# NAR-Former: Neural Architecture Representation Learning towards Holistic Attributes Prediction

Yun Yi[1]    Haokui Zhang[2,3,*]    Wenze Hu[2]    Nannan Wang[1,*]    Xiaoyu Wang[2]

[1]Xidian University    [2]Intellifusion    [3]Harbin Institute of Technology, Shenzhen

yuny220@163.com    hkzhang1991@mail.nwpu.edu.cn

windsor.hwu@gmail.com    nnwang@xidian.edu.cn    fanghuaxue@gmail.com

## Abstract

*With the wide and deep adoption of deep learning models in real applications, there is an increasing need to model and learn the representations of the neural networks themselves. These models can be used to estimate attributes of different neural network architectures such as the accuracy and latency, without running the actual training or inference tasks. In this paper, we propose a neural architecture representation model that can be used to estimate these attributes holistically. Specifically, we first propose a simple and effective tokenizer to encode both the operation and topology information of a neural network into a single sequence. Then, we design a multi-stage fusion transformer to build a compact vector representation from the converted sequence. For efficient model training, we further propose an information flow consistency augmentation and correspondingly design an architecture consistency loss, which brings more benefits with less augmentation samples compared with previous random augmentation strategies. Experiment results on NAS-Bench-101, NAS-Bench-201, DARTS search space and NNLQP show that our proposed framework can be used to predict the aforementioned latency and accuracy attributes of both cell architectures and whole deep neural networks, and achieves promising performance. Code is available at* https://github.com/yuny220/NAR-Former.

## 1. Introduction

As an ever increasing variety of deep neural network models are widely adopted in academic research and real applications, neural architecture representation is emerging as an universal need to predict model attributes holistically. For example, modern neural architecture search (NAS) methods can depend on the neural architecture rep-

resentation to build good model accuracy predictors [5, 23, 24, 26, 31, 34], which estimate model accuracies without running the expensive training procedure. In order to find faster execution graphs when deploying models to neural network accelerators, neural network compilers [3, 20] need it to build network latency predictors, which estimate the real time cost without running it on the corresponding real hardware. As a straightforward approach to solve these holistic prediction tasks, a neural architecture representation model should be built to take the symbolic description of the network as input, and generate its numerical representation which can be easily used with existing modeling tools to perform the desired downstream tasks.

There are some neural architecture representation models proposed in solving the individual tasks mentioned above, but we find they are rather application specific and have obvious drawbacks when used in new tasks. For example, the early MLP, LSTM based accuracy prediction approaches [6, 18, 24, 34] improve the efficiency and performance of NAS, while there is a limitation of prediction performance resulting from the the nonexistent or implicit topology encoding. Some later proposed GCN based methods [4, 17, 36] achieve better performance of accuracy prediction than the above methods. However, due to the use of adjacency matrix, the encoding dimension of these methods scales quadratically with the depth of the input architecture, making them difficult to model large networks. NNLQP [20] implements latency prediction at the model level, but GNN-based encoding scheme makes it inadequate in modeling long range interactions between nodes.

Inspired by the progress in natural language understanding, our network uses a hand designed yet general and extensible token encoding approach to encode the topology information and key parameters of the input neural network. Specifically, we design a generic real value tokenizer to encode neural network nodes' operation type, location, and their inputs into vectors using the positional embedding approach that is widely used in transformers [33] and NERF [25]. This fixed length node encoding scheme makes

---

the network encoding scale linearly with the size of the input network rather than quadratically as in models that rely on taking adjacency matrix as input.

Based on the tokenized input sequence, we design a transformer based model to further encode and fuse the network descriptions to generate a compact representation, *e.g.* a fixed length feature vector. Therefore, long range dependencies between nodes can be established by transformer. Besides the conventional multi-head self attention based transformers, we propose to use a multi-stage fusion transformer structure at the deep stages of the model to further fuse and refine the representation in a cascade manner.

For the network representation learning task, we also propose a data augmentation method called information flow consistency augmentation. The augmentation permutes the encoding order of nodes in the original network under our specified conditions without changing the structure of the network. We find it empirically improves the performance of our method for downstream tasks.

The main contributions of this paper can be summarized as the following points:

1. We propose a simple and effective neural network encoding approach which tokenizes both operation and topology information of a neural network node into a sequence. When it is used to encode the entire network, the tokenized network encoding scheme scales better than adjacency matrix based ones, and builds the foundation for seamlessly using transformer structures for network representation learning.

2. We design a multi-stage fusion transformer to learn feature representations. Benefiting from the proposed tokenizer, a concise pure transformer based neural architecture representation learning framework (NAR-Former) is proposed for the first time. Our NAR-Former makes full use of the capacity of transformers in handling sequence inputs, and gets promising performance.

3. To facilitate efficient model training, we propose an information flow consistency augmentation and correspondingly design an architecture consistency loss, which brings more benefits with less augmentation samples compared with the existing random augmentation strategy.

