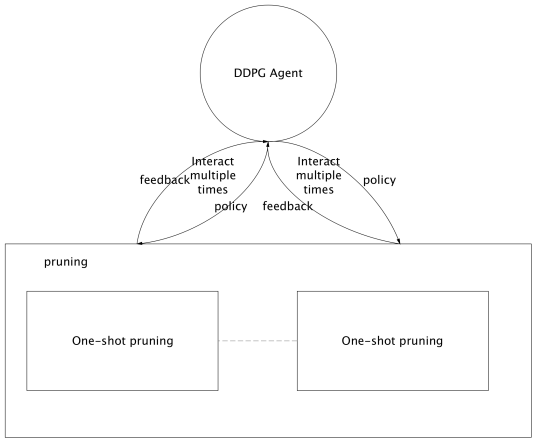

Figure 5: Illustration of the interactions of DDPG agent in APAS-light pipeline. APAS-light requires less training iterations for producing the optimal model structure. In APAS-light pipeline, DDPG agent interacts with APAS during the iterative pruning instead of repeating one-shot pruning several times.

We propose APAS-light to further accelerate the pruning process, which can complete the pruning task with much fewer iterations. Originally, we formulate model pruning as an single-step decision making problem. Namely, each episode of the APAS pipeline receives only one reward after each APAS one-shot pruning finished. Even though the reward in this way is quite unbiased, it is so delayed and sparse that require several one-shot pruning processes to train the DDPG agent sufficiently. Owing to the pruning-while-training paradigm, we could actually consider the transition dynamics and discretize the one-shot pruning process, which can enrich the samples and solve the problem of reward's sparsity. We describe APAS-light pruning pipeline in Algorithm 3, which replaces Algorithm 2 in Algorithm 1. In detail, the APAS pipeline is built upon iterative pruning and l1 penalization for scaling factors. Hence, we can discretize one episode during the masking pause of iterative pruning, and append several transitions, i.e. (s_i, a_i, r_i, s_{i+1}) to replay buffer in one training pass. APAS-light can finish the pruning with much fewer training passes. The trial and error will terminate once the interactions with DDPG agent are sufficient enough to learn the pruning policy well. In real-world system, the iterative pruning process can actually provide sufficient samples to train the DDPG agent where models are usually trained in distributed systems with large volume of data.

Algorithm 3 APAS-light pruning

Input: $M_{\zeta_{full}}$, e, R, Q
Add scaling factor η_j for each neuron n_j
Initialize $t = 0$
while ($m_{\zeta_{e_t}} \geq C$):
  if $t\%T == 0$:
    Mask neurons and get $M_{\zeta_{e_t}}$.
    Fine-tune $M_{\zeta_{e_t}}$ for several iterations
    Evaluate $m_{\zeta_{e_t}}$ and get $\hat{SCORE}_{\zeta}(x_{test}, y_{test})$
    Append $(s_{l,e_t}, a_{l,e_t}, r_{e_t}, s_{l+1,e_t})$ to replay buffer R
    Append ζ_{e_t} to model queue Q
    Agent generates layerwise soft-thresholding upwardly.
    Update DDPG agent
  else:
    Continue pruning by penalizing scaling factors.
  $t+ = 1$
end while

4 EXPERIMENTS

4.1 Datasets, Models and Experimental Setup

Amazon Dataset contains product reviews and metadata from Amazon. We use the Books subset of Amazon dataset which is comprised of user behavior logs between May 1996 and July 2014. We take all the user reviews as user click behaviors and construct user behavior sequences through reviews as well. All behavior sequences are truncated at length 100.

Taobao Dataset is a collection of user behaviors from Taobao's recommender system. The dataset contains different types of user behaviors including click, purchase. User behavior sequences from nearly one million users are contained in the dataset. We take click as the target and construct user behavior sequences through clicks as well. All behavior sequences are truncated at length 200.

Wide and Deep has two main modules: linear model and deep model. Wide and deep model jointly trains wide linear models and deep neural networks to combine the benefits of memorization and generalization for recommender systems.

