Latency-aware Neural Architecture Performance Predictor with Query-to-Tier Technique

Bicheng Guo, *Student Member, IEEE*, Lilin Xu, *Student Member, IEEE*, Tao Chen *Senior Member, IEEE*, Peng Ye, Shibo He, *Senior Member, IEEE*, Haoyu Liu, Jiming Chen, *Fellow, IEEE*

***Abstract*—Neural Architecture Search (NAS) is a powerful tool for automating effective image and video processing DNN designing. The ranking of the accuracy has been advocated to design an efficient performance predictor for NAS. The previous contrastive method solves the ranking problem by comparing pairs of architectures and predicting their relative performance. However, it only focuses on the rankings between the two involved architectures and neglects the overall quality distributions of the search space, which may suffer generalization issues. On the contrary, we propose to let the performance predictor concentrate on the global quality level of specific architecture, and learn the tier embeddings of the whole search space automatically with learnable queries. The proposed method, dubbed as Neural Architecture Ranker with Query-to-Tier technique (NARQ2T), explores the quality tiers of the search space globally and classifies each individual to the tier they belong to. Thus, the predictor gains knowledge of the performance distributions of the search space which helps to generalize its ranking ability to the datasets more easily. Thanks to the encoder-decoder design, our method is able to predict the latency of the searched model without deteriorating the performance prediction. Meanwhile, the global quality distribution facilitates the search phase by directly sampling candidates according to the statistics of quality tiers, which is free of training a search algorithm, e.g., Reinforcement Learning or Evolutionary Algorithm, thus it simplifies the NAS pipeline and saves the computational overheads. The proposed NARQ2T achieves state-of-the-art performance on two widely used datasets for NAS research. Moreover, extensive experiments have validated the efficacy of the designed method.**

***Index Terms*—Neural Architecture Search, performance predictor, ranking problem, latency.**

I. Introduction

Part of this work was presented at the ACM Multimedia 2022 and appeared in the conference proceedings. This work was supported by the National Natural Science Foundation of China under grant No. U1909207, and NSFC 62088101 Autonomous Intelligent Unmanned Systems. This article was recommended by Associate Editor Jungong Han. (*Corresponding author: Shibo He*)

B. Guo, L. Xu, and J. Chen are with the College of Control Science and Engineering, Zhejiang University, Hangzhou, Zhejiang, 310027, China. E-mail: {guobc, lilinxu, cjm}@zju.edu.cn. (*Bicheng Guo and Lilin Xu have made equal contributions in terms of technical expertise.*)

S. He is with the College of Control Science and Engineering, Zhejiang University, Hangzhou, Zhejiang, 310027, China, and with the Key Laboratory of Collaborative Sensing and Autonomous Unmanned Systems of Zhejiang Province. E-mail: s18he@zju.edu.cn.

T. Chen and P. Ye are with the School of Information Science and Technology, Fudan University, Shanghai 200433, China. E-mail: {eetchen, yepeng20}@fudan.edu.cn.

H. Liu is with the Fuxi AI Lab, NetEase Games, Hangzhou, Zhejiang, 310056, China. E-mail: liuhaoyu03@corp.netease.com.

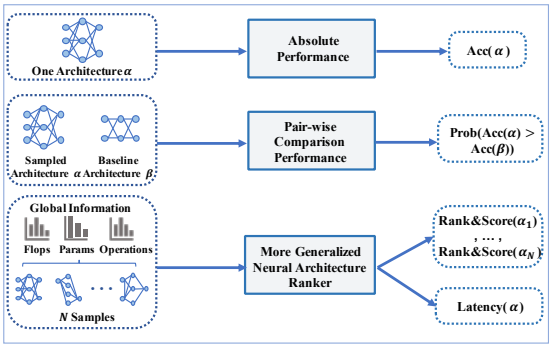


Fig. 1. Comparison between previous performance predictors (top and middle) and ours (bottom). Different from predicting the absolute or pair-wise comparison performance, the NARQ2T ranks the architectures with quality tier and scores them relative metric. Thanks to the encoder-decoder design, the NARQ2T is able to predict the latency of the searched architectures without deteriorating the performance prediction.

DEEP neural networks (DNNs) have received great attention in recent years. Many DNNs that are artificially designed by researchers have been applied to various scenarios, such as image classification [1], [2], object detection [3]–[5], semantic segmentation [6], [7], image/video coding [8]–[10] and other real-world applications [11]–[17]. Though these artificially designed DNNs have been proven to be powerful, designing them requires rich human expertise and is labor-intensive. Furthermore, dedicated knowledge is needed in the network architecture design for many specific target domains.

Neural architecture search (NAS) offers a powerful tool for automating effective DNN designing for specific objectives, for example, recent work [18] uses the NAS technique to generate watermarked architectures that can uniquely represent ownership. Previous studies directly apply different search and optimization methods, including Reinforcement Learning (RL) [19], [20], Evolutionary Algorithm (EA) [21], [22], and differentiable methods [23]–[27], to find candidates in the search space. In order to reduce the prohibitive cost of evaluating a population of candidates, performance predictor is proposed to replace the evaluation metrics with the predicted performance of architectures [28]–[30]. However, these predictors approximate the absolute performance of architectures and suffer the ranking problem, i.e., the architectures with similar ground-truth performance have incorrectly predicted rankings due to the prediction bias, as shown in the top of Fig. 1. As a result, the search algorithm could be misled to select the

low-ranking architectures [31].

The most recent contrastive method [32] solves the ranking problem by comparing pairs of sampled architectures and calculating the probability that one architecture is better than the other, as shown in the middle of Fig. 1. Even though it can achieve good ranking ability, such an approach may suffer from generalization issues. This is because it only focuses on the rankings between the two involved architectures and neglects the overall quality distributions of the search space. As a result, the predictor is limited only to the architecture ranking instead of generalizing its ranking ability over various target datasets. To solve this, we propose to utilize the quality distributions of the search space where the architectures are classified into multiple quality tiers by their global rankings according to the ground-truth performance. In this way, we can train the predictor by learning the features of quality tiers and classifying each individual to the tier they belong to according to its global ranking. This classification-enhanced ranking paradigm has also been explored previously [33]. The rationale, *class membership information is important for determining a quality ranking over a dataset* [34], inspires us to first roughly classify the quality of the architecture and then score them. As a result, the predictor gains global knowledge of the performance distributions of the search space which helps to generalize the ranking ability to various datasets more easily than previous local methods.

Besides the accuracy ranking, computational properties such as latency are vital for the deployment of the searched architectures, especially for mobile devices [35]. However, most of the performance predictors merely focus on predicting accuracy and associated ranking [31], while they cannot handle accuracy and latency simultaneously. The reason is that those methods combine two architecture features for predicting relative ranking, which cannot be used for single-model property prediction. We solve this by designing an encoder-decoder structure that handles a single architecture feature in the encoder for property prediction, and performs the comparison of architectures and classification of the architectures' quality in the decoder. We demonstrate that it is possible to predict the task-related metrics, i.e., relative score and global quality ranking, and the device-related properties such as latency. Moreover, we can still ensure competitive accuracy prediction performance, even given a smaller search space when considering the device constraints. This is critical for deploying the searched architectures to real-world scenarios, which makes our architecture search pipeline more practical.

During the search phase, most of the previous studies either adopt reinforcement learning [19], [36] or evolutionary algorithms [37], [38], which requires additional training cost and complicates the NAS pipeline. Other method searches candidates by sampling with the distributions which are approximated by a subset of architectures from the search space [39]. Interestingly, we can benefit from collecting the distributions of the top-quality tiers and focus on the outperformed architectures by directly sampling with them in the search space. This makes our method free of training an RL controller or employing EA methods, thus saving computational overheads.

In this work, we advance the original Neural Architecture Ranker (NAR) [40] with the Query-to-Tier (Q2T) technique, dubbed as NARQ2T. By automatically learning the tier embeddings with learnable queries, we can drop the previous manual tier embedding updating, enabling an end-to-end training style and saving the search costs significantly: almost 30% for training the NARQ2T. Thanks to the encoder-decoder design which handles the single architecture feature and compares with tiers embeddings separately, we can easily predict the computational properties in the encoder while not affecting the performance prediction. For the performance prediction, we relax it into a quality classification problem by learning the global distributions of the architectures with various qualities and identifying each individual's quality level (tier) among the search space according to its performance. Specifically, we first divide the search space into five quality tiers according to the performance distributions of the architectures. Then, each individual architecture is encoded to represent its structural and computational features and is matched with the embeddings of all tiers alternately to decide which tier it belongs to. We also leverage the extracted feature to predict the relative scores of the sampled architectures. Consequently, the NARQ2T is capable of ranking and scoring the candidates according to their global ranking in the search space. Furthermore, the distributions of the different quality tiers are collected to guide the sampling procedure in the search phase, which requires no additional computational overheads for searching and simplifies the pipeline. We empirically demonstrate the effectiveness of the proposed method. The strong representation ability of NARQ2T helps to achieve state-of-the-art average results: 94.11% on NAS-Bench-101 [41]; 94.35%, 73.22% and 46.93% on three datasets of the NAS-Bench-201 [42]. We further perform more comprehensive ablation studies to verify the effectiveness of the designed components.

The overall contribution is summarized as follows:

- Different from locally comparing architecture pairs, we propose NARQ2T that matches architectures to various quality tiers, hence the rank of each architecture can be obtained from a global perspective. Meanwhile, the latency of the architecture is predicted simultaneously.
- We design a Q2T technique that enables automatic tier embedding learning. In this way, an end-to-end training style can be reached without any more manual efforts, so that the training overheads can be reduced.
- We propose to collect the distributions with different quality tiers to guide the sampling during the search, which reduces cost compared with previous methods.
- We achieve state-of-the-art results on two widely used NAS datasets. On NAS-Bench-101, our method finds the architecture with top 0.02‰ performance among the vast search space of 423k individual architectures only by sampling, outperforming all other RL and EA methods. On NAS-Bench-201 with three different image datasets, i.e., CIFAR-10, CIFAR-100, and ImageNet-16-120, NARQ2T generalizes well and is able to identify the optimal architecture on each of them.

The rest of this paper is organized as follows. A brief review of the recent research work and advanced methods related to

NAS and the performance predictor is presented in Sec. II. The proposed method and algorithm implementation are explained in Sec. III. The experimental results and ablation studies are reported in Sec. IV. The conclusions follow in Sec. V.

II. Related Work

A. Neural architecture search

NAS offers to automate the design procedure of an efficient neural network given scenario constraints. It is often formulated as a constrained optimization problem:

$$\begin{aligned} &\min_{\alpha \in \mathcal{A}} \mathcal{L}\left(W_{\alpha}^{*}; \mathcal{D}_{val}\right), \\ \text{s.t.} &W_{\alpha}^{*} = \arg\min_{W_{\alpha}} \mathcal{L}\left(W_{\alpha}; \mathcal{D}_{trn}\right), \\ &\operatorname{cost}(\alpha) < \tau, \end{aligned} \tag{1}$$

where W_α are the weights of the architecture α, $\mathcal{A}$ denotes the search space, $\mathcal{D}_{trn}$ and $\mathcal{D}_{val}$ mean the training and validation set, respectively, $\mathcal{L}(\cdot)$ is the loss function, and $\operatorname{cost}(\alpha)$ denotes the computational cost with respect to α, e.g., FLOPs, #parameters[1] or latency for different devices. Pioneering work applies RL [19], [20], [36], [43] and EA [21], [22], [44] to select α to evolve into the training and validation procedure which endures prohibitive cost.

In order to save the evaluation cost and simplify the optimization difficulties, two-stage NAS decouples the training and searching into two separate steps [37], [39], [45]. The first step is to jointly optimize all the candidates in one supernet:

$$\min_{W} \mathbb{E}_{\alpha \in \mathcal{A}}\left[\mathcal{L}\left(W_{\alpha}; \mathcal{D}_{trn}\right)\right]. \tag{2}$$

Then, RL or EA methods are utilized to select the subnet with the best performance from the supernet given the constraints during the second search phase:

$$\begin{aligned} \{\alpha_i^*\} = &\arg\min_{\alpha_i \in \mathbf{A}} \mathcal{L}\left(W_{\alpha_i}^{*}; \mathcal{D}_{val}\right) \\ &\text{s.t. } \operatorname{cost}(\alpha_{\mathrm{i}}) < \tau_i, \forall\ i, \end{aligned} \tag{3}$$

where W^* denotes the sharing weights inheriting from the trained supernet yielded in Eq. 2.

Much attention has been paid to improving the training quality and efficiency of the supernet in Eq. 2, e.g., progressive shrinking [29], sandwich rule [46], and attentive sampling [47]. During the search phase, all of the existing methods either train an RL controller or employ the EA to select the candidates with top quality. In this paper, we try to avoid such costs by collecting the distribution of outperformed architectures in the process of training the predictor and sampling according to them directly in the search phase.

B. Predictor and the Ranking problem

To further boost the training and searching efficiency, the performance predictor is widely accepted both in one-stage and two-stage NAS [48]. FBNetv3 [49] applies a multilayer perceptron to predict the accuracy and corresponding training recipe simultaneously. When training the supernet, the predictor is utilized to facilitate the Pareto-aware sampling which improves the upper or lower performance bounds of the supernet [47]. Predictors can also be used to generate the weights [50] or predict the latency of the network which pushes NAS to focus more on the hardware efficiency [51].