We conduct extensive experiments on accuracy prediction, neural architecture search as well as latency prediction. Experiments demonstrate the effectiveness of the proposed representation model on processing both cell leveled network components and whole deep neural networks. Specifically, our accuracy predictor based on this representation achieves highly competitive accuracy performance on cell-based structures in NAS-Bench-101 [40] and NAS-Bench-201 [10] datasets. Compared with other predictor-based NAS methods [23, 26], we efficiently find the architecture with 97.52% accuracy in DARTS [19] by only querying 100 neural architectures. We also conduct latency prediction experiments on neural networks deeper than 200 layers to demonstrate the universality of our model.

## 2. Related Work

### 2.1. Representation of Neural Architectures

Neural architecture representation learning is essential for implementing downstream tasks (*e.g.* accuracy prediction and latency prediction). Many accuracy predictors model the input architectures by adopting LSTM [6, 18, 24] and MLP [18, 37], which are used to accelerate NAS. They usually suffer from a limited accuracy because of the insufficient encoding of the topology structure. For example, NAO [24] directly encodes the whole architecture with a flatted vector followed by an embedding layer. Recently, many predictors explore how to use GCNs [4, 17, 31, 36] to model the graph format data(*e.g.* adjacency matrix and node features) of architectures. Neural Predictor [36] encodes each node in architecture with the mean of two modified GCN layers, which use the adjacency matrix and the transpose of adjacency matrix to propagate information, respectively. When the input architecture becomes deeper, there is a quadratically increase of encoding dimension in these methods caused by the adjacency matrix. There are methods [5, 39] that use other models for representation learning. HOP [5] proposes an adaptive attention module to model the relative significance of different network nodes for the first time, and achieved attractive prediction performance. However, the lacking consideration of the networks' overall characteristics leads to the fact that these methods are not suitable for extending to whole networks.

Latency prediction often needs to take the information of whole network into consideration when modeling the input architecture. TPU-Performance [14, 15] and nn-Meter [41] encode architectures by dividing a whole network inference pass into multiple kernels, and calculate the sum of kernels' latency. In real applications, the hardware graph fusion may bring a gap between the sum of kernel latencies and the real latency of the whole network. To alleviate this problem, NNLQP [20] proposes a unified GNN-based representation model for various neural architectures and predicts the latency in model level. But the model in NNLQP [20] is rather preliminary since it generates architecture representations by directly summing up the features of all nodes and is not good at capturing mode sequence information.

### 2.2. Transformers

The transformer model [33], originally proposed in natural language processing(NLP), proposes to draw global
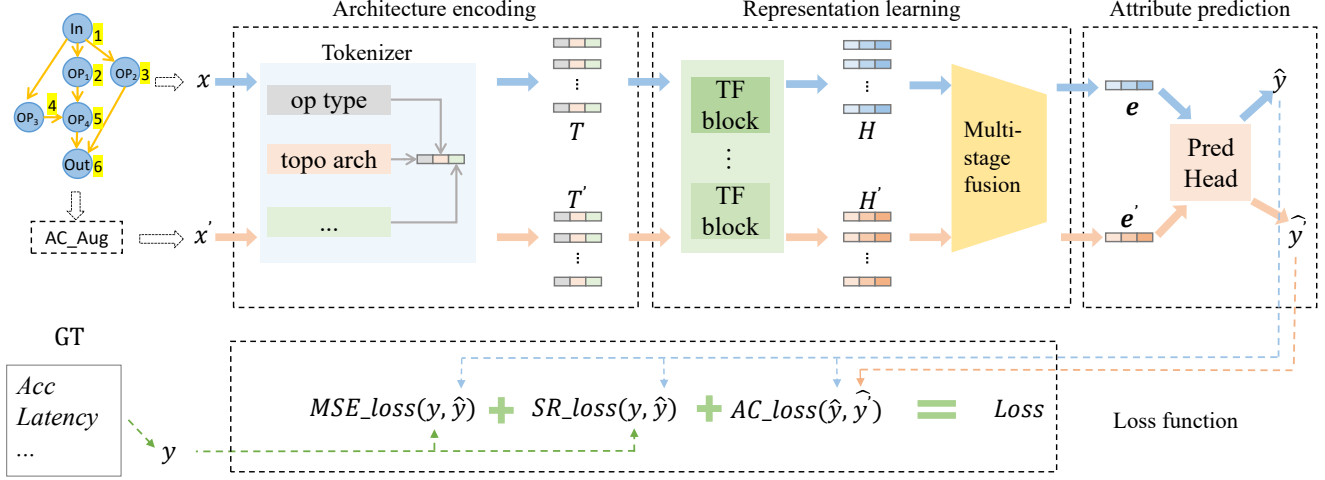
Figure 1. Overview of our NAR-Former. We first encode an input architecture $x$ to a pure sequence $T$ with a proposed tokenizer. A multi-stage fusion transformer is designed to learn a vector representation $e$ from $T$. $x'$(optional) is an augmented architecture of $x$ generated by our information flow consistency augmentation. The bottom part shows the loss function used in this paper. SR_loss is designed for learning more accurate sequence ranking. AC_loss is a proposed architecture consistency loss.