Deep Interest Network (DIN) is dedicated to adaptively learn the representation of user interests from historical behaviors with respect to a target item. DIN widely adopts attention mechanism to capture user's multi-modal interests.

Deep Interest Evolution Network(DIEN) derives from DIN. Besides, it contains an interest evolving layer to capture interest evolving process. In DIEN, both attention mechanism and gate recurrent unit(GRU) are widely used to further model the user's behaviour sequence.

Experimental Setup. We use Adam solver and apply exponential decay with the learning rate starting at 0.001 and decay rate of 0.5 for model optimization. Every model's layout exactly follows its original source code in github. And we evaluate the model prediction by AUC, which is widely used in real-world systems. For both APAS and APAS-light pipelines, all scaling factors are initialized to be 1. For reinforcement learning, the continuous action for each layer is in the range of $[0, 1]$. Both action network and critic network have two fully connected layers, each with 300 neurons. We use the soft target updates with $\tau = 0.01$ and train the action/critic network with batch size 64. The DDPG agent first warms up by 50 episodes with a constant noise $\sigma = 0.5$, and then continues training by 150 episodes with exponentially decayed noise σ. In APAS-light pipeline, we shorten these two processes correspondingly because samples can be enriched during iterative pruning.

4.2 Results on Public Datasets

As shown in Table 1, FLOPs is set as the constraint in this experiment. All the pruned model only need 50% FLOPs of computation for inference compared with the unpruned model. We can see that APAS outperforms AutoML for Model Compression (AMC) and naive scaling factor pruning based on proximal gradient(PG). Moreover, our pruned model even outperforms the unpruned one, which illustrates the hand-crafted model structure is suboptimal and redundant in this case. Besides, the APAS-light can also achieve comparable performance with APAS.

As shown in Table 2, we set latency as the constraint in this experiment. We decrease the latency for Wide and Deep model(W&D) and Deep Interest Network(DIN) by 5% and 10% respectively. When we set the latency as the constraint, it shows that APAS is still superior over AMC and PG method. Notably, pruning based on latency is much more challenging than on FLOPs. For example, even though we decrease latency by 5% for W&D model, we've already cut down 90% FLOPs correspondingly.

Table 1: Model performance (AUC) on public datasets – FLOPS

Dataset	Model	No prune	AMC	PG prune	APAS	APAS light
Amazon	W&D	0.7731	0.7673	0.7684	**0.7732**	**0.7751**
	DIN	0.7901	0.7865	0.7791	**0.7910**	0.7873
	DIEN	0.8476	0.8424	0.8409	**0.8513**	0.8505
taobao	W&D	0.8690	0.8640	0.8660	**0.8665**	**0.8669**
	DIN	0.8793	0.8715	0.8613	**0.8875**	0.8773
	DIEN	0.9057	0.8948	0.8994	**0.9053**	0.9034

Table 2: Model performance (AUC) on public datasets – Latency

Dataset	Model	No prune	AMC	PG prune	APAS	APAS light
Amazon	W&D	0.7731	0.7605	0.7579	**0.7625**	0.7598
	DIN	0.7901	0.7839	0.7441	**0.7911**	0.7892
taobao	W&D	0.8690	0.8595	0.8593	**0.8637**	0.8621
	DIN	0.8793	0.8700	0.8632	**0.8765**	0.8736

4.3 Online A/B Testing

We deploy APAS pipeline in a world-leading display advertising system, where we optimized a bunch of model structures at different stages. Real-world recommender system follows a multi-stage