Recently, the ranking problem has attracted significant attention. For weight-sharing NAS, individual architectures with different parameters are actually shared with others in the supernet and this causes ranking disorder between standalone architectures and corresponding shared-weight networks. To tackle this, a set of landmark architectures with known standalone performance is selected to calculate a regulation term during the training of the weight-sharing supernet [52]. Knowledge distillation is also a promising way to address this issue. During the supernet training, subnetworks with superior performance are selected as prioritized paths to transfer their knowledge to others, and the overheads of training a searching algorithm are waived by directly selecting the best architectures from the prioritized paths [53]. For performance predictors, previous studies adopt absolute performance as a metric to guide the following search procedure. However, the poor ranking correlation between ground-truth performance and the evaluated metric of the architectures could deteriorate the search results, since the incorrectly predicted rankings caused by the prediction bias will mislead the search algorithm to select the low-ranking architectures. In this case, the predictor-based NAS algorithms focus on learning the relative ranking of the neural architectures instead of the absolute ones and can achieve state-of-the-art results. ReNAS [32] learns the correct rankings between pairs of architectures by leveraging a ranking loss to punish the disordering predicted metric. CTNAS [31] directly compares two architectures and predicts the probability of one being better than the other. These works solve the ranking problem issues of designing performance predictors from a local perspective, i.e., regulating or comparing the pare-wise ranking. However, these methods may suffer from generalization issues due to the local perspective. To tackle this, we are the first to utilize the global quality distributions of the search space. Specifically, we take as input the feature tensor of the sampled architecture and classify the architecture according to its global quality tier ranking. Moreover, we propose the Query-to-Tier (Q2T) technique to automatically learn the embedding of the global quality tier, which brings higher average performance prediction and enables an end-to-end training style with 30% searching cost saved.

III. Q2T enhanced Neural Architecture Ranker

In this section, we advance the original Neural Architecture Ranker [40] with Q2T technique, dubbed as NARQ2T, which enables an end-to-end training style and strong tier representation ability. Moreover, thanks to the encoder design which decouples the individual feature handling and groups of architectures classification, both the original NAR and novel NARQ2T can predict the latency of the architectures by adding a prediction head, which is vital for searching and deploying DNN models for edge devices. For the accuracy prediction problem, we first relax it into a quality classification problem and select the candidates from the top quality tier.

[1]#parameters denotes the number of parameters.

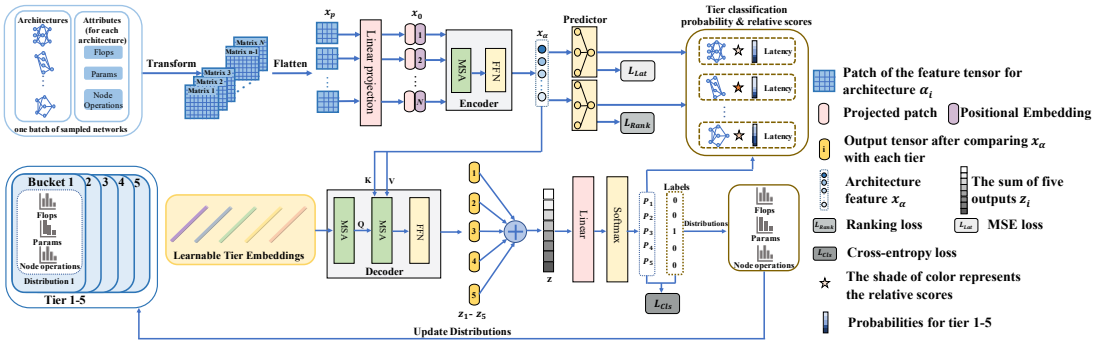


Fig. 2. The systematic illustration of the NARQ2T. The NARQ2T consists of an encoder and a decoder. The encoder extracts the feature of the sampled architecture and predicts its relative metric and latency. The decoder matches the extracted feature with the learnable tier embeddings and classifies it to the corresponding quality tier. The batch distributions are then updated to the corresponding tier according to the ground-truth performance of the architectures.

Algorithm 1: Overall Framework

Input: Search space Ω, # of Tiers N, # of batches t, # of batch size k, number of Epochs E, # of sampled archs k, constraints reuse limit R, # of candidate archs m, # of sampling iterations T_s.

// divide training data into five quality tiers.
$\mathcal{D}_{trn}, \boldsymbol{Q}_0, \Phi \leftarrow$ *Tier Dividing*(Ω, N, t, k) ◁ Algo. 2
// train the accuracy predictor.
$f^t_{\text{enc}}, f^t_{\text{dec}}, \Phi, \boldsymbol{Q}_t \leftarrow$ *Ranker Training*$(\mathcal{D}_{trn}, \boldsymbol{Q}_0, \Phi, E)$ ◁ Algo. 3
// search for promising architectures.
$\{\alpha_{cand}\}, \hat{y}_{\alpha_{cand}} \leftarrow$ *Arch Searching*$(f^t_{\text{enc}}, f^t_{\text{dec}}, \Phi, \boldsymbol{Q}_t)$ ◁ Algo. 4
// evaluation.
test accuracy and rank ← evaluate candidates $\{\alpha_{cand}\}$.
Return test accuracy, rank, latency.

Algorithm 2: Quality Tier Dividing.

Input: Search space Ω, # of Tiers N, # of batches t, # of batch size k.

$\Phi = \{\phi_{T_1}, \cdots, \phi_{T_N}\} \leftarrow$ initialize tier statistics set. // $\phi_{T_i} = \{\hat{\pi}^{\text{FLOPs}}_{T_i}, \hat{\pi}^{\text{Params}}_{T_i}, \hat{\pi}^{\text{Opt.}}_{T_i}\}$.
$\boldsymbol{Q}_0 \leftarrow$ initialize trainable tier embeddings.
$\{\alpha^m\} \leftarrow$ randomly sample a set of architectures from Ω.
$y_{\alpha^m}, h_{\alpha^m} \leftarrow$ calculate the accuracy and latency of α^m.
$\boldsymbol{x}^m_0 \leftarrow$ encode architecture α^m into embedding.
$\boldsymbol{l}^m_i = [\mathbb{I}(\alpha^m \in T_1), \cdots, \mathbb{I}(\alpha^m \in T_N)]^{\mathrm{T}} \leftarrow$ build label. //equally dividing Φ into N.
$\mathcal{D}_i = \{\boldsymbol{x}^m_0, y_{\alpha^m}, h_{\alpha^m}, \boldsymbol{l}^m_i\}^k_{m=1} \leftarrow$ build batch.
$\mathcal{D}_{trn} = \{\mathcal{D}_1, \cdots, \mathcal{D}_t\} \leftarrow$ build t batches of training data.
Return $\mathcal{D}_{trn}$, $\boldsymbol{Q}_0$, Φ.

During the classification, the distributions of the representative parameters of the architectures, e.g., FLOPs, #parameters, and node operations, are collected with respect to their quality. Finally, we utilize the distributions of the top tier to guide the sampling procedure, which is free of training an RL controller or employing EA in the search phase. The Overall Algorithm is in Algo. 1. The proposed method is divided into three stages: 1) Quality Tier Dividing in Algo. 2, 2) NARQ2T Training in Algo. 3, and 3) Architecture Searching in Algo. 4.

A. Model Design

We utilize the original Transformer [54] to implement the main procedures of *matching and classifying*. We first determine five different quality tiers $\mathcal{T} = \{T_1, T_2, T_3, T_4, T_5\}$, according to the ground-truth performance of architectures. For specific architecture α, $\alpha \in T_1$ indicates that the performance of the architecture is among the top 20% rank. The Quality Tier Dividing algorithm is shown in Algo. 2. Specifically, a batch of architectures from the search space is randomly sampled from the search space. The architectures are encoded into feature tensors following the methods discussed below, and we obtain their ground-truth accuracy and latency. We then divide the entire search space into five quality tiers and take the rank of each architecture as the classification label. Meanwhile, we initialize an empty set to store the distributions of quality tiers and the learnable query embeddings to represent the tiers. Then, we match each sampled architecture with the embeddings representing the quality tiers, and classify it into the corresponding tier. The training pipeline of the proposed NARQ2T is shown in Fig. 2.

Architecture encoding. Similar to ReNAS [31], we encode each architecture of NAS-Bench-101 [41] and NAS-Bench-201 [42] datasets into feature tensor, $\mathcal{X} \in \mathbb{R}^{N \times P \times P}$, where N denotes the number of the patches and (P, P) is the patch resolution. Based on these two datasets, we have released the detailed cell information datasets, including node FLOPs and #parameters in each cell, as well as the feature tensor encoding codes to boost the NAS research one step further.

Specifically, for NAS-Bench-101, each cell of the network contains at most 7 nodes, and there is a total of 9 cells. Nodes in each cell are denoted as operations and edges as connections. Following ReNAS [31], cell connection is modeled by an adjacent matrix $\mathcal{A} \in \{0, 1\}^{7 \times 7}$. When nodes are less than 7, we pad the missing rows and columns with 0. An operation type vector $\mathbf{o} \in \{1, \cdots, 5\}^5$ is built to represent `IN` node, 1×1 convolution, 3×3 convolution, 3×3 max-pooling and `OUT` node, respectively. Besides, in order to represent the computational ability of the cell, the FLOPs and #parameters of each node are calculated and formed into FLOP vector $\mathbf{f} \in \mathbb{R}^7$ and #parameters vector $\mathbf{p} \in \mathbb{R}^7$ for all nodes in each cell, where the missing nodes are padded with 0 when nodes of the cell are less than 7. Vectors $\mathbf{o}$ are broadcast into matrices first, and then element-wisely multiplied with the adjacent matrix $\mathcal{A}$ to form the operation matrix $\mathbf{O}$. Vectors $\mathbf{f}$ and $\mathbf{p}$ of each cell are transformed into the FLOPs matrix $\mathbf{F}$ and #parameters matrix $\mathbf{P}$ in the same way. Finally, operation

matrix $\mathbf{O}$ and matrices of all cells are concatenated into one feature tensor $\mathcal{X}$, whose size is (19, 7, 7).

For NAS-Bench-201, each node represents the sum of the feature maps and each edge as an operation. Every architecture has fixed 4 nodes and we build an adjacent matrix $\mathcal{A} \in \{0,1\}^{4\times4}$ without padding. The operation vector $\mathbf{o} \in \{0,\cdots,4\}^5$ represents zeroize, skip connect, 1×1 convolution, 3×3 convolution and 3×3 avg-pooling. The FLOPs vector $\mathbf{f} \in \mathbb{R}^4$ and the #parameters vector $\mathbf{p} \in \mathbb{R}^4$ are obtained identically and broadcast into matrices. We concatenate those matrices according to the order of the cells and obtain the final feature tensor $\mathcal{X}$ with size (31, 4, 4). Every two matrices, from the second matrix to the last, correspond to the FLOPs and #parameters of nodes for each cell, except for the first matrix representing the operations.

A key difference between the NAS-Bench-101 and NAS-Bench-201 is the role of nodes and edges in the building cell, leading to different choices for candidate operations. For the cell in NAS-Bench-101, each node represents an operation as discussed above, and each edge serves as the feature map flow, whereas, for the cell in NAS-Bench-201, each node represents the sum of the feature maps and each edge serves as an operation. Apart from having the same 3×3 convolution and 1×1 convolution, the NAS-Bench-201 has a different average-pooling operation. Since NAS-Bench-201 cannot approximate skip-connection by connecting two nodes without an inter-node (the way in NAS-Bench-101), it has to include a skip operation for the choice of edge. Moreover, a zeroize operation is included to indicate the drop of the associated edge.

Once the feature tensor is obtained, the $\mathcal{X}$ is reshaped into a sequence of flattened patches $\boldsymbol{x}_p \in \mathbb{R}^{N\times P^2}$. The $\boldsymbol{x}_p$ is then mapped to constant dimension D by a trainable linear projection which is similar to the ViT [55]. We split the tensor into patches in a channel-wise way while ViT is along the image. Positional embeddings $\boldsymbol{E}_{pos}$, which helps to keep the architecture macro skeleton information, are added to obtain the input embeddings $\boldsymbol{x}_0$ as

$$\boldsymbol{x}_0 = \boldsymbol{x}_p\boldsymbol{E} + \boldsymbol{E}_{pos}, \tag{4}$$

where $\boldsymbol{E} \in \mathbb{R}^{P^2\times D}$ denotes the weights of the linear projection, $\boldsymbol{E}_{pos} \in \mathbb{R}^{N\times D}$ adopts the sine and cosine as in [54].