dependencies and realize parallel computation using multi-head self-attention mechanism. Because of the outstanding performance in NLP [8,28] and its domain agnostic network structure, transformers are quickly being used in many other fields [2, 11, 22]. TNASP [23] adopts transformer to encode neural network for the first time, which uses the MLP transform of the Laplacian matrix as the position encoding. It achieves better performance compared to some existing methods, which confirms the advantage of transformer for architecture representation learning. However, the Laplacian matrix of the network with different depths needs to be filled to a preset shape when calculating the position encoding, which may introduce unwanted information. We argue that the ability to handle sequences of variable length and to model long range dependency makes transformer an ideal backbone for neural architecture representation learning, if the given structure is mapped into a single sequence as its input.

## 3. The Proposed Method

### 3.1. Overall Framework

Fig. 1 shows the overall framework of our proposed NAR-Former. Firstly, the input cell or network $x$ that has $N$ nodes will be encoded to a token sequence $T \in \mathbb{R}^{(N+2) \times D}$ with a proposed tokenizer(Sec. 3.2). Then a multi-stage fusion transformer(Sec. 3.3) is designed to transform $T$ to a one-token feature representation $e \in \mathbb{R}^{1 \times D}$. At the last stage, attribute value $\widehat{y}$ will be predicted using a prediction head that consists of fully connected layers and activation layers. After training with $M$ different architecture-attribute pairs $\{(x_i, y_i)\}_{i=1}^{M}$, NAR-Former can predict the

attribute of an unseen architecture by only one forward inference pass, regardless of the number of operations.

We propose an information flow consistency augmentation method to further improve the accuracy of attribute prediction(Sec. 3.4), which is labeled "AC_Aug" in Fig. 1. $x'$ is an architecture obtained by augmenting $x$, and $\widehat{y'}$ is its prediction. Corresponding to the proposed augmentation, an architecture consistency loss is designed(Sec. 3.5).

The bottom part of Fig. 1 illustrates the loss function(Sec. 3.5) we used, consisting of MSE loss and two designed loss, which are a loss related to sequence ranking(SR_loss) and architecture consistency loss(AC_loss).

### 3.2. Architecture Encoding

In order to avoid information loss of the input architecture and use transformer seamlessly, we propose a tokenizer to encode operations and their connections of a neural network into a sequence of tokens. When given a neural network, any operation that exists in this network can be considered as a node. Our tokenizer can encode the operator and topological structure of the input architecture $x$ that has $N$ operations into a $N + 2$ token sequence $T$, where each token $t$ is a $D$ dimension vector.

$$T = (t_1, t_2, \cdots, t_N, t_{\text{End}}, t_{\text{Dep}}). \tag{1}$$

The first $N$ tokens are the tokens for the $N$ nodes and their connections. $t_{\text{End}}$ is an end token that signifies the end of network encoding, and the last token $t_{\text{Dep}}$ encodes the depth of the input network. Each token $t$ in $t_i(i \in (1, 2, \cdots, N, \text{End}))$ is further decomposed into 3 sub vectors:

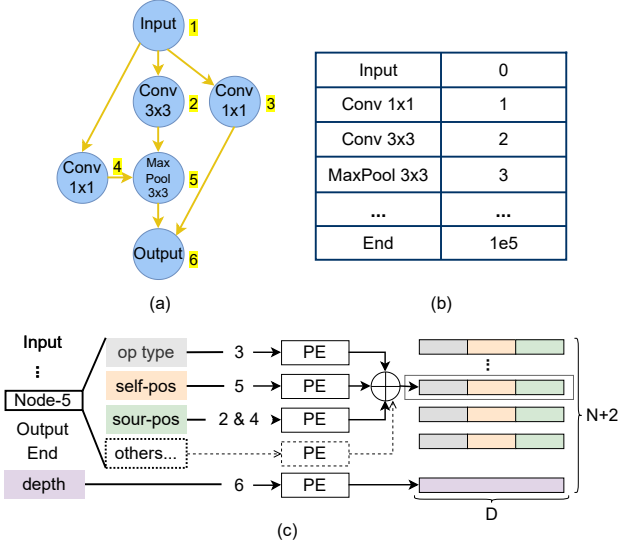$$t_i = \left(t_i^{\text{op}}, t_i^{\text{self}}, t_i^{\text{sour}}\right), \tag{2}$$

Figure 2. (a) An example of architecture with 6 operations($N = 6$). (b) Conversion table from operation categories to indexes. (c) Encoding scheme of our tokenizer.

which are the encodings of operator type $t_i^{\mathrm{op}}$, its location $t_i^{\mathrm{self}}$ and the summarized location of its source nodes $t_i^{\mathrm{sour}}$.