cascade architecture, which usually consist of candidate generation, pre-ranking, ranking ,etc. We apply APAS to models at pre-ranking and ranking stages. These stages are responsible for CTR prediction, where models are usually most complicated in the whole system. We follow the APAS pipeline by setting GAUC as the objective. After model pruning, the change of GAUC is almost trivial for any model at pre-ranking stage and ranking stage. For online A/B testing, we evaluate the models by effective cost per mille (eCPM) and CTR directly. Online A/B testing is performed from 2020-08-12 to 2020-09-12 in two commercial scenarios. In one scenario where the deep CTR model consists of several fully-connected layers. Our pruned CTR model in ranking modules improves the throughput by 40%, eCPM by 2.25%, CTR 0.91% and decrease the latency by 30%. In another scenario, the deep CTR model contains multi-heads attention layers, gate recurrent unit (GRU) layers and fully-connected layers. Our pruned CTR model in ranking modules improves the throughput by 10%, eCPM by 0.36%, and decease the latency by 10%. For model in pre-ranking, the pruned CTR model improves the throughput by 20%, decrease the latency by 20% and maintain the same eCPM and CTR. In production environment, APAS brings about an improvement to the performance of model inference significantly. Moreover, as shown in online A/B testing, APAS can even bring higher eCPM's and and CTR's, which demonstrates that model pruning can contribute to the model prediction by reasonably controlling the model complexity.

4.4 Model Structure Analysis

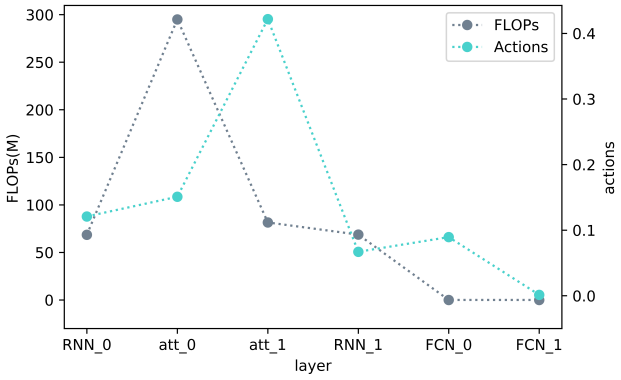

Figure 6: Illustration of layerwise FLOPs and actions for DIEN model on Amazon dataset. APAS penalizes the attention layers and RNN layers with relatively higher magnitude.

The layerwise actions from DDPG agent could provide us some insights to analyse model structure. As illustrated in previous sections, these actions are actually the results of trading off between model complexity and accuracy learned by the agent. For example, some layers can be penalized to a much greater extent because of their redundancies. Others can probably receive mild or even no penalization because they are the important parts for model prediction. Hence, APAS pipeline can be taken as a powerful tool for model structure analysis.

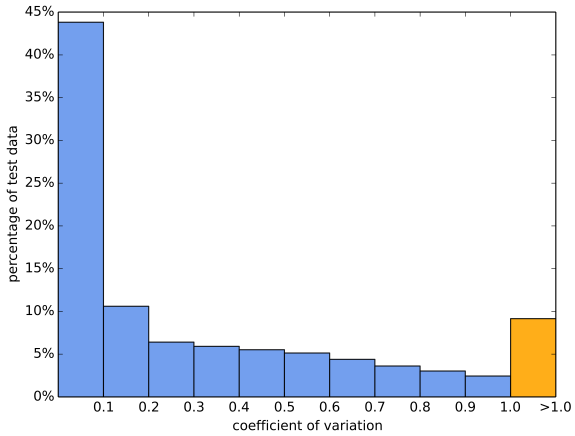

Figure 7: Illustration of the coefficient of variation for the attention layer's outputs. We analyse all samples from test dataset, predict through the pruned model and collect the outputs of attention layer. The outputs with lower variance account for the majority.

In Figure 6, we take as an example that DIEN model on Amazon dataset. From the view of model complexity, we show the relation between layerwise FLOPs and its corresponding actions generated by APAS agent. In this analysis, we use FLOPs as the constraint. In DIEN model, the FLOPs of attention layers account for the largest proportion of the total amount. The APAS agent automatically assigns this part with higher penalization by increasing the magnitude of soft-thresholding λ.