Relative ranking and latency prediction. We first design an encoder branch to compute the feature of each sampled architecture by self-attention. This feature is then used to predict the relative ranking among a batch of sampled architectures, and its individual latency. The encoder stacks $L = 6$ identical layers and each layer consists of a Multiheaded Self-Attention (MSA) block and a fully connected Feed-Forward Network (FFN) block. LayerNorm (LN) is applied to each input of the block, and the block outputs are added with the value passed by the residual connection. We obtain the extracted architecture feature $\boldsymbol{x}_\alpha \in \mathbb{R}^{N\times D}$ by applying LN to the output feature of the last encoder layer (Eq. 7).

$$\boldsymbol{x}'_l = \mathrm{MSA}(\mathrm{LN}(\boldsymbol{x}_{l-1})) + \boldsymbol{x}_{l-1}, \qquad l = 1,\cdots,L, \tag{5}$$

$$\boldsymbol{x}_l = \mathrm{FFN}(\mathrm{LN}(\boldsymbol{x}'_l)) + \boldsymbol{x}'_l, \qquad l = 1,\cdots,L, \tag{6}$$

$$\boldsymbol{x}_\alpha = \mathrm{LN}(\boldsymbol{x}_L). \tag{7}$$

Based on the obtained feature $\boldsymbol{x}_\alpha$, we first predict the relative ranking of architecture α. We employ two layers of linear projection with ReLU activation on the feature $\boldsymbol{x}_\alpha$, to predict the relative metric $\hat{y}_\alpha$ of the architecture following,

$$\hat{y}_\alpha = \max(0, \boldsymbol{x}_\alpha\boldsymbol{W}_r^1)\boldsymbol{W}_r^2, \tag{8}$$

and adopt the ranking loss [31] for training:

$$\mathcal{L}_{Rank} = \sum_{m=1}^{k-1}\sum_{n=m+1}^{k} \psi((\hat{y}_{\alpha_m} - \hat{y}_{\alpha_n}) * \mathrm{sign}(y_{\alpha_m} - y_{\alpha_n})), \tag{9}$$

where y_{α_i} is the ground-truth accuracy of architecture α_i, k is the batch size, and $\psi(\epsilon) = \log(1 + e^{-\epsilon})$ denotes the logistic loss function. With the supervised feature extraction, we expect that $\hat{y}_\alpha$ is able to prominently distinguish one architecture from others, and obtain the correct rankings while ignoring the true accuracy value. This predicted relative ranking score is then utilized to select the final candidate among the architectures in the classified top tier during the sampling procedure, which helps to save the evaluation costs as discussed in III-D.

Thanks to the decoder design that handles the information for single architecture, we can utilize this feature to predict the computation properties of the sampled architecture. Latency is a critical metric for measuring the performance of a DNN model on various mobile and edge devices. By predicting the ranking of the accuracy and the latency of the model, the proposed method achieves the goal that searching proper model which satisfies the precision and on-device running requirements. Here we regress the absolute latency value the same way as relative ranking prediction,

$$\hat{h}_\alpha = \max(0, \boldsymbol{x}_\alpha\boldsymbol{W}_l^1)\boldsymbol{W}_l^2, \tag{10}$$

and adopt MSE loss for training:

$$\mathcal{L}_{Lat} = \|\hat{h}_\alpha - h_\alpha\|_2, \tag{11}$$

where h_α is the ground-truth latency of the architecture.

Matching-based tier classification. The decoder is utilized to calculate the cross-attention score between the tier embeddings and the feature of the sampled architecture, and then to predict the probabilities of which quality tier it belongs to. The decoder stacks the same number of layers as the encoder, and each layer is inserted by one additional MSA block to perform the cross-attention operation. We first initialize five tier embeddings, $\{\boldsymbol{e}_1, \boldsymbol{e}_2, \boldsymbol{e}_3, \boldsymbol{e}_4, \boldsymbol{e}_5\}$, where $\boldsymbol{e}_i \in \mathbb{R}^{N\times D}$. Each tier embedding $\boldsymbol{e}_i$ represents the architecture information of the corresponding tier T_i and is added with the positional embeddings, yielding the decoder input $\boldsymbol{z}_i^0$. The first MSA block of the decoder layer acts on $\boldsymbol{z}_i^l$ and outputs $\boldsymbol{q}_i^l$ as the query like the way in Eq. 5. We calculate the cross-attention by matching the sampled architecture feature $\boldsymbol{x}_\alpha$ with $\boldsymbol{q}_i^l$ in the second MSA block as

$$\boldsymbol{z}_i'^l = \mathrm{MSA}(\mathrm{LN}(\boldsymbol{q}_i^{l-1}), \boldsymbol{x}_\alpha) + \boldsymbol{q}_i^{l-1}, \ l = 1,\cdots,L. \tag{12}$$

Then $\boldsymbol{z}_i'^l$ goes through the FFN sub-layer of the decoder and in the last layer it yields the output $\boldsymbol{z}_i = \mathrm{LN}(\boldsymbol{z}_i^L)$. We sum up all the five cross-attention features, and apply two layers of learnable linear transformation with ReLU in-between, as well

Algorithm 3: NARQ2T Training.

Input: Training data $\mathcal{D}_{trn}$, initial tier query $\boldsymbol{Q}_0$, initial tier statistics Φ, number of Epochs E

$e \leftarrow 0$ // initialize an epoch counter.
$f_{\text{enc}}^0, f_{\text{dec}}^0 \leftarrow$ initialize encoder and decoder.
while $e < E$ **do**
  for $iter \leftarrow 1 : t$ **do**
    $\boldsymbol{x}_{\alpha^m}, \hat{y}_{\alpha^m}, \hat{h}_{\alpha^m} = f_{\text{enc}}^{t-1}(\boldsymbol{x}_0^m)$. // predict relative ranks and latency.
    $\mathcal{L}_{Rank}(y_{\alpha^m}, \hat{y}_{\alpha^m})$, $\mathcal{L}_{Lat}(h_{\alpha^m}, \hat{h}_{\alpha^m}) \leftarrow$ compute ranking and MSE losses.
    $\boldsymbol{Q}_{\boldsymbol{L_d}} = f_{\text{dec}}^{t-1}(\boldsymbol{Q}_{t-1}, \boldsymbol{x}_{\alpha^m})$. // cross-attention between tier and arch embeddings.
    $\boldsymbol{p}_{\alpha^m} \leftarrow$ predict tier probability for arch using $\boldsymbol{Q}_{\boldsymbol{L_d}}$.
    $\mathcal{L}_{Cls}(\boldsymbol{p}_{\alpha^m}, \boldsymbol{l}_i^m) \leftarrow$ compute classification loss.
    $\mathcal{L} = \mathcal{L}_{Rank} + \lambda_1 \mathcal{L}_{Lat} + \lambda_2 \mathcal{L}_{Cls} \leftarrow$ compute total loss.
    $f_{\text{enc}}^t, f_{\text{dec}}^t \leftarrow$ update parameters with $\mathcal{L}$.
    $\boldsymbol{Q}_t \leftarrow$ update parameters of learnable tier query $\boldsymbol{Q}_{t-1}$.
    $\Phi \leftarrow$ update tier statistics.
  end
  $e \leftarrow e + 1$
end
Return f_{enc}^t, f_{dec}^t, Φ, $\boldsymbol{Q}_t$.

as the softmax function to predict the probabilities of which tier the architecture belongs to:

$$\boldsymbol{p}_\alpha = \text{softmax}(\max(0, \boldsymbol{z}\boldsymbol{W}_p^1)\boldsymbol{W}_p^2), \tag{13}$$

where $\boldsymbol{z} = \sum_{i=1}^5 \boldsymbol{z}_i$ and $\boldsymbol{p}_\alpha \in \mathbb{R}^5$. We employ cross-entropy loss, denoted as $\mathcal{L}_{Cls}$, to jointly train the relative ranking, latency prediction, and quality tier classification tasks,

$$\mathcal{L} = \mathcal{L}_{Cls} + \lambda_1 \mathcal{L}_{Lat} + \lambda_2 \mathcal{L}_{Rank}, \tag{14}$$

where λ controls the importance between different loss functions. In this way, the proposed method achieves to extract the feature of the sampled architecture, match it with the embeddings of all five tiers, and decide its quality tier.

B. Query to Tier

In the previous NAR [40], when the training of one batch of k sampled architectures is completed, the extracted architecture features $\{\boldsymbol{x}_{\alpha_1}, \boldsymbol{x}_{\alpha_2}, \cdots, \boldsymbol{x}_{\alpha_k}\}$ obtained by Eq. 7 are updated to the corresponding tier embedding $\boldsymbol{e}_i$ by calculating the mean value of the features as follows:

$$\boldsymbol{e}_i^t = \frac{\boldsymbol{e}_i^{t-1} \sum_{it=1}^{t-1} c_i^{it} + \sum \boldsymbol{x}_{\alpha_i}}{\sum_{it=1}^{t} c_i^{it}}, \tag{15}$$

where $\boldsymbol{x}_{\alpha_i} \in \{\boldsymbol{x}_\alpha \mid \text{GT}(\alpha) = T_i\}$, c_i^{it} denotes the counts of the architecture features belonging to T_i at iteration it. However, the quality of the tier embeddings is questioned because it depends on the feature-handling ability of the encoder. Even worse, since no labels are available during the search phase, the tier embeddings are updated according to the predicted classification results. This may further dampen the efficacy of the proposed method.

To address this issue, we propose to learn the tiers embedding automatically by learnable queries rather than updating them manually. Inspired by the previous work [56], [57], the five tier embeddings $\{\boldsymbol{e}_1, \boldsymbol{e}_2, \boldsymbol{e}_3, \boldsymbol{e}_4, \boldsymbol{e}_5\}$ are replaced with trainable parameters $\boldsymbol{Q}_0 \in \mathbb{R}^{5\times D}$. The tier embeddings $\boldsymbol{Q}_0$ are

Algorithm 4: Architecture Searching.

Input: Trained encoder f_{enc} and decoder f_{dec}, tier query $\boldsymbol{Q}_t$, collected tier statistics Φ, # of sampled archs k, constraints reuse limit R, # of candidate archs m, # of sampling iterations T_s, search space Ω.

$t \leftarrow 0$ // initialize an iteration counter.
while $t < T_s$ **do**
  $i \leftarrow 0$ // initialize reuse counter.
  $A = \emptyset$ // store sampled archs.
  while $\#A < k$ **do**
    // reuse constraints.
    if $i > R$ **then**
      $\hat{\pi}_{T_1}^{\text{FLOPs}}$ or $r_{T_1}^{\text{FLOPs}} \leftarrow$ selection for FLOPs.
      $\hat{\pi}_{T_1}^{\text{Params}}$ or $r_{T_1}^{\text{Params}} \leftarrow$ selection for #Params.
      $C_{\text{FLOPs}}, C_{\text{Params}} \leftarrow$ sample constraints.
    end
    $E \leftarrow$ randomly sample edges. // required for NAS-Bench-101.
    $O \leftarrow$ sample operators from $\hat{\pi}_{T_1}^{\text{Opt.}}$.
    $\alpha(O, E) \leftarrow$ build architecture.
    A.add($\alpha(O, E)$) **if** α satisfies C_{FLOPs} and C_{Params}.
    $i \leftarrow i + 1$
  end
  // inference on each α in A.
  $\boldsymbol{x}_\alpha, \hat{y}_\alpha, \hat{h}_\alpha = f_{\text{enc}}(\alpha) \leftarrow$ predict relative score and latency.
  $\boldsymbol{p} = f_{\text{dec}}(\boldsymbol{Q}_t, \boldsymbol{x}_\alpha) \leftarrow$ classify quality tier, $\boldsymbol{p} = \left[p_{\alpha\in T_1}, \cdots, p_{\alpha\in T_5}\right]^{\text{T}}$.
  $\alpha_{T_1} \leftarrow$ select archs with the highest prediction for Tier 1.
  $\alpha_{cand} \leftarrow$ select the top-m archs from α_{T_1} according to $\hat{y}_\alpha$.
  $t \leftarrow t + 1$
end
Return $\{\alpha_{cand}\}$, $\hat{y}_{\alpha_{cand}}$.

utilized to perform the self-attention and cross-attention by the same MSA block. The only difference is where the query and key come from. For self-attention, the query, key, and value are all from tier embeddings $\boldsymbol{Q}_0$ to calculate an output; for the following cross-attention calculation, it takes the output of the self-attention module as query, and the architectural features from the encoder output as key and value to perform the calculation and the results finally go through the FFN layer to obtain the final output of this decoder layer:

$$\boldsymbol{Q}_l^{(1)} = \text{MSA}(\boldsymbol{Q}_{l-1}) + \boldsymbol{Q}_{l-1}, \tag{16}$$

$$\boldsymbol{Q}_l^{(2)} = \text{MSA}(\boldsymbol{Q}_l^{(1)}, \boldsymbol{x}_\alpha) + \boldsymbol{Q}_l^{(1)}, \tag{17}$$

$$\boldsymbol{Q}_l = \text{FFN}((\boldsymbol{Q}_l^{(2)})) + \boldsymbol{Q}_l^{(2)}, \tag{18}$$

where $l = 1, \cdots, L_d$ is the number of the decoder layers, and the LN layers are omitted for simplicity. In practice, the decoder has the same number of layers $L_d = 6$ as that in the encoder. The key process is located in the calculation of the cross-attention: each architecture feature $\boldsymbol{x}_\alpha$ checks the tier embeddings which to attend, and selects the embedding to combine; while the tier embeddings learn to represent the quality tier by the supervised quality tier classification learning at the same time. In this way, each tier embedding learns a tier-related feature by updating itself through the training procedure. Besides the strong tier representation ability, we drop the manual tier embedding updating procedure, yielding end-to-end training while saving training expenses.