**Generic real number tokenizer.** The information to be encoded above can all be represented first in the form of real numbers. Inspired by the encoders used in NERF [25], we propose the following encoding function $f$ that maps a real number $p$ to a vector in $\mathbb{R}^{2L}$:

$$f(p) = [\sin(b_1 p\pi), \cos(b_1 p\pi), \cdots, \\ \sin(b_L p\pi), \cos(b_L p\pi)]. \quad (3)$$

.

Different from the geometric frequencies $(2^0, 2^1, \cdots, 2^{L-2}, 2^{L-1})$ used in [25], we obtain the frequencies by linear interpolation in range $[2^0, 2^{L-1}]$:

$$\boldsymbol{b} = (2^0, 2^0 + s, \cdots, 2^0 + (L-2)s, 2^{L-1}), \quad (4)$$

where,

$$s = \frac{2^{L-1} - 2^0}{L - 1}. \quad (5)$$

Depending on the range of the encoded real number, the length parameter $L$ can be specified on demand.

**Encoding for Operation Type Information.** As is illustrated in Fig. 2(b), we can use a conversion table to map the operator types into their indexes. The integer index can then be converted to a vector using the Generic real number tokenizer. Denoting $\mathcal{C}(v_i)$ as the operation type index of node $v_i$, the operation type encoding can be defined as

$$t_i^{\mathrm{op}} = f_{\mathrm{op}}(\mathcal{C}(v_i)). \quad (6)$$

The operations that can be encoded here includes, but are not limited to, basic mathematical operation(*e.g.* addition, multiplication), convolution operation, normalization operation, even channel operation(*e.g.* concatenation).

**Encoding for Topology Information.** The encoding of topology structure can be decomposed to encoding of the information flow in each node of the network. For each node, we encode the location of the node itself and its input nodes in the network structure $x$. Location of the node $v_i$ itself ("self-pos" in Fig. 2(c)) is encoded as

$$t_i^{\mathrm{self}} = f_{\mathrm{self}}(i). \quad (7)$$

The locations of its source nodes are encoded into one vector as the sum of all source location encodings ("sour-pos" in Fig. 2(c)):

$$t_i^{\mathrm{sour}} = \sum_{j \in \mathcal{P}(v_i)} f_{\mathrm{sour}}(j), \quad (8)$$

where $\mathcal{P}(v_i)$ is the set of locations of all the node's precursors. Then the concatenation of $t_i^{\mathrm{self}}$ and $t_i^{\mathrm{sour}}$ along the feature dimension is used to represent topology information.

**Special Tokens** For $v_0$, which has no precursor node, we set $\mathcal{P}(v_0)$ equals to $\{-1\}$. For the end token, both the self position encoding and source position encoding can be encodings of a value that is unlikely to appear in the architecture code, such as $1 \times 10^5$ in our experiments.

Considering that the depth of a network is also an important factor affecting its attributes, an additional depth token is encoded by using Eq. (3) to map the depth $N$ into a $D$-length vector:

$$t_{\mathrm{Dep}} = \mathrm{ReLU}(\mathrm{FC}(f_{\mathrm{Dep}}(N))), \quad (9)$$

where FC denotes a fully connected layer.

**Extensibility.** Following the approach to encode operator types and locations, the token $t$ in Eq. (1) can be naturally extended to further encode other information necessary for downstream tasks. For example, we further encode key parameters and output shape of each node by concatenating their encodings from real value tokenizers in our network latency prediction experiments (Sec. 4.6). Due to space limitations, we will introduce the content of these additional encoded information in the supplementary material.

### 3.3. Multi-Stage Fusion Transformer

Our transformer-based feature representation learning model consists of normal transformer blocks and the multi-stage fusion part as shown in Fig. 3. The red, yellow, and green arrows from left to right indicate the value, key, and query inputs for each transformer block, respectively. At the first stage, normal transformer blocks are used to further transform the tokens in $T$ and learn an intermediate token sequence $H$ through global interactions.
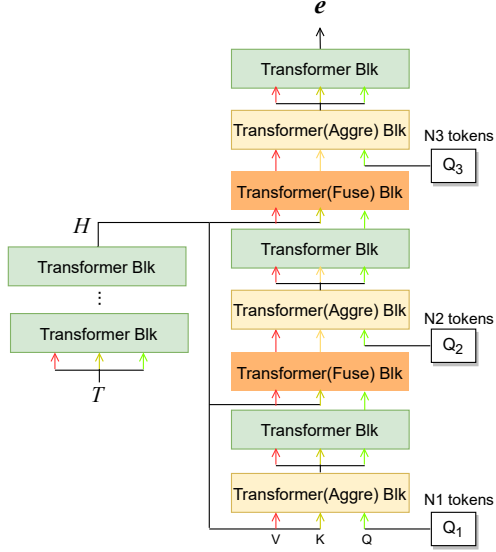
Figure 3. Multi-Stage Fusion Transformer. Token sequence $T$ is first transformed into feature map $H$ by standard transformer blocks, and then gradually fused into a one-token feature vector $\boldsymbol{e}$.