Besides, we also analyze the importance of attention layers towards model prediction in Figure 7. We use coefficient of variation to analyze the outputs of attention layers, which is defined as the ratio of the standard deviation to the mean. In recommender system, the attention mechanism is used to capture the relevance between user behaviour sequence and target item. It plays an important role in model prediction when different parts of user behaviour sequence have different degrees of relevance with the target item. In this work, we specifically analyse the output of second attention layer which receives the greatest penalization from APAS pipeline. We calculate the coefficient of variation for the weight vector generated by second attention layer to reveal its deviation from mean for each sample. The samples with lower coefficient of variation account for the majority, which means that these weights are almost equal to the average in these cases. In other words, the attention mechanism doesn't have much influence on the majority of samples as it almost assigns same weight for each state. Hence, the attention mechanism is probably redundant for this particular dataset when trading off between model complexity and prediction. In this case study, we empirically show that APAS pipeline can automatically learn which parts of the model contains more redundancy, and prune the unnecessary modules according to the rewards. Moreover, APAS actually provides us with a tool to analyze the each

layer's performance because the actions generated by DDPG agent can reveal the balance between model speedup and accuracy.

5 CONCLUSION

In this paper, we device an end-to-end AutoML pipeline, named APAS, in real-world systems, which could automatically trade off model speedup and accuracy. The APAS has been deployed in a world-leading display advertising system, where various model structures in several scenarios could be pruned by APAS pipeline without loss of accuracy. Besides, by leveraging reinforcement learning, arbitrary metrics can be introduced into APAS pipeline and optimized directly, which is vital to real-world systems. Moreover, we also propose APAS-light pipeline that can achieve comparable performances with significantly less computation.

In the future, we will empower APAS pipeline with the ability to optimize real-world models comprehensively. At current stage, APAS pipeline can only optimize the model inference through structured pruning. However, the bottleneck for serving performance cannot be simply attributed to computing overload. Model pruning can only solve part of the problem, when the model performance is bounded by computation. In real-world systems, many techniques like batching, caching, kernel fusion can also improve the model serving performance. All these techniques now requires domain expertise to put into use. In principle, these human heuristics can be substituted by learning-based optimization policy as well. We will further develop APAS pipeline with more actions and optimization policy to integrate with these techniques subsequently.

REFERENCES

[1] Yoshua Bengio, Aaron Courville, and Pascal Vincent. 2013. Representation learning: A review and new perspectives. *IEEE transactions on pattern analysis and machine intelligence* 35, 8 (2013), 1798–1828.

[2] Giovanna Castellano, Anna Maria Fanelli, and Marcello Pelillo. 1997. An iterative pruning algorithm for feedforward neural networks. *IEEE transactions on Neural networks* 8, 3 (1997), 519–531.

[3] 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.

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

[5] Shaopeng Guo, Yujie Wang, Quanquan Li, and Junjie Yan. 2020. DMCP: Differentiable Markov Channel Pruning for Neural Networks. In *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition*. 1539–1547.

[6] Song Han, Huizi Mao, and William J Dally. 2015. Deep compression: Compressing deep neural networks with pruning, trained quantization and huffman coding. *arXiv preprint arXiv:1510.00149* (2015).

[7] Song Han, Jeff Pool, John Tran, and William Dally. 2015. Learning both weights and connections for efficient neural network. In *Advances in neural information processing systems*. 1135–1143.

[8] Yihui He, Ji Lin, Zhijian Liu, Hanrui Wang, Li-Jia Li, and Song Han. 2018. Amc: Automl for model compression and acceleration on mobile devices. In *Proceedings of the European Conference on Computer Vision (ECCV)*. 784–800.

[9] Yang He, Ping Liu, Ziwei Wang, Zhilan Hu, and Yi Yang. 2019. Filter pruning via geometric median for deep convolutional neural networks acceleration. In *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*. 4340–4349.

[10] Hengyuan Hu, Rui Peng, Yu-Wing Tai, and Chi-Keung Tang. 2016. Network trimming: A data-driven neuron pruning approach towards efficient deep architectures. *arXiv preprint arXiv:1607.03250* (2016).

[11] Zehao Huang and Naiyan Wang. 2018. Data-driven sparse structure selection for deep neural networks. In *Proceedings of the European conference on computer vision (ECCV)*. 304–320.