The NARQ2T training procedure is elaborated in Algo. 3. The NARQ2T consists of two components: an encoder and a decoder. The encoder takes as input the feature tensor of the

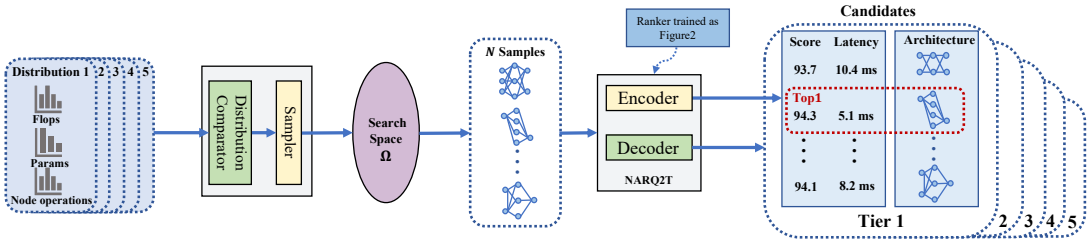


Fig. 3. The sampling procedure in the search phase. First, the distribution of the top tier is used to compare with the ones in the last ranking tiers. Then, the full statistic or only its interval is utilized to sample in the search space. The sampled architectures are ranked and scored by the trained NARQ2T. It also predicts the latency for each individual architecture. Finally, the NARQ2T selects the architecture with the highest score in tier 1.

sampled architecture and outputs its latent embedding, relative ranking, and latency prediction. We then compute the relative ranking loss and the latency regression loss. The decoder takes as input the tier query and architecture embedding, and calculates its cross-attention to classify the quality tier of the architecture. We then calculate the classification loss and sum up all three losses to update the parameters of the NARQ2T and the learnable query. At the same time, we collect the tier distributions (FLOPs, #parameters, and Operations).

C. Tier statistics collection

The distributions about FLOPs, #parameters and node operations are also collected to guide the sampling procedure at the same time. For every batch, we discretize the interval of the FLOPs and #parameters into q constraints equally where step is calculated as

$$\delta' = \frac{\tau_{\max} - \tau_{\min}}{q}, \tag{19}$$

where $\tau_{\max}$ and $\tau_{\min}$ are the maximum and minimum of the FLOPs or #parameters in one batch. We round up the step $\delta = \lceil \delta' \rceil$ and then empirically approximate the distributions $\pi(\tau)$ as follows,

$$\hat{\pi}(\tau_\kappa) = \frac{\#(\tau_\kappa = \tau_{\min} + \kappa \cdot \delta)}{k}, \ \kappa = 1, \cdots, q, \tag{20}$$

where τ_κ denotes the κ_{th} constraint on the interval, $\#(\tau_\kappa)$ denotes the number of the architecture located in constraint range $(\tau_{\kappa-1}, \tau_\kappa]$ for FLOPs or #parameters. As for every type of node operation, we just collect its total counts in each tier without discretization. Different from the offline way in AttentiveNAS [47], we train the NARQ2T and simultaneously count the distributions for different quality tiers, $\{\hat{\pi}_{T_i}(\tau_\kappa) \mid i = 1, \cdots, 5; \kappa = 1, \cdots, q\}$. In this way, we have a thorough understanding about the hierarchical distribution of the search space for guiding the sampling process.

D. Searching with tier statistics

We utilize the trained NARQ2T and the collected distributions to guide the search phase. Distributions are carefully selected first. Because, in some cases, one distribution depicting the top tier may be similar to those of low-ranking tiers. Therefore, we apply Kullback-Leibler divergence to measure the difference between top and last ranking distributions as shown in Algo. 5. Specifically, once the divergence is less than the specified threshold, we discard the possibilities of distribution but sample architectures randomly on their interval. Noted that, this is not equivalent to discarding the entire distribution since we still sample on the interval where good candidates may locate in. Besides, we sample randomly on the interval in the same way when the population of the top tier is less than a specific proportion of the sample size.

Algorithm 5: Distribution Selection.

Input: Tier distributions $\{\hat{\pi}_{T_1}, \cdots, \hat{\pi}_{T_5}\}$, batch size k and factor θ, Kullback-Leibler divergence threshold ζ, tier index β ($\beta > 1$).

```
for i ← β : 5 do
    d = KL(π̂_{T_1} ‖ π̂_{T_i}) ← compute KL divergence.
    if #(samples) in π̂_{T_1} < θ · k or d < ζ then
        Drop π̂_{T_1} possibilities but keep its interval r_{T_1}.
        Return r_{T_1}.
    end
end
Return π̂_{T_1}.
```

For every iteration, we sample k subnets from the search space. Specifically, we sample FLOPs and #parameters as constraints with the selected distributions yielded in Algo. 5 and reuse the constraints for every m ($m < k$) subnets. The implementation of the architectures sampling for NAS-Bench-101 and NAS-Bench-201 are detailed in Sec. IV-A. After sampling the architecture, we build the subnet and reject those exceeding the constraints. In addition, we randomly sample a certain proportion of subnets from the entire search space to increase the diversity.

Notice that the sampled architectures may not come from tier T_1 because each node operation is sampled independently and all these nodes are formed into the final architecture. In order to further narrow the searching phase to only the tier T_1 architectures, we first use the NARQ2T to classify the sampled architectures and then score the ones from tier T_1. Specifically, giving a batch of k samples, we leverage the trained NARQ2T to rank and score the architectures, i.e., select the top 5 architectures according to the predicted score in the classified tier T_1 as candidates at each iteration. Then, We train these candidates and obtain their evaluated accuracy. Thus, our sampling method fully exploits the knowledge of the training set to approximate the search space and achieves a top-quality aware sampling procedure. The sampling procedure in the searching phase is shown in Fig. 3.

The detailed searching algorithm is shown in Algo. 4. using

TABLE I
COMPARISON RESULTS ON THE NAS-BENCH-101 SEARCH SPACE.

Methods	Average Accuracy (%)	Best Accuracy (%)	Best Rank (‰)	Cost (seconds) train	Cost (seconds) search
DARTS [23]	92.21±0.61	93.02	13.47	-	-
ENAS [20]	91.83±0.42	92.54	22.88	-	-
FairNAS [45]	91.10±1.84	93.55	0.77	-	-
SPOS [37]	89.85±3.80	93.84	0.07	-	-
FBNet [58]	92.29±1.25	93.98	0.05	-	-
CTNAS† [32]	93.93±0.12	94.14	0.02	188.37	750.97
ReNAS† [31]	93.93±0.09	94.02	0.09	**73.68**	-
GMAE [59]	93.98±0.15	93.99	0.14	-	-
G-EA [60]	93.99±0.25	-	-	30128.32	
NAR (random) [40]	94.06±0.04	94.10	0.03	255.86	**53.51**
NAR (statistics) [40]	94.07±0.09	**94.19**	**0.01**	292.61	189.94
NARQ2T (random)	94.06±0.04	94.10	0.03	182.38	59.21
NARQ2T (statistics)	**94.11±0.04**	94.17	0.02	205.30	209.17

"Average Accuracy" denotes the averaged results of the best-searched architectures of five independent runs on the CIFAR-10 dataset. "Best Accuracy" and "Best Rank" denote the top-1 classification accuracy (%) and thousandth rank of the best searched architecture . "†" denotes our implementation.

the trained NARQ2T and the collected distributions from the previous step, we perform searching in the search space. First, after selecting the distributions according to Algo. 5, we sample FLOPs and the number of parameter constraints. Then we sample operations and edge connections to build the architecture. We reject architectures that do not satisfy the constraints and keep the ones that meet the constraints. Finally, we use trained NARQ2T to predict their relative rankings and latency and classify quality tiers. We select the ones from Tier 1 with the highest relative score as the candidates. They are then evaluated to query their ground-truth metric for measuring the performance of the proposed method.

IV. EXPERIMENTS

In this section, we conduct extensive experiments to verify the effectiveness of the proposed NARQ2T on two widely accepted NAS search spaces, namely NAS-Bench-101 [41], and NAS-Bench-201 [42]. We further verify the latency-predicting function on the previous NAR [40] and the enhanced NARQ2T. All our experiments are implemented on one NVIDIA TITAN RTX GPU. The implementation codes and the detailed cell information datasets are available[2].

A. Implementation

Datasets. We evaluate the proposed methods from two perspectives:

1) Test the validity of the NARQ2T to find the outperformed architectures from large search space.
2) Test the generalization of the NARQ2T over different image datasets. This can be validated by finding the most suitable architecture given a specific dataset from the same search space.

To this end, for the first perspective, we adopt the NAS-Bench-101 [41], which is a cell-based search space containing over 423k unique convolutional architectures. All of the architectures are trained on the CIFAR-10 3 times to obtain validation and test accuracy. For the second, we employ the NAS-Bench-201 [42]. It is also a cell-based dataset with 15625 unique convolutional architectures and corresponding training, validation, and test accuracy trained on three different image datasets, i.e., CIFAR-10, CIFAR-100, and ImageNet-16-120.

Settings. For the training details, on NAS-Bench-101, the AdamW optimizer is set with $\beta_1 = 0.9$, $\beta_2 = 0.982$, weight decay term is 5×10^{-4} and $\epsilon = 10^{-9}$. The batch size is set to 256 and two variants of NAR are both trained for 35 epochs, with 50 iterations as a warm-up. When on NAS-Bench-201, the AdamW optimizer is set with $\beta_1 = 0.9$, $\beta_2 = 0.99$, weight decay term is 1×10^{-2} and $\epsilon = 10^{-9}$. The batch size is set to 128 and the training epoch is set to 55 epochs with 30 iterations as a warm-up.

As for the sampling details, when sampling a batch of architectures as shown in Fig. 3, the sample size is the same as the training batch size, and we set the random sample rate at 0.5, in order to balance between the stable average accuracy and the superior individual selection ability, the ablation is at Section IV-D. For the FLOPS and #parameters constraints, they are reused for every 25 sampling trails and we sample a total of 50 iterations. For variables in Algo. 5, thresholds ζ of Kullback-Leibler divergence for FLOPs and #parameters are both 2.5, the batch factor $\theta = 0.1$ and tier index $\beta = 4$.

When sampling an individual architecture, for a total of 7 nodes of the cell in NAS-Bench-101, traversing from the second node (the first node is `IN` node) to the last `OUT` node, we first randomly sample a previous node of one specific node and build their connection, then we sample the operation type of the node according to the collected distributions. For the remaining 3 edges (maximum 9 edges in each cell), two nodes are randomly sampled to build the connection and we repeat the procedure 3 times. For NAS-Bench-201, since each cell has 4 nodes with fixed connections (each node connects to all of its previous nodes), we can only sample the operation type for the edges of all nodes.

B. Comparing with state-of-the-art methods

Results on NAS-Bench-101. We randomly sample 1% (4236) of the architectures with corresponding averaged accuracy as

[2]https://github.com/AlbertiPot/nar.git

TABLE II
COMPARISON RESULTS ON THE NAS-BENCH-201 SEARCH SPACE.

Methods	Cost (seconds)	CIFAR-10 validation	CIFAR-10 test	CIFAR-100 validation	CIFAR-100 test	ImageNet-16-120 validation	ImageNet-16-120 test
RSPS [61]	7587	87.60±0.61	91.05±0.66	68.27±0.72	68.26±0.96	39.73±0.34	40.69±0.36
SETN [62]	31010	90.00±0.97	92.72±0.73	69.19±1.42	69.36±1.72	39.77±0.33	39.51±0.33
ENAS [20]	13315	90.20±0.00	93.76±0.00	70.21±0.71	70.67±0.62	40.78±0.00	41.44±0.00
FairNAS [45]	9845	90.07±0.57	93.23±0.18	70.94±0.94	71.00±1.46	41.90±1.00	42.19±0.31
ReNAS [31]	86.31	90.90±0.31	93.99±0.25	71.96±0.99	72.12±0.79	45.85±0.47	45.97±0.49
GenNAS [63]	1080	-	94.18±0.10	-	72.56±0.74	-	45.59±0.54
DARTS- [64]	11520	91.03±0.44	93.80±0.40	71.36±1.51	71.53±1.51	44.87±1.46	45.12±0.82
BOHB [65]	**3.59**	91.17±0.27	93.94±0.28	72.04±0.93	72.00±0.86	45.55±0.79	45.70±0.86
GradSign [66]	-	89.84±0.61	93.31±0.47	70.22±1.32	70.33±1.28	42.07±2.78	42.42±2.81
GMAE [59]	-	-	94.03±0.11	-	72.56±0.16	-	46.09±0.27
G-EA [60]	26911	91.26±0.20	93.99±0.23	72.62±0.77	72.36±0.66	45.97±0.72	46.04±0.67
PRE-NAS [67]	-	91.37±0.28	94.04±0.34	71.95±1.21	72.02±1.22	45.16±1.00	45.34±1.03
NAR (average) [40]	168.99	91.44±0.10	94.33±0.05	72.54±0.44	72.89±0.37	46.16±0.37	46.66±0.23
NAR (best) [40]	168.99	91.61	94.37	73.49	73.51	46.50	47.31
NARQ2T (average)	162.06	**91.48±0.04**	**94.35±0.04**	**72.73±0.44**	**73.22±0.17**	**46.24±0.37**	**46.93±0.33**
NARQ2T (best)	162.06	91.55	94.37	73.49	73.51	46.50	47.31
Optimal	-	91.61	94.37	73.49	73.51	46.73	47.31

"validation" and "test" denote the top-1 classification accuracy (%) on the validation set and test set, respectively. "average" denotes the averaged top-1 classification accuracy (%) of the best architecture for 10 runs. "best" denotes the top-1 classification accuracy (%) of the best architecture among 10 runs. "Optimal" denotes the optimal accuracy for each dataset split. In the search phase, only the interval of the collected distribution is adopted to sample architectures. The reported cost of the NAR and NARQ2T includes the total cost of training and searching.

our training set and another 1024 architectures as the validation set. Two types of sampling methods are tested: 1) *random*: randomly sample one batch of architectures from the entire search space in every iteration; 2) *statistics*: randomly sample a certain proportion of architectures and the rest of the batch is sampled with the collected distributions.