The multi-stage fusion part of our model is consisted of three types transformer blocks which we call aggregation blocks, standard transformer blocks and fusion blocks respectively. The standard blocks are optional depending on the requirements of model size and performance. In the aggregation block, we use a learnable parameter matrix $Q_k$ as the query, and use $H$(only in the first aggregation blok) or the previous fuse block's output as both key and value of the transformer input. Depending on the level parameter $k$, the number of query tokens(denoted as $N_k$) in $Q_k$ is different, which helps shrink (thus aggregate information) the number of output tokens as the level goes up. The fuse block uses the output of a dependent standard transformer as the query, and uses the $H$ as key and value of the transformer input. Specifically, the number of query tokens in the last aggregation block is shrinked to 1. Thus, the output is a single feature vector regardless of the size of the input network $x$.

### 3.4. Information Flow Consistency Augmentation

The amount of training architecture-attribute data pairs has a significant impact on the effectiveness of our model. However, these datasets are usually small because these attributes of architectures are usually expensive to acquire. For the above reasons, we propose a simple but effective data augmentation method.

For a given architecture with $N$ operations, we can label all operations from 1 to $N$ (the numbers to the right of operations in Fig. 4), which indicates the order in which the operations are encoded. Such two different graphs are isomorphic [38] if they have the same node attributes and
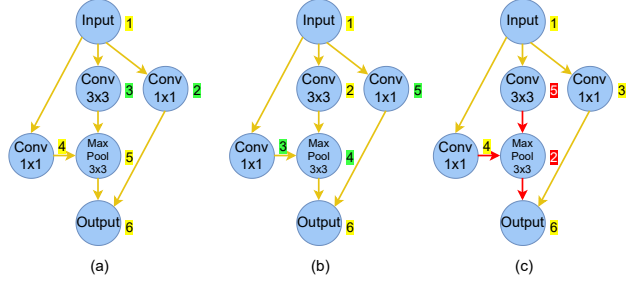


Figure 4. Isomorphic examples of Fig. 2(a). (a) and (b) are augmented architectures that maintain the consistency of information flow, while (c) is not.

node connections, only differ in the order of nodes. For instance, architectures in Fig. 4 and Fig. 2(a) are isomorphic. Recent works [21, 38] use the idea of isomorphism to augment, and use all isomorphic architectures as augmented data. However, we argue that the prior knowledge that the information flows from shallow to deep nodes facilitates the learning of more accurate representations. Therefore, the architecture with edges flowing from deep nodes to shallow nodes as shown in Fig. 4(c) is discarded in our augmentation to maintain the information flow consistency between the augmented structure and the original structure.

### 3.5. Loss Function

The total loss function is as follows:

$$\text{Loss} = \text{MSE\_loss} + \lambda_1 \text{SR\_loss} + \lambda_2 \text{AC\_loss}, \quad (10)$$

where AC_loss is the proposed architecture consistency loss, SR_loss is a loss related to sequence ranking. $\lambda_1$, $\lambda_2$ are the weight coefficients. $\lambda_2$ is equal to zero when there is no augmented data.

**Architecture Consistency Loss.** In order to make full use of the augmented data, we propose an architecture consistency loss(AC_loss) accordingly. It is obvious that the structure obtained by augmentation is equivalent to the original one in terms of operation and topology information. For this reason, we propose such a loss term that their attribute prediction results should be consistent:

$$\text{AC\_loss}(\widehat{y}, \widehat{y'}) = \sum_{i=1}^{M} |\widehat{y_i} - \widehat{y'_i}|. \quad (11)$$

To train models for the down stream attributes prediction task, we use the Mean Squared Error(MSE) between the predicted results and the true attributes to guide the predictor to generate accurate predictions:

$$\text{MSE\_loss}(y, \widehat{y}) = \sum_{i=1}^{M} (\widehat{y_i} - y_i)^2. \quad (12)$$

For tasks such as NAS, the prediction accuracy of architectures' relative ranking is more important than that of absolute attribute value. Based on this observation, we introduce a loss related to sequence ranking:

$$\text{SR\_loss}(y, \widehat{y}) = \sum_{i=1}^{M} [(\widehat{y}_{I(i)} - \widehat{y}_i) - (y_{I(i)} - y_i)]. \quad (13)$$

$I(\cdot)$ is the result of random shuffling of sequences 1 to $M$. Note that when architecture augmentation is used, the augmented architectures will also participate in the calculation of MSE_loss and SR_loss.

## 4. Experiments

In Sec. 4.2 and Sec. 4.3, we apply our NAR-Former first on NAS-Bench-101 [40] and NAS-Bench-201 [10] to demonstrate the superiority of our representation in accuracy prediction tasks. Then in Sec. 4.4 and Sec. 4.5, experiments are conducted to explore the effect of using our predictor on the efficiency and performance of NAS. A small-scale latency prediction experiment is used to verify the generality of the proposed representation model as shown in Sec. 4.6. We conduct ablation studies in Sec. 4.7 to show the effeciveness of our model design choices.