[12] Adam Daniel Laud. 2004. *Theory and application of reward shaping in reinforcement learning*. Technical Report.

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

[14] Hao Li, Asim Kadav, Igor Durdanovic, Hanan Samet, and Hans Peter Graf. 2016. Pruning filters for efficient convnets. *arXiv preprint arXiv:1608.08710* (2016).

[15] Timothy P Lillicrap, Jonathan J Hunt, Alexander Pritzel, Nicolas Heess, Tom Erez, Yuval Tassa, David Silver, and Daan Wierstra. 2015. Continuous control with deep reinforcement learning. *arXiv preprint arXiv:1509.02971* (2015).

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

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

[18] Zhuang Liu, Jianguo Li, Zhiqiang Shen, Gao Huang, Shoumeng Yan, and Changshui Zhang. 2017. Learning efficient convolutional networks through network slimming. In *Proceedings of the IEEE International Conference on Computer Vision*. 2736–2744.

[19] Zhuang Liu, Mingjie Sun, Tinghui Zhou, Gao Huang, and Trevor Darrell. 2018. Rethinking the value of network pruning. *arXiv preprint arXiv:1810.05270* (2018).

[20] Jian-Hao Luo, Jianxin Wu, and Weiyao Lin. 2017. Thinet: A filter level pruning method for deep neural network compression. In *Proceedings of the IEEE international conference on computer vision*. 5058–5066.

[21] Pavlo Molchanov, Stephen Tyree, Tero Karras, Timo Aila, and Jan Kautz. 2016. Pruning convolutional neural networks for resource efficient inference. *arXiv preprint arXiv:1611.06440* (2016).

[22] Qi Pi, Guorui Zhou, Yujing Zhang, Zhe Wang, Lejian Ren, Ying Fan, Xiaoqiang Zhu, and Kun Gai. 2020. Search-based User Interest Modeling with Lifelong Sequential Behavior Data for Click-Through Rate Prediction. In *Proceedings of the 29th ACM International Conference on Information & Knowledge Management*. 2685–2692.

[23] Robert Tibshirani. 1996. Regression shrinkage and selection via the lasso. *Journal of the Royal Statistical Society: Series B (Methodological)* 58, 1 (1996), 267–288.

[24] Wei Wen, Chunpeng Wu, Yandan Wang, Yiran Chen, and Hai Li. 2016. Learning structured sparsity in deep neural networks. In *Advances in neural information processing systems*. 2074–2082.

[25] Zhonghui You, Kun Yan, Jinmian Ye, Meng Ma, and Ping Wang. 2019. Gate decorator: Global filter pruning method for accelerating deep convolutional neural networks. In *Advances in Neural Information Processing Systems*. 2133–2144.

[26] Ruichi Yu, Ang Li, Chun-Fu Chen, Jui-Hsin Lai, Vlad I Morariu, Xintong Han, Mingfei Gao, Ching-Yung Lin, and Larry S Davis. 2018. Nisp: Pruning networks using neuron importance score propagation. In *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*. 9194–9203.

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

[28] 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.

[29] Han Zhu, Daqing Chang, Ziru Xu, Pengye Zhang, Xiang Li, Jie He, Han Li, Jian Xu, and Kun Gai. 2019. Joint optimization of tree-based index and deep model for recommender systems. In *Advances in Neural Information Processing Systems*. 3971–3980.

[30] Han Zhu, Xiang Li, Pengye Zhang, Guozheng Li, Jie He, Han Li, and Kun Gai. 2018. Learning tree-based deep model for recommender systems. In *Proceedings of the 24th ACM SIGKDD International Conference on Knowledge Discovery & Data Mining*. 1079–1088.

[31] Jingwei Zhuo, Ziru Xu, Wei Dai, Han Zhu, Han Li, Jian Xu, and Kun Gai. 2020. Learning Optimal Tree Models under Beam Search. In *Proceedings of the International Conference on Machine Learning*.

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