The comparison results with previous state-of-the-art methods are shown in Tab. I. Both the NAR and NARQ2T outperform the previous methods on the measure of average accuracy, namely 94.07% and 94.11% for 5 independent runs. This demonstrates that our method of combining the classification of quality tiers and regulating relative ranking simultaneously is a promising way for enhancing the architecture performance predictors. Moreover, the proposed methods both perform well when searching with different sampling methods, i.e., *random* and *statistics*, showing robustness when adapting to different search pipelines. Further, the NARQ2T achieves state-of-the-art on average accuracy (94.11%) with relatively low variance. This shows that the proposed learnable Query to Tier technique has a strong ability to represent the characteristics of the quality tiers. However, upon comparing the best results of the two methods from their respective five sets of experiments, it was observed that the NAR has a slight advantage. It is assumed that this is due to the randomness of architecture sampling, which can be verified by comparing the standard deviation of the five experiments. Notably, the NAR exists the highest standard deviation (0.09) among all the experiments. As for the search cost, our sampling cost is lower compared to the CTNAS which requires training an RL during searching. When sampling with *random*, the NAR and NARQ2T achieve the lowest costs, 55.51 and 59.21 seconds respectively, since it does not require rejection sampling. As for the training cost, different from ReNAS, our pipeline encodes architectures and trains the NAR simultaneously which costs more but is closer to the real application. Thanks to the proposed Query to Tier technique, we have dropped the complex and manual tier embedding updating, which saves nearly 30% training overheads for both sampling methods.

Results on NAS-Bench-201. We randomly sample 1000 architectures with validation accuracy as our training set and another 256 architectures as the validation set. The training and searching are the same with NAS-Bench-101 except for we sample randomly on the interval of the collected distribution since the search space is small.

The comparison results are shown in Tab. II. The enhanced NARQ2T achieves new state-of-the-art average validation and test accuracy on all three datasets with relatively low variance. This further demonstrates that the Query to Tier technique is capable of representing the tier feature even in a small search space. Moreover, though we drop the collected distributions and sample randomly, the best results of the ten search trails are close to optimal. This shows our combination of classification and ranking is capable of finding the top-performing architectures in the search space. Same with NAS-Bench-101 results, the total costs show a reduction from 168.99 to 162.06 seconds due to the enhanced tier embedding updating.

Our search expenses cost larger than BOHB [65], because for the tabular benchmark which offers ground-truth performance for each architecture, the searching procedure of BOHB does not include weights training but only explores (by random sampling or Bayesian optimization) various configurations of the architectures under a selected budget. Therefore, the search costs 3.59 seconds of BOHB actually consists of architecture sampling and program processing time. On the contrary, learning-based search method, such as SETN [62], ENAS [20], DARTS [23], requires training the weights of specific components: evaluator in the SETN, controller in the ENAS, the architectural parameters for DARTS, and the performance predictor for our method. And this requires instantiating a real neural network. Thus, the reported costs of the learning-based methods include the costs of architecture instantiating, data loading, and model optimization. Without considering the evaluation of the sampled architectures when training the

TABLE III
LATENCY PREDICTION RESULTS ON THE NAS-BENCH-201 SEARCH SPACE.

Methods	Cost (seconds)	CIFAR-10 validation (%)	CIFAR-10 test (%)	CIFAR-10 RMSE (ms)	CIFAR-10 ±5%	CIFAR-10 ±10%	CIFAR-100 validation (%)	CIFAR-100 test (%)	CIFAR-100 RMSE (ms)	CIFAR-100 ±5%	CIFAR-100 ±10%	ImageNet-16-120 validation (%)	ImageNet-16-120 test (%)	ImageNet-16-120 RMSE (ms)	ImageNet-16-120 ±5%	ImageNet-16-120 ±10%
NAR [40]	168.99	91.44	94.33	-	-	-	72.54	72.89	-	-	-	46.16	46.66	-	-	-
NARLat	197.16	91.44	94.29	1.49	48%	78%	72.83	73.11	1.87	44%	64%	45.91	46.52	2.70	20%	48%
NARQ2T	162.06	91.48	94.35	-	-	-	72.73	73.22	-	-	-	46.24	46.93	-	-	-
NARQ2TLat	161.24	91.27	94.21	1.64	36%	72%	72.86	73.00	1.95	44%	62%	45.66	46.75	2.74	14%	36%
Optimal	-	91.61	94.37	-	-	-	73.49	73.51	-	-	-	46.73	47.31	-	-	-

"validation" and "test" denote the top-1 classification accuracy (%) on the validation set and test set, respectively. "Optimal" denotes the optimal accuracy for each dataset split. The latency prediction results are measured by the RMSE, ±5% and ±10% accuracy of the predicted latency from the final top-5 ranking searched architectures. The reported values of classification accuracy and latency measurements in the table are averaged of 10 independent runs. The reported cost of the NAR and NARQ2T includes the total cost of training and searching.

TABLE IV
COMPARISONS RESULTS FOR QUALITY TIERS.

#tiers	top-1 Avg. Acc. (Best Acc.) (%)	top-5 Avg. Acc. (Best Acc.) (%)
-	93.38±0.41 (93.70)	93.55±0.49 (93.85)
3	94.02±0.08 (94.10)	94.03±0.07 (94.10)
5	94.02±0.12 (94.19)	94.07±0.09 (94.19)
7	93.95±0.10 (94.11)	94.04±0.13 (94.19)

"top-k" denotes the k architectures with the highest prediction scores. The reported accuracy is averaged of five independent runs on the NAS-Bench-101 dataset. The accuracy of the best architecture among five runs is reported in the brackets.

TABLE V
ABLATION RESULTS ON DIFFERENT LOSSES.

top-k	Cls. Loss Avg. Acc. (%)	Ranking Loss Avg. Acc. (%)	Cls.+Ranking Loss Avg. Acc. (%)
1	93.81±0.08(93.91)	94.00±0.08(94.11)	94.02±0.12(94.19)
3	93.86±0.04(93.91)	94.02±0.08(94.11)	94.07±0.09(94.19)
5	94.02±0.11(94.22)	94.02±0.08(94.11)	94.07±0.09(94.19)

The reported top-1 classification accuracy is averaged of five independent runs on the NAS-Bench-101. The performance of the best architecture is reported in the brackets.

search algorithms, the overhead of these algorithms that are closer to real NAS application scenarios costs larger.

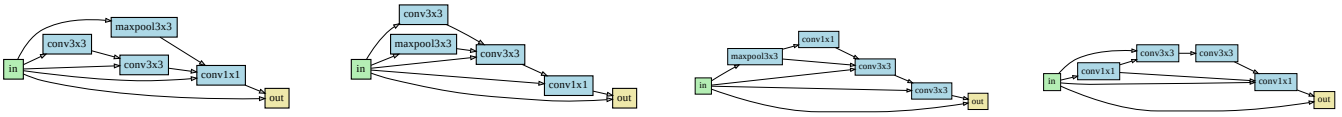
Architecture Visualization. We visualize the obtained computation cells on NAS-Bench-101 in Fig. 4, the top-1 test accuracy and ranks in the benchmark are reported in the caption of the figure. For NAS-Bench-101, the computation cells are stacked three times, followed by a downsampling layer, in which the image height and width are halved via max-pooling, and the channel count is doubled. The pattern is repeated three times and followed by global average pooling and a final dense softmax layer. The initial layer of the model is a stem consisting of one 3×3 convolution with 128 output channels. For NAS-Bench-201, the searched computation cells are shown in Fig. 5, Fig. 6 and Fig. 7. For the macro model, it is initiated with one 3×3 convolution with 16 output channels and a batch normalization layer. The searched cells are stacked 5 times with the number of output channels as 16, 32, and 64 for the first, second, and third stage. Differently, the residual block with stride 2 is adopted as the downsampling layer. The model ends up with a global average pooling layer and softmax layer. Notice that the function of the node and edge is different for the two benchmarks.

C. Latency prediction

Settings. The NAS-Bench-201 search space also provides the latency for each model on all three datasets, namely CIFAR-10, CIFAR-100, and ImageNet-16-120. The latency is obtained by running each model with batch size 256 on a single GPU (GeForce GTX 1080 Ti). We verify the latency prediction function for both NAR and NARQ2T by adding an additional prediction head based on the obtained architecture feature $\boldsymbol{x}_\alpha$. The loss factor λ_1 is set to 7 and the training and sampling details follow the previous NAS-Bench-201 experiments. We report RMSE, ±5% and ±10% accuracy of the predicted latency from the final top-5 ranking architectures. **Results.** The latency prediction results for NAR and NARQ2T are shown in Tab. III. The RMSE of the prediction results is rather low for both NAR and NARQ2T: less than 1.7 ms for CIFAR-10, less than 2 ms for CIFAR-100, and less than 2.8 ms for ImageNet-16-120, indicating an accurate latency prediction. From the ±5% and ±10% accuracy on CIFAR-10 and CIFAR-100, the latency prediction for most of the selected structures falls into a reasonable region, showing an ability to search DNN models for mobile and edge devices. Moreover, though adding an additional latency prediction task, the NAR and NARQ2T still maintain the ability to find the high accuracy architectures: for CIFAR-10 and ImageNet-16-120, the validation and test accuracy of the obtained architectures drop slightly and can be ignored; for CIFAR-100, it even obtains better candidates for NAR.

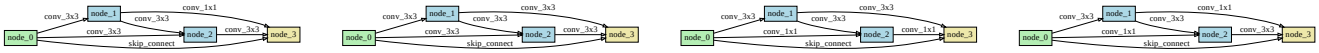
D. Ablation Study

Effect of the quality tiers partition. The quality distribution of the search space is divided into five tiers for both the NAR and NARQ2T. During the training, it collects the statistics of the FLOPs and the #parameters for each tier. These statistics will be used to sample candidates in the later search phase (Sec. III-D). Given a fixed number of training samples, if the number of tiers is set larger, the samples of each tier will be inadequate. If it is set smaller, each tier will cover multiple quality levels and could blur the useful classification information. We further visualize the tier distribution of a batch of 1024 architectures randomly sampled from the search space, and their classification results by NAR in Fig. 8. It shows that when the number of tiers is set to five, each tier has enough samples and they span uniformly in the search space, showing a clear quality distribution. We conduct ablation studies on NAS-Bench-101 with NAR to testify the



(a) Test Accuracy 94.17%, Rank 7. (b) Test Accuracy 94.11%, Rank 11. (c) Test Accuracy 94.11%, Rank 12. (d) Test Accuracy 94.07%, Rank 22.

Fig. 4. NAS-Bench-101 computation cells discovered on CIFAR-10.

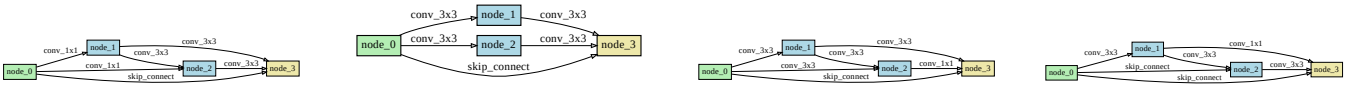


(a) Test Accuracy 94.37%, Rank 2. (b) Test Accuracy 94.36%, Rank 3. (c) Test Accuracy 94.34%, Rank 4. (d) Test Accuracy 94.30%, Rank 7.

Fig. 5. NAS-Bench-201 computation cells discovered on CIFAR-10.

(a) Test Accuracy 73.49%, Rank 1. (b) Test Accuracy 73.26%, Rank 2. (c) Test Accuracy 73.22%, Rank 3. (d) Test Accuracy 73.16%, Rank 5.

Fig. 6. NAS-Bench-201 computation cells discovered on CIFAR-100.



(a) Test Accuracy 47.31%, Rank 1. (b) Test Accuracy 46.85%, Rank 4. (c) Test Accuracy 46.71%, Rank 6. (d) Top-1 Test Accuracy 46.67%, Rank 8.