### 4.1. Implementation Details.

For the tokenizers, we fix $L$ for $f_{op}, f_{self}, f_{sour}$ all at 32 in Sec. 4.2, Sec. 4.3, and Sec. 4.4. For settings in Sec. 3.3, we use 6 standard transformer blocks to get $H$ and don't use standard transformer blocks during the fusion stage. We fix the ratio of the hidden dimension to the input dimension in the feed forward networks of transformers to 1:4 and the dimension of each head in multi-head attention to 32. $N_1$, $N_2$ and $N_3$ in Fig. 3 are fixed to 4, 2, and 1, respectively. $\lambda_1$ is fixed to 0.1 for accurate and latency prediction and 0 for other experiments. $\lambda_2$ is set to 0.5 in Sec. 4.2 and Sec. 4.3. In section Sec. 4.4, Sec. 4.5, and Sec. 4.6, $\lambda2$ is equal to 0 because augmentation is not used. More details about implementation are provide in the supplementary material.

### 4.2. Experiments on NAS-Bench-101

**Setup.** NAS-Bench-101 [40] is a "operation on nodes" (OON) space consisting of 423,624 unique architectures. In this experiment, we use Kendall's Tau [30] as the evaluation metric, and a higher value indicates that the relative order of the predicted values is more related to that of the ground-truth values. Following the settings in [23], we use 0.1%, 1% and 50% of the whole data as training set. Another 200 samples are used as validation set.
**Result Comparisons.** The results are shown in Tab. 1, where the values except for our method are reported by TNASP [23]. When the amount of available training samples is very small, e.g. with only 424, the more recent

Table 1. Results on NAS-Bench-101 [40](depth=2∼7). Kendall's Tau is calculated using predicted accuracies and ground-truth accuracies. Three different proportions of the whole dataset are used as the training set. "SE" refer to self-evolution proposed by TNASP [23] to improve prediction performance.

| Backbone | Method | Training Samples | | | |
|---|---|---|---|---|---|
| | | 0.1% (424) | 0.1% (424) | 1% (4236) | 5% (21180) |
| | | Test Samples | | | |
| | | 100 | all | all | all |
| CNN | ReNAS [39] | 0.634 | 0.657 | 0.816 | - |
| LSTM | NAO [24] | 0.704 | 0.666 | 0.775 | - |
| | NAO+SE | 0.732 | 0.680 | 0.787 | - |
| GCN | NP [36] | 0.710 | 0.679 | 0.769 | - |
| | NP + SE | 0.713 | 0.684 | 0.773 | - |
| | CTNAS [4] | 0.751 | - | - | - |
| Transformer | TNASP [23] | 0.752 | 0.705 | 0.820 | - |
| | TNASP + SE | 0.754 | 0.722 | 0.820 | - |
| | NAR-Former | **0.801** | **0.765** | **0.871** | **0.891** |

Table 2. Results of on NAS-Bench-201 [10](depth=8). Kendall's Tau is calculated using predicted accuracies and ground-truth accuracies. Three different proportions of the whole dataset are used as the training set. "SE" refer to self-evolution proposed by TNASP [23] to improve prediction performance.

| Backbone | Model | Training Samples | | |
|---|---|---|---|---|
| | | (781) | (1563) | (7812) |
| | | 5% | 10% | 50% |
| LSTM | NAO [24] | 0.522 | 0.526 | - |
| | NAO + SE | 0.529 | 0.528 | - |
| GCN | NP [36] | 0.634 | 0.646 | - |
| | NP + SE | 0.652 | 0.649 | - |
| Transformer | TNASP [23] | 0.689 | 0.724 | - |
| | TNASP + SE | 0.690 | 0.726 | - |
| | NAR-Former | **0.849** | **0.901** | **0.947** |

graph-based methods [4] [36] can obtain higher score than sequence-based methods [24]. When more samples are available for training, CNN-based method [39] performs better. However, results of above methods are not as good as those of TNASP [23] in all settings, which confirms the superiority of transformer models in performance prediction. By making better use of transformer's capacity and leveraging proposed training strategies, our NAR-Former achieves much higher metrics compared to TNASP [23], with an average improvement of 0.047 in the above three settings.

### 4.3. Experiments on NAS-Bench-201

**Setup.** NAS-Bench-201 [10] has 15,625 different cell candidates, all of which are composed of 4 nodes and 6 edges. Unlike NAS-Bench-101 [40], architectures in NAS-Bench-201 [10] have its operations on the edges, which belong to "operation on edge" (OOE) search space. This change of emphasis has no impact to our implementation, since only the operation types and connection relations are needed to learn the representation. The evaluation indicators is

Kendall's Tau [30]. The dataset is still divided according to [23], where 5%, 10% and 50% of the whole dataset is used for training, and another 200 samples are used for validation. Kendall's Tau is calculated on the whole dataset.

**Results Comparison.** As shown in Tab. 2, whose values except for our method are reported by TNASP [23], earlier method [24] using LSTM for encoding has the ability to roughly predict accuracy. Compared with NAO [24], there is a marked increase in the results of the GCN-based method [36]. Taking advantage of transformer , TNASP [23] makes a further improvement. As we can see, our NAR-Former significantly outperforms by 0.159, 0.175 compared to the previous highest TNASP+SE [23] under two different experiment settings, respectively. This provides evidence that NAR-Former can learn pretty effective representation of architectures for accuracy prediction.

Table 3. Performance of searched architectures using different NAS algorithms in DARTS [19] space on CIFAR-10 [16]. † denotes using cutout [9] as data augmentation.

| Model | Params (M) | Top1 Acc(%) | No. of archs | Search Cost(G·D) |
|---|---|---|---|---|
| VGG-19 [43] | 20.0 | 95.10 | 0 | 0 |
| DenseNet-BC [13] | 25.6 | 96.54 | 0 | 0 |
| Swin-S [22] | 50 | 94.17 | 0 | 0 |
| Nest-S [42] | 38 | 96.97 | 0 | 0 |
| Ransom search | 3.2 | 96.71 | - | - |
| NASNet-A† [44] | 3.3 | 97.35 | 20000 | 1800 |
| AmoebaNet-A† [29] | 3.2 | 96.66 | 27000 | 3150 |
| PNAS [18] | 3.2 | 96.59 | 1160 | 225 |
| NAONet [24] | 28.6 | 97.02 | 1000 | - |
| GATES† [26] | 4.1 | 97.42 | 800 | - |
| ENAS† [27] | 4.6 | 97.11 | - | 0.5 |
| DARTS† [19] | 3.4 | 97.24 | - | 4 |
| CTNAS† [4] | 3.6 | 97.41 | - | 0.3 |
| TNASP† [23] | 3.7 | 97.48 | 1000 | 0.3 |
| NAR-Former† | 3.8 | **97.52** | **100** | 0.24 |

Table 4. Average cost for one sample of NAS-Bench-101 [40].

| Model | Encode(ms) | Infer(ms) | Total(ms) |
|---|---|---|---|
| TNASP [23] | 0.0001 | 1.1030 | 1.1031 |
| NAR-Former | 2.4784 | 6.4747 | 8.9531 |

## 4.4. Experiments on DARTS

**Setup.** DARTS [19] is an open-domain search space, where each architecture consists of normal and reduced cells with 7 nodes and 14 edges. We use three metrics, which are the number of parameters of the searched architecture, the test accuracy on CIFAR-10 [16] and the number of structures queried during the search. We use an evolutionary algorithm following NPENAS [35]. Firstly, we randomly sample 10 architectures and train them for 50 epochs on CIFAR-10 as the initial architecture pool. At each round

of evolution, we use our predictor that is trained with the structure-accuracy pairs in the structure pool to select the top-10 from the candidate structures generated by mutation, and then the selected structures are added to the structure pool after being trained for 50 epochs. The above steps are repeated until a total of 100 structures are queried. Among these 100 architectures, we choose the architecture with the highest accuracy on validation set to retrain and test.

**Results Comparison.** Results of this part are shown in Tab. 3. For predictor-based methods, The search cost does not include the time to train the supernet or train the queried architecture to obtain the ground truth. Since 2020, the methods adopting graph-based predictor have achieved better results than previous methods, such as 97.41% for CTNAS [4] and 97.42% for GATES [26]. The transformer-based predictor TNASP [23] further improved the accuracy to 97.48%. Taking advantage of our predictor, we continue to improve the accuracy slightly. Compared to [23], although our method has a disadvantage in the speed of a single run as shown in Tab. 4, only one-tenth of the queried architectures help us reduce our search time by 20%.

## 4.5. Experiments on MobileNet Space

With the exception of the performance predictor being replaced with our NAR-Former, we strictly follow OFA's [1] search and evaluation settings. As shown in Tab. 7, it takes much less time to search for two architectures of different sizes than OFA, and the Top1 accuracy on ImageNet [7] is still improved by 0.36% and 0.9% respectively in the mobile setting (<600M MACs).

## 4.6. Experiments on NNLQP

**Setup.** NNLQP [20] is a latency dataset built for evaluating the performance of latency prediction models. We use the "unseen structure" part of this dataset to conduct latency prediction experiments, which contains variants of 9 state-of-the-art DNN models(*e.g.* ResNet [12]) and NAS-Bench-201 [10] model, with 2000 different architectures in each model type. The metrics used in this part are Mean Absolute Percentage Error (MAPE) and Error Bound Accuracy (Acc($\delta$)), which are the same as used in [20].