Fig. 7. NAS-Bench-201 computation cells discovered on ImageNet-16-120.

above discussion. From the results shown in Tab. IV, when the number of tiers is set to five, the results are better than others in terms of average or best performance. We have also constructed a baseline model without quality tiers. This makes our model degenerate into a pure property predictor consisting of only the encoder, and it has to sample randomly in the entire search space and selects candidate architecture according to the predicted valid accuracy. The average and best performance of this baseline are reported in the first line of Tab. IV and it shows a large performance drop. This indicates that the quality tier plays an indispensable role in assisting our method to search and select the superior architectures from the search space. The NARQ2T maintains the same partition.

Ablation on classification and ranking losses. We apply classification and ranking loss simultaneously. If we only adopt classification loss, the NAR and NARQ2T roughly classify the architectures into five tiers and choose the architectures according to the highest classification probabilities. However, the probabilities merely indicate whether a sampled architecture belongs to a given tier and does not include their ranking information inside the tiers. So, we add a ranking loss to select the one within the tier. When only adopting the ranking loss, we also consider the generalization problem that predictors may fail when they encounter architectures with large distinctions. To tackle this drawback, we adopt the classification loss to roughly pick up the candidates which is much more generalized than a pair-wised ranking loss. From the ablation results of NAR in Tab. V, we observe that the combination of classification and ranking can improve the mean performance. Moreover, when only adopting ranking loss, the NAR is not able to find the superior architectures (94.11% vs 94.19%). This is possibly because the top-performing ones are so rare that the predictor with only pair-wise ranking loss cannot generalize well to identify them. More interestingly, when adopting the classification loss only, the predictor can find one with 94.22% accuracy. This indicates that the predictor trained with classification loss can generalize well to identify those high-performed ones. We further ablate on the ranking loss factor λ_2 for NARQ2T on NAS-Bench-101 and NAS-Bench-201, the results are shown in Tab. VI.

Effect of the random samples. We conduct experiments to investigate the effect of the proportion of the randomly sampled architectures with the original NAR. As shown in Tab. VII, when sampling entirely with the distribution collected during training ($p = 0$), the NAR framework yields a deteriorated performance. However, with a certain proportion of randomly sampled architectures $p \in 0.3, 0.5, 0.7$ added in or even all ($p = 1$), the proposed method reaches a high mean accuracy. This may attribute to two points: 1) the distribution is collected in each batch during the training, and distributions from all batches will be used by turns across all sampling iterations, so the samples of each distribution might not be adequate; 2) The NAR is trained with different tiers of architectures while it can not distinguish architectures

TABLE VI
ABLATION RESULTS ON THE RANK LOSS FACTOR FOR NARQ2T.

NAS-Bench-101		NAS-Bench-201								
		CIFAR-10			CIFAR-100			ImageNet-16-120		
loss factor	Average Accuracy (%)	loss factor	validation (%)	test (%)	loss factor	validation (%)	test (%)	loss factor	validation (%)	test (%)
19	94.05±0.08	3	91.42±0.08	94.29±0.07	21	72.82±0.42	73.26±0.14	16	46.22±0.33	46.78±0.26
20	94.03±0.09	4	91.48±0.07	94.33±0.08	22	72.85±0.41	73.23±0.16	17	45.84±0.44	46.81±0.23
21	**94.11±0.04**	5	**91.48±0.04**	**94.35±0.04**	23	**72.73±0.44**	**73.22±0.17**	18	**46.24±0.37**	**46.93±0.33**
22	94.03±0.05	6	91.46±0.12	94.29±0.10	24	72.81±0.35	73.09±0.25	19	46.00±0.37	46.83±0.30
23	94.06±0.06	7	91.48±0.07	94.32±0.06	25	72.77±0.41	73.11±0.37	20	45.89±0.47	46.85±0.28

The reported top-1 classification accuracy is averaged of five runs for NAS-Bench-101, and 10 runs for NAS-Bench-201, respectively.

TABLE VII
ABLATION RESULTS ON THE PROPORTIONS (p) OF RANDOM SAMPLES .

p	Avg. Acc. (%)	Best Acc. (%)
0.0	93.87±0.09	93.91
0.3	93.99±0.10	94.17
0.5	**94.07±0.09**	**94.19**
0.7	94.06±0.05	94.14
1.0	94.06±0.04	94.10

We perform five independent runs on the NAS-Bench-101 dataset.

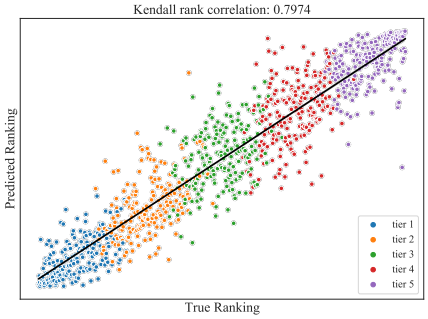


Fig. 8. Rank correlation between the predicted and the ground-truth ranking. 1024 architectures are randomly sampled from the NAS-Bench-101 dataset. Architectures are marked with their ground-truth tiers. Kendall's $\tau = 0.7974$ is obtained in one trial of five experiments.

from the same tier, so we propose to add a certain proportion of randomly sampled architectures to the batch to improve the data diversity. The NAR and NARQ2T share the same proportion of the random samples.

Effect of sampling constraints. We sample the candidate architectures from the search space with the collected distributions of FLOPs and #parameters following [47]. Since we do not apply any of the search algorithms in the search phase, the quality of the sampling plays an indispensable role in selecting the top-ranking architecture. Thus, we further ablate the properties of the sampling procedure, i.e., sample 1) only with FLOPs, 2) only with #parameters and 3) with both. As shown in Tab. VIII, sampling with both properties yields better performance. The NAR and NARQ2T share the same sampling constraint settings in our experiments.

Ablation on latency loss factor. Our encoder design which handles a single architecture feature enables us to predict the computational properties, e.g., latency, of one sampled architecture. This can be accomplished by adding a latency prediction head to the encoder and an MSE loss to the total loss in Eq. 14. We further ablate on the latency loss factor

TABLE VIII
ABLATION RESULTS ON SAMPLING CONSTRAINTS.

top-k	FLOPs Avg. Acc. (%)	#Params Avg. Acc. (%)	FLOPs&#Params Avg. Acc. (%)
1	93.93±0.11 (94.11)	93.97±0.11 (94.17)	94.02±0.12 (94.19)
3	93.99±0.11 (94.14)	94.01±0.09 (94.17)	94.07±0.09 (94.19)
5	94.00±0.11 (94.14)	94.01±0.09 (94.17)	94.07±0.09 (94.19)

The reported top-1 classification accuracy is averaged of five independent runs on the NAS-Bench-101. The performance of the best architecture is reported in the brackets.

λ_1 based on the NAR, and the results are shown in Tab. IX. When the λ_1 is set to 7, both the average test accuracy and the latency prediction accuracy of the obtained architectures are the highest, while the validation accuracy and latency prediction RMSE fall slightly behind others. The NARQ2T maintains the same factor setting in our experiments.

Ablation on training samples. Two factors determine the number of training samples: 1) having sufficient training samples can make the predictor predict more accurately; 2) having sufficient samples makes the distribution used in the sampling stage more accurate. We ablate the effect on final search results for the number of the training samples in Fig. 9. Specifically, we conducted two sets of experiments: in the first one, we only vary the training samples but still use the collected distributions, which reflected the overall impact of the training samples on our method, denoted as "statistics"; in the second one, we fix the sampling method with uniform sampling to eliminate the influence of insufficient samples, denoted as "uniform". As shown in Fig. 9, when there are few training samples (400 and 1000), the performance of the "statistics" method has a relatively large fluctuation range, and the median is also significantly lower than the method that uses a uniform sampling method. When increasing the training samples to 2000 or above, the performance distribution is fluctuate smaller and higher than that of the uniform sampling. This indicates that an adequate number of training samples can help to improve the accuracy of the collected distributions. Finally, we use 1% of the architectures offered in the NAS-Bench-101 to train the proposed method, which are 4236, and this achieves state-of-the-art results compared to previous methods that use the same amount of training samples or more [31], [32], [59].

Ablation on architecture encoding attributes. We further ablate on the attributes of the encoding method to investigate the inherent relationship between them in Fig. 10. First, we observe that node FLOPs information is critical for the

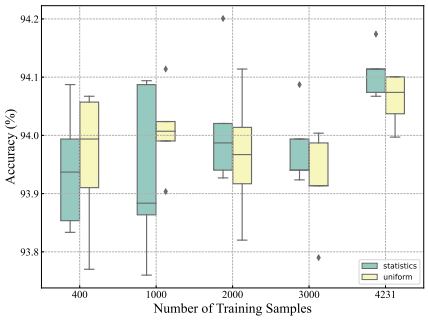


Fig. 9. The effect of the number of training samples. "statistics" denotes sampling architecture with the distributions collected in the training phase; "uniform" denotes sampling architecture uniformly from the interval of the collected distributions. The number of training samples is set to 1% of the architectures in NAS-Bench-101, for both sampling settings. The results are obtained from five independent runs on the NAS-Bench-101.

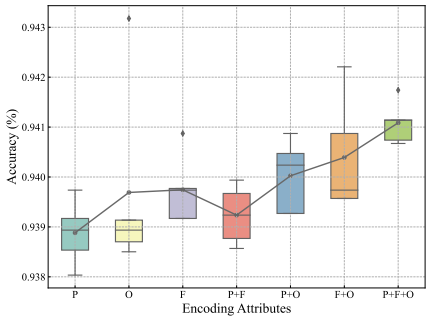


Fig. 10. The effect of the attributes used to encode the architecture. "P" denotes Parameters; "F" denotes FLOPs; "O" denotes the type of Operation; The architecture representation encompassing all three attributes yields the highest level of performance. The results are obtained from five independent runs on the NAS-Bench-101.

TABLE IX
ABLATION RESULTS ON THE LATENCY LOSS FACTOR.

loss factor	Val. Acc. (%)	Test Acc. (%)	RMSE (ms)	±5%	±10%
1	91.29±0.25	94.17±0.19	1.80±0.51	40%	64%
2	91.40±0.22	94.21±0.18	1.63±0.57	44%	72%
3	91.29±0.26	94.13±0.21	1.78±0.37	36%	56%
4	91.27±0.18	94.17±0.19	1.65±0.52	44%	64%
5	91.44±0.23	94.29±0.19	1.66±0.57	44%	64%
6	91.35±0.21	94.25±0.14	1.62±0.63	36%	72%
7	91.44±0.09	**94.31±0.07**	1.40±0.47	**52%**	**84%**
8	**91.45±0.07**	94.29±0.07	1.48±0.58	40%	76%
9	91.39±0.15	94.22±0.20	**1.38±0.28**	**52%**	**84%**
10	91.45±0.11	94.26±0.10	1.54±0.59	44%	72%

We perform five independent runs on the NAS-Bench-101 and report the average classification accuracy and latency prediction measurements.

TABLE X
COMPARISONS OF THE BUILDING BLOCK.

top-k	Avg. Acc. (%)	Best Acc. (%)	Building block
1	93.70±0.27	93.99	ConvNet
3	93.83±0.19	93.99	
5	93.89±0.16	94.04	
1	94.02±0.12	94.19	MSA
3	94.07±0.09	94.19	
5	94.07±0.09	94.19	

"top-k" denotes the k architectures with the highest prediction scores. The reported top-1 classification accuracy is averaged of five independent runs on the NAS-Bench-101.

final performance, since 1) the average performance of the method that only encodes FLOPs is slightly higher than the method that only encodes operator types, and much higher than the way of only encoding the number of the parameters; 2) when encoding together with operator types, the method that includes FLOPs information again achieves higher performance. Second, encoding with operation type is important since when encoding the number of parameters or FLOPs together with the operations separately, the performance is higher than combining FLOPs with parameters and the single-attribute encoding method. Finally, when combing all three attributes, the performance achieves the highest, indicating the representation ability of our encoding method that includes not only the topology and note operation types but also detailed information for each node inside each cell located in different depths of the architecture. To further promote the architecture representation study, we calculate the FLOPs and the number of the parameters of each node (operation) for each architecture in the NAS-Bench-101 and NAS-Bench-201 and release this dataset. This could further help the community to look in-depth at the powerful representation including detailed node information, which further boosts the NAS research.

Effect of the building block Different from previous methods which adopt convolution network [31] as the predictor, we adopt the MSA block for the NAR and NARQ2T because we want the predictor handles patch data, which is exactly our method for architecture encoding as mentioned in Section III-A. Specifically, the neural architecture is encoded into patches. MSA figures out which cell is important in the architecture and how it affects other cells by calculating the multi-attention score along the direction of the stacked patches. However, when using the convolution network, it operates on sub-regions inside the patches. This is equivalent to obtaining the relationship between different operations among different cells, which do not have direct connections in real architecture. Besides, we add positional embeddings to each patch to preserve the architecture depth information while the convolution network will make it invalid. To showcase how well the proposed method adapts to architecture encoding, we compare the NAR with a 3-layer convolutional network trained with the ranking loss [31] to predict the ranking of the architectures and then sample candidates directly from the entire search space. The results in Tab. X show that MSA is the proper building block for the proposed method. The NARQ2T maintain the same building block.

Evaluate cost during sampling. During the search phase, we sample for 50 iterations and select the top five architectures to query their test accuracy in every iteration, which is a total of 250 architectures to evaluate. It should be noted that querying the metric for candidates from the tabular NAS benchmark, e.g., NAS-Bench-101 or NAS-Bench-201, is equal to training and evaluating a true neural network in real-world scenarios. In other words, the fewer architectures selected, the more evaluate cost we save. So, one should better to save the number of queries to save the evaluation costs. This requires the NARQ2T to hold a high ranking and classification ability to ensure that we can find the outperforming candidates under limited

TABLE XI
COMPARISONS OF DIFFERENT NUMBERS OF THE SELECTED ARCHITECTURES DURING THE SEARCH PHASE.

top-k	Avg. Acc. (%)	Queries	Ranking loss
1	94.02±0.12	50	
3	94.07±0.09	150	
5	94.07±0.09	250	✓
7	94.07±0.09	350	
10	94.07±0.09	500	
1	93.81±0.08	50	
3	93.86±0.04	150	✗
5	94.02±0.11	250	

"top-k" denotes the k architectures with the highest prediction scores. "Queries" denotes the total number of queries to the ground-truth test accuracy for 50 iterations. The reported top-1 classification accuracy is averaged of five independent runs on the NAS-Bench-101.

trials. We compare the results of the different numbers of the selected architectures in Tab. XI with basic NAR. It shows that it can still achieve higher performance even with only one architecture selected, which dramatically reduces the search cost. We further investigate the results of the NAR trained without ranking loss, the average accuracy degrades more when one architecture is selected. This demonstrates that the ranking loss is essential to improve the ranking ability.

V. CONCLUSION

In this work, we propose the Neural Architecture Ranker with Query to Tier technique to rank and score the accuracy and predict the latency of architectures for improving the search efficiency and quality of NAS. The NARQ2T framework classifies the architectures into five different quality tiers and scores them with the relative metric. The tier distributions are collected to guide the sampling during the search phase which is free of excessive search costs. The advanced tier representation ability of the learnable query makes the proposed method outperform previous NAS competitors on both the NAS-Bench-101 and NAS-Bench-201 datasets, The NARQ2T stably finds superior architectures from the search space with competitive costs. We further release two detailed cell information datasets to boost in-depth research into the micro structure in the NAS field.

REFERENCES

[1] K. Simonyan and A. Zisserman, "Very deep convolutional networks for large-scale image recognition," in *3rd International Conference on Learning Representations, ICLR 2015, San Diego, CA, USA, May 7-9, 2015, Conference Track Proceedings*, Y. Bengio and Y. LeCun, Eds., 2015. [Online]. Available: http://arxiv.org/abs/1409.1556

[2] K. He, X. Zhang, S. Ren, and J. Sun, "Deep residual learning for image recognition," in *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, June 2016.

[3] S. Ren, K. He, R. Girshick, and J. Sun, "Faster r-cnn: Towards real-time object detection with region proposal networks," in *Advances in Neural Information Processing Systems*, C. Cortes, N. Lawrence, D. Lee, M. Sugiyama, and R. Garnett, Eds., vol. 28. Curran Associates, Inc., 2015. [Online]. Available: https://proceedings.neurips.cc/paper/2015/file/14bfa6bb14875e45bba028a21ed38046-Paper.pdf

[4] J. Redmon, S. Divvala, R. Girshick, and A. Farhadi, "You only look once: Unified, real-time object detection," in *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, June 2016.

[5] J. Han, D. Zhang, X. Hu, L. Guo, J. Ren, and F. Wu, "Background prior-based salient object detection via deep reconstruction residual," *IEEE Transactions on Circuits and Systems for Video Technology*, vol. 25, no. 8, pp. 1309–1321, 2015.

[6] J. Long, E. Shelhamer, and T. Darrell, "Fully convolutional networks for semantic segmentation," in *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, June 2015.

[7] L. Chen, G. Papandreou, I. Kokkinos, K. Murphy, and A. L. Yuille, "Semantic image segmentation with deep convolutional nets and fully connected crfs," in *3rd International Conference on Learning Representations, ICLR 2015, San Diego, CA, USA, May 7-9, 2015, Conference Track Proceedings*, Y. Bengio and Y. LeCun, Eds., 2015. [Online]. Available: http://arxiv.org/abs/1412.7062

[8] Y. Li, D. Liu, H. Li, L. Li, Z. Li, and F. Wu, "Learning a convolutional neural network for image compact-resolution," *IEEE Transactions on Image Processing*, vol. 28, no. 3, pp. 1092–1107, 2019.

[9] Z. Chen, T. He, X. Jin, and F. Wu, "Learning for video compression," *IEEE Transactions on Circuits and Systems for Video Technology*, vol. 30, no. 2, pp. 566–576, 2020.

[10] J. Lin, D. Liu, H. Li, and F. Wu, "M-lvc: Multiple frames prediction for learned video compression," in *2020 IEEE/CVF Conference on Computer Vision and Pattern Recognition (CVPR)*, 2020, pp. 3543–3551.

[11] Y. Wang, Z. Chen, F. Wu, and G. Wang, "Person re-identification with cascaded pairwise convolutions," in *2018 IEEE/CVF Conference on Computer Vision and Pattern Recognition*, 2018, pp. 1470–1478.

[12] C. Lin, J. Lu, and J. Zhou, "Multi-grained deep feature learning for robust pedestrian detection," *IEEE Transactions on Circuits and Systems for Video Technology*, vol. 29, no. 12, pp. 3608–3621, 2019.

[13] S. He, K. Shi, C. Liu, B. Guo, J. Chen, and Z. Shi, "Collaborative sensing in internet of things: A comprehensive survey," *IEEE Communications Surveys & Tutorials*, vol. 24, no. 3, pp. 1435–1474, 2022.

[14] I. Hasan, S. Liao, J. Li, S. U. Akram, and L. Shao, "Generalizable pedestrian detection: The elephant in the room," in *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition (CVPR)*, June 2021, pp. 11 328–11 337.

[15] Y. Zhang, J. Lu, and J. Zhou, "Objects are different: Flexible monocular 3d object detection," in *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition (CVPR)*, June 2021, pp. 3289–3298.

[16] X. Deng, B. Wang, W. Liu, and L. T. Yang, "Sensor scheduling for multi-modal confident information coverage in sensor networks," *IEEE Transactions on Parallel and Distributed Systems*, vol. 26, no. 3, pp. 902–913, 2015.

[17] D. Liu, H. Zhang, Z.-J. Zha, and F. Wu, "Learning to assemble neural module tree networks for visual grounding," in *2019 IEEE/CVF International Conference on Computer Vision (ICCV)*, 2019, pp. 4672–4681.

[18] X. Lou, S. Guo, J. Li, and T. Zhang, "Ownership verification of dnn architectures via hardware cache side channels," *IEEE Transactions on Circuits and Systems for Video Technology*, vol. 32, no. 11, pp. 8078–8093, 2022.

[19] B. Zoph and Q. V. Le, "Neural architecture search with reinforcement learning," in *5th International Conference on Learning Representations, ICLR 2017, Toulon, France, April 24-26, 2017, Conference Track Proceedings*, 2017. [Online]. Available: https://openreview.net/forum?id=r1Ue8Hcxg

[20] H. Pham, M. Guan, B. Zoph, Q. Le, and J. Dean, "Efficient neural architecture search via parameters sharing," in *Proceedings of the 35th International Conference on Machine Learning*, ser. Proceedings of Machine Learning Research, J. Dy and A. Krause, Eds., vol. 80. PMLR, 10–15 Jul 2018, pp. 4095–4104. [Online]. Available: https://proceedings.mlr.press/v80/pham18a.html

[21] H. Liu, K. Simonyan, O. Vinyals, C. Fernando, and K. Kavukcuoglu, "Hierarchical representations for efficient architecture search," in *International Conference on Learning Representations*, 2018. [Online]. Available: https://openreview.net/forum?id=BJQRKzbA-

[22] E. Real, A. Aggarwal, Y. Huang, and Q. V. Le, "Regularized evolution for image classifier architecture search," *Proceedings of the AAAI Conference on Artificial Intelligence*, vol. 33, no. 01, pp. 4780–4789, Jul. 2019. [Online]. Available: https://ojs.aaai.org/index.php/AAAI/article/view/4405

[23] H. Liu, K. Simonyan, and Y. Yang, "DARTS: Differentiable architecture search," in *International Conference on Learning Representations*, 2019. [Online]. Available: https://openreview.net/forum?id=S1eYHoC5FX

[24] H. Cai, L. Zhu, and S. Han, "ProxylessNAS: Direct neural architecture search on target task and hardware," in *International*

Conference on Learning Representations, 2019. [Online]. Available: https://openreview.net/forum?id=HylVB3AqYm

[25] L. Zhang, S. Wang, X. Chang, J. Liu, Z. Ge, and Q. Zheng, "Auto-fsl: Searching the attribute consistent network for few-shot learning," *IEEE Transactions on Circuits and Systems for Video Technology*, vol. 32, no. 3, pp. 1213–1223, 2022.

[26] L. Cai, Y. Fu, W. Huo, Y. Xiang, T. Zhu, Y. Zhang, H. Zeng, and D. Zeng, "Multi-scale attentive image de-raining networks via neural architecture search," *IEEE Transactions on Circuits and Systems for Video Technology*, pp. 1–1, 2022.

[27] H. Huang, L. Shen, C. He, W. Dong, and W. Liu, "Differentiable neural architecture search for extremely lightweight image super-resolution," *IEEE Transactions on Circuits and Systems for Video Technology*, pp. 1–1, 2022.

[28] R. Luo, F. Tian, T. Qin, E. Chen, and T.-Y. Liu, "Neural architecture optimization," in *Advances in Neural Information Processing Systems*, S. Bengio, H. Wallach, H. Larochelle, K. Grauman, N. Cesa-Bianchi, and R. Garnett, Eds., vol. 31. Curran Associates, Inc., 2018. [Online]. Available: https://proceedings.neurips.cc/paper/2018/file/933670f1ac8ba969f32989c312faba75-Paper.pdf

[29] H. Cai, C. Gan, T. Wang, Z. Zhang, and S. Han, "Once-for-all: Train one network and specialize it for efficient deployment," in *International Conference on Learning Representations*, 2020. [Online]. Available: https://openreview.net/forum?id=HylxE1HKwS

[30] 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*, A. Vedaldi, H. Bischof, T. Brox, and J.-M. Frahm, Eds. Cham: Springer International Publishing, 2020, pp. 660–676.

[31] Y. Xu, Y. Wang, K. Han, Y. Tang, S. Jui, C. Xu, and C. Xu, "Renas: Relativistic evaluation of neural architecture search," in *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition (CVPR)*, June 2021, pp. 4411–4420.

[32] Y. Chen, Y. Guo, Q. Chen, M. Li, W. Zeng, Y. Wang, and M. Tan, "Contrastive neural architecture search with neural architecture comparators," in *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition (CVPR)*, June 2021, pp. 9502–9511.

[33] J. Fürnkranz, E. Hüllermeier, E. Loza Mencía, and K. Brinker, "Multilabel classification via calibrated label ranking," *Machine learning*, vol. 73, no. 2, pp. 133–153, 2008.

[34] M. Ji, J. Han, and M. Danilevsky, "Ranking-based classification of heterogeneous information networks," in *Proceedings of the 17th ACM SIGKDD International Conference on Knowledge Discovery and Data Mining*, ser. KDD '11. New York, NY, USA: Association for Computing Machinery, 2011, p. 1298–1306. [Online]. Available: https://doi.org/10.1145/2020408.2020603

[35] L. L. Zhang, S. Han, J. Wei, N. Zheng, T. Cao, Y. Yang, and Y. Liu, "Nn-meter: Towards accurate latency prediction of deep-learning model inference on diverse edge devices," in *Proceedings of the 19th Annual International Conference on Mobile Systems, Applications, and Services*, ser. MobiSys '21. New York, NY, USA: Association for Computing Machinery, 2021, pp. 81–93. [Online]. Available: https://doi.org/10.1145/3458864.3467882

[36] M. Tan, B. Chen, R. Pang, V. Vasudevan, M. Sandler, A. Howard, and Q. V. Le, "Mnasnet: Platform-aware neural architecture search for mobile," in *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition (CVPR)*, June 2019.

[37] Z. Guo, X. Zhang, H. Mu, W. Heng, Z. Liu, Y. Wei, and J. Sun, "Single path one-shot neural architecture search with uniform sampling," in *Computer Vision – ECCV 2020*, A. Vedaldi, H. Bischof, T. Brox, and J.-M. Frahm, Eds. Cham: Springer International Publishing, 2020, pp. 544–560.

[38] Z. Lu, G. Sreekumar, E. Goodman, W. Banzhaf, K. Deb, and V. N. Boddeti, "Neural architecture transfer," *IEEE Transactions on Pattern Analysis and Machine Intelligence*, vol. 43, no. 9, pp. 2971–2989, 2021.

[39] H. Wu and J. Zhou, "Iid-net: Image inpainting detection network via neural architecture search and attention," *IEEE Transactions on Circuits and Systems for Video Technology*, vol. 32, no. 3, pp. 1172–1185, 2022.

[40] B. Guo, T. Chen, S. He, H. Liu, L. Xu, P. Ye, and J. Chen, "Generalized global ranking-aware neural architecture ranker for efficient image classifier search," in *Proceedings of the 30th ACM International Conference on Multimedia*, ser. MM '22. New York, NY, USA: Association for Computing Machinery, 2022, p. 3730–3741. [Online]. Available: https://doi.org/10.1145/3503161.3548149

[41] C. Ying, A. Klein, E. Christiansen, E. Real, K. Murphy, and F. Hutter, "NAS-bench-101: Towards reproducible neural architecture search," in *Proceedings of the 36th International Conference on Machine Learning*, ser. Proceedings of Machine Learning Research, K. Chaudhuri and R. Salakhutdinov, Eds., vol. 97. PMLR, 09–15 Jun 2019, pp. 7105–7114. [Online]. Available: https://proceedings.mlr.press/v97/ying19a.html

[42] X. Dong, L. Liu, K. Musial, and B. Gabrys, "Nats-bench: Benchmarking nas algorithms for architecture topology and size," *IEEE Transactions on Pattern Analysis and Machine Intelligence*, vol. 44, no. 7, pp. 3634–3646, 2022.

[43] A. Howard, M. Sandler, G. Chu, L.-C. Chen, B. Chen, M. Tan, W. Wang, Y. Zhu, R. Pang, V. Vasudevan, Q. V. Le, and H. Adam, "Searching for mobilenetv3," in *Proceedings of the IEEE/CVF International Conference on Computer Vision (ICCV)*, October 2019.

[44] E. Real, S. Moore, A. Selle, S. Saxena, Y. L. Suematsu, J. Tan, Q. V. Le, and A. Kurakin, "Large-scale evolution of image classifiers," in *Proceedings of the 34th International Conference on Machine Learning*, ser. Proceedings of Machine Learning Research, D. Precup and Y. W. Teh, Eds., vol. 70. PMLR, 06–11 Aug 2017, pp. 2902–2911. [Online]. Available: https://proceedings.mlr.press/v70/real17a.html

[45] X. Chu, B. Zhang, and R. Xu, "Fairnas: Rethinking evaluation fairness of weight sharing neural architecture search," in *Proceedings of the IEEE/CVF International Conference on Computer Vision (ICCV)*, October 2021, pp. 12 239–12 248.

[46] J. Yu, P. Jin, H. Liu, G. Bender, P.-J. Kindermans, M. Tan, T. Huang, X. Song, R. Pang, and Q. Le, "Bignas: Scaling up neural architecture search with big single-stage models," in *Computer Vision – ECCV 2020*, A. Vedaldi, H. Bischof, T. Brox, and J.-M. Frahm, Eds. Cham: Springer International Publishing, 2020, pp. 702–717.

[47] D. Wang, M. Li, C. Gong, and V. Chandra, "Attentivenas: Improving neural architecture search via attentive sampling," in *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition (CVPR)*, June 2021, pp. 6418–6427.

[48] C. Liu, B. Zoph, M. Neumann, J. Shlens, W. Hua, L.-J. Li, L. Fei-Fei, A. Yuille, J. Huang, and K. Murphy, "Progressive neural architecture search," in *Proceedings of the European Conference on Computer Vision (ECCV)*, September 2018.

[49] X. Dai, A. Wan, P. Zhang, B. Wu, Z. He, Z. Wei, K. Chen, Y. Tian, M. Yu, P. Vajda, and J. E. Gonzalez, "Fbnetv3: Joint architecture-recipe search using predictor pretraining," in *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition (CVPR)*, June 2021, pp. 16 276–16 285.

[50] C. Zhang, M. Ren, and R. Urtasun, "Graph hypernetworks for neural architecture search," in *International Conference on Learning Representations*, 2019. [Online]. Available: https://openreview.net/forum?id=rkgW0oA9FX

[51] L. Dudziak, T. Chau, M. Abdelfattah, R. Lee, H. Kim, and N. Lane, "Brp-nas: Prediction-based nas using gcns," in *Advances in Neural Information Processing Systems*, H. Larochelle, M. Ranzato, R. Hadsell, M. F. Balcan, and H. Lin, Eds., vol. 33. Curran Associates, Inc., 2020, pp. 10 480–10 490. [Online]. Available: https://proceedings.neurips.cc/paper/2020/file/768e78024aa8fdb9b8fe87be86f64745-Paper.pdf

[52] K. Yu, R. Ranftl, and M. Salzmann, "Landmark regularization: Ranking guided super-net training in neural architecture search," in *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition (CVPR)*, June 2021, pp. 13 723–13 732.

[53] H. Peng, H. Du, H. Yu, Q. LI, J. Liao, and J. Fu, "Cream of the crop: Distilling prioritized paths for one-shot neural architecture search," in *Advances in Neural Information Processing Systems*, H. Larochelle, M. Ranzato, R. Hadsell, M. Balcan, and H. Lin, Eds., vol. 33. Curran Associates, Inc., 2020, pp. 17 955–17 964. [Online]. Available: https://proceedings.neurips.cc/paper/2020/file/d072677d210ac4c03ba046120f0802ec-Paper.pdf

[54] A. Vaswani, N. Shazeer, N. Parmar, J. Uszkoreit, L. Jones, A. N. Gomez, L. u. Kaiser, and I. Polosukhin, "Attention is all you need," in *Advances in Neural Information Processing Systems*, I. Guyon, U. V. Luxburg, S. Bengio, H. Wallach, R. Fergus, S. Vishwanathan, and R. Garnett, Eds., vol. 30. Curran Associates, Inc., 2017. [Online]. Available: https://proceedings.neurips.cc/paper/2017/file/3f5ee243547dee91fbd053c1c4a845aa-Paper.pdf

[55] A. Dosovitskiy, L. Beyer, A. Kolesnikov, D. Weissenborn, X. Zhai, T. Unterthiner, M. Dehghani, M. Minderer, G. Heigold, S. Gelly, J. Uszkoreit, and N. Houlsby, "An image is worth 16x16 words: Transformers for image recognition at scale," in *International Conference on Learning Representations*, 2021. [Online]. Available: https://openreview.net/forum?id=YicbFdNTTy

[56] N. Carion, F. Massa, G. Synnaeve, N. Usunier, A. Kirillov, and S. Zagoruyko, "End-to-end object detection with transformers," in *ECCV*, 2020.

[57] S. Liu, L. Zhang, X. Yang, H. Su, and J. Zhu, "Query2label: A simple transformer way to multi-label classification," *arXiv preprint arXiv:2107.10834*, 2021.

[58] B. Wu, X. Dai, P. Zhang, Y. Wang, F. Sun, Y. Wu, Y. Tian, P. Vajda, Y. Jia, and K. Keutzer, "Fbnet: Hardware-aware efficient convnet design via differentiable neural architecture search," in *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition (CVPR)*, June 2019.

[59] K. Jing, J. Xu, and P. Li, "Graph masked autoencoder enhanced predictor for neural architecture search," in *Proceedings of the Thirty-First International Joint Conference on Artificial Intelligence, IJCAI-22*, 2022, pp. 3114–3120.

[60] V. Lopes, M. Santos, B. Degardin, and L. A. Alexandre, "Efficient guided evolution for neural architecture search," in *Proceedings of the Genetic and Evolutionary Computation Conference Companion*, ser. GECCO '22. New York, NY, USA: Association for Computing Machinery, 2022, p. 655–658. [Online]. Available: https://doi.org/10.1145/3520304.3528936

[61] L. Li and A. Talwalkar, "Random search and reproducibility for neural architecture search," in *Proceedings of The 35th Uncertainty in Artificial Intelligence Conference*, ser. Proceedings of Machine Learning Research, R. P. Adams and V. Gogate, Eds., vol. 115. PMLR, 22–25 Jul 2020, pp. 367–377. [Online]. Available: https://proceedings.mlr.press/v115/li20c.html

[62] X. Dong and Y. Yang, "One-shot neural architecture search via self-evaluated template network," in *Proceedings of the IEEE/CVF International Conference on Computer Vision (ICCV)*, October 2019.

[63] Y. Li, C. Hao, P. Li, J. Xiong, and D. Chen, "Generic neural architecture search via regression," *CoRR*, vol. abs/2108.01899, 2021. [Online]. Available: https://arxiv.org/abs/2108.01899

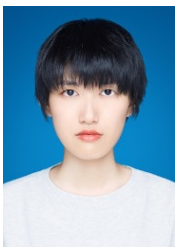
[64] X. Chu, X. Wang, B. Zhang, S. Lu, X. Wei, and J. Yan, "{DARTS}-: Robustly stepping out of performance collapse without indicators," in *International Conference on Learning Representations*, 2021. [Online]. Available: https://openreview.net/forum?id=KLH36ELmwIB

[65] S. Falkner, A. Klein, and F. Hutter, "BOHB: Robust and efficient hyperparameter optimization at scale," in *Proceedings of the 35th International Conference on Machine Learning*, ser. Proceedings of Machine Learning Research, J. Dy and A. Krause, Eds., vol. 80. PMLR, 10–15 Jul 2018, pp. 1437–1446. [Online]. Available: https://proceedings.mlr.press/v80/falkner18a.html

[66] Z. Zhang and Z. Jia, "Gradsign: Model performance inference with theoretical insights," in *International Conference on Learning Representations*, 2022. [Online]. Available: https://openreview.net/forum?id=HObMhrCeAAF

[67] Y. Peng, A. Song, V. Ciesielski, H. M. Fayek, and X. Chang, "Pre-nas: Predictor-assisted evolutionary neural architecture search," in *Proceedings of the Genetic and Evolutionary Computation Conference*, ser. GECCO '22. New York, NY, USA: Association for Computing Machinery, 2022, p. 1066–1074. [Online]. Available: https://doi.org/10.1145/3512290.3528727

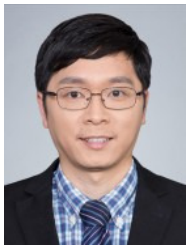
Bicheng Guo (Student Member, IEEE) received his B.S. degree from Chongqing University in 2017; and his M.S. degree from Wuhan University in 2020. He is currently working toward a Ph.D. degree with the College of Control Science and Engineering, Zhejiang University, Hangzhou. His research interests include pattern recognition, neural architecture search, and autonomous driving.



Lilin Xu (Student Member, IEEE) received her B.S. degree in Control Science and Engineering from Zhejiang University, Hangzhou, in 2021, where she is currently pursuing her master's degree at the College of Control Science and Engineering. Her research interests include computer vision, AIoT, and mobile sensing.

Tao Chen (Senior Member, IEEE) received his Ph.D. degree in information engineering from Nanyang Technological University, Singapore, in 2013. He was a Research Scientist at the Institute for Infocomm Research, A*STAR, Singapore, from 2013 to 2017, and a Senior Scientist at the Huawei Singapore Research Center from 2017 to 2018. He is currently a Professor with the School of Information Science and Technology, Fudan University, Shanghai, China. His main research interests include computer vision and machine learning. He has published more than 70 papers in reputable journals and conferences such as IEEE TPAMI/TIP/CVPR/IJCV, and granted 5 patents.

Peng Ye is currently working toward his Ph.D. degree at Fudan University, Shanghai. His research interests include computer vision, network design, and optimization. He has published papers in leading journals and conferences including IJCV, CVPR Oral, NeurIPS, ACM MM, ICME Best Student Paper, TGRS, TCSVT, ICASSP, etc. He serves as a reviewer in kinds of journals and conferences including IJCV, TCSVT, CVPR, ECCV, ICCV, etc.



Shibo He (Senior Member, IEEE) received his Ph.D. degree in control science and engineering from Zhejiang University, Hangzhou, in 2012. He is currently a Professor with Zhejiang University. He was an Associate Research Scientist from March 2014 to May 2014, and a Postdoctoral Scholar from May 2012 to February 2014, with Arizona State University, Tempe. From November 2010 to November 2011, he was a Visiting Scholar with the University of Waterloo, Waterloo. His research interests include the IoT, crowdsensing, big data analysis, etc.

Haoyu Liu received the Ph.D. degree from Zhejiang University, Hangzhou, China, in 2021. He is currently working with NetEase Fuxi AI Lab, Hangzhou. His research interests include data mining and machine learning, with particular interests in anomaly detection and crowdsourcing.

Jiming Chen (Fellow, IEEE) received the Ph.D. degree in control science and engineering from Zhejiang University, Hangzhou, China, in 2005. He is currently a Professor with the Department of Control Science and Engineering, deputy director of the State Key Laboratory of Industrial Control Technology, director of Institute of Industrial Process Control, Zhejiang University. His research interests include Internet of Things (IoT), networked control, and wireless networks.

Dr. Chen is a fellow of the CAA. He was a recipient of the 7th IEEE ComSoc Asia/Pacific Outstanding Paper Award, the JSPS Invitation Fellowship, and the IEEE ComSoc AP Outstanding Young Researcher Award. He serves as the general Co-Chairs for the IEEE RTCSA'19, the IEEE Datacom'19, and the IEEE PST'20. He is an IEEE VTS Distinguished Lecturer. He serves on the editorial boards of multiple IEEE TRANSACTIONS.