**Result Comparison.** As can be seen from Tab. 6, our method can be applied for both the EfficientNet [32] and its variants with depths of 242 and the NAS-Bench-201 networks [10] with depths ranging from 112 to 247. Combined with the experiments above, these experiment reflect that our model can be used to model structures of very different lengths and to predict different attributes.

## 4.7. Ablation Study

We conduct a series of accuracy prediction experiments to verify our contributions following the setting of 1% proportion in Sec. 4.2. The results are shown in Tab. 5. The ex-

Table 5. Ablation study. The "Self_ID" represents the way in which the self position information is combined with other information. The number in parentheses indicates the amount of augmented data. All experiments follow the setting of 1% proportion in Sec. 4.2.

| Row | Architencture Encoder | Predictor Type | Self_ID | SR_loss | GI Aug [38] (+3812) | Our Aug (+2421 ) | AC_loss | Kendall's Tau |
|-----|-----|-----|-----|-----|-----|-----|-----|-----|
| 1 | TNASP [23] | Transformer in [23] | - | - | - | - | - | 0.8200 |
| 2 | NP [36] | GCN in [36] | - | - | - | - | - | 0.7694 |
| 3 | Tokenizer | Transformer in [23] | Add | - | - | - | - | 0.8416 |
| 4 | Tokenizer | Transformer in [23] | Concat | - | - | - | - | 0.8477 |
| 5 | Tokenizer | GCN in [36] | Concat | - | - | - | - | 0.7953 |
| 6 | Tokenizer | Multi-stage fusion | Concat | - | - | - | - | 0.8481 |
| 7 | Tokenizer | GCN in [36] | Concat | - | - | ✓ | - | 0.8035 |
| 8 | Tokenizer | GCN in [36] | Concat | - | - | ✓ | ✓ | 0.8060 |
| 9 | Tokenizer | Multi-stage fusion | Concat | ✓ | - | - | - | 0.8495 |
| 10 | Tokenizer | Multi-stage fusion | Concat | ✓ | ✓ | - | - | 0.8625 |
| 11 | Tokenizer | Multi-stage fusion | Concat | ✓ | ✓ | - | ✓ | 0.8643 |
| 12 | Tokenizer | Multi-stage fusion | Concat | ✓ | - | ✓ | - | 0.8579 |
| 13 | Tokenizer | Multi-stage fusion | Concat | ✓ | - | ✓ | ✓ | 0.8712 |

Table 6. Latency prediction on NNLQP [20]. "Test Model" denotes the model type that used as test set.

| Test Model | Method | MAPE↓ | ACC(10%)↑ |
|-----|-----|-----|-----|
| EfficientNet depth=242 | FLOPs | 58.36% | 0.05% |
| | TPU [14] | **16.74%** | 17.00% |
| | NNLQP [20] | 21.33% | **24.65%** |
| | NAR-Former | 28.05% | 24.08% |
| Nas-Bench-201 depth=112~247 | FLOPs | 80.41% | 0.00% |
| | TPU [14] | 58.94% | 2.50% |
| | NNLQP [20] | 8.76% | 67.10% |
| | NAR-Former | **4.19%** | **95.12%** |

Table 7. Verification of predictor's effectiveness using neural structure search experiments on MobileNet space.

| Model | ImageNet Top1(%) | MACs | Search Cost (GPU hours) |
|-----|-----|-----|-----|
| OFA [1] | 76.00 | 230M | 40 |
| NAR-Former | 76.36 | 378M | 1.63 |
| NAR-Former | 76.90 | 571M | 2.00 |

periments of Row(3)-Row(6) aim to find the optimal structure. By encoding the architecture into a pure sequence with mapping function Eq. (3) and seamlessly using transformer, we achieve a significant improvement (0.0216) compared to the similar transformer-based baseline [23](Row(3) vs Row(1)). We argue that, compared with directly adding self position encoding in traditional way, the concatenation of self position information and source position information introduced in Sec. 3.2 shows a superior representation of topology information(See Row(3) and Row(4)). A comparison between Row(5) and Row(2) shows that the tokenizer we designed is also applicable to the GCN-based approach.

We explore other strategies to further improve performance, shown as Row(7)-Row(13). For more accurate prediction of relative ranking, we propose SR_loss, which precisely measures the difference between the predicted ordering and the ground-truth ordering of two inputs. The results in Row(9) and Row(6) show that it works. By the comparison with Row(9), the results in Row(10) and

Row(12) demonstrate that both augmentation methods are useful. The data set generated by our AC_Aug is a subset of that generated by GIAug. The additional data generated by GI_Aug can be seen as a secondary augmentation, which brings more benefit(0.0130) than AC_Aug(0.0084) when only using data augmentation. The extra augmentation samples can not keep the consistency of information flow, making it less advantageous in obtaining performance improvement when using AC_loss(Row(11) and Row(13)). The results in Row(7) and Row(8), compared with Row(5), also verify the effectiveness of the proposed AC_Aug and AC_loss on the GCN-based model, respectively.

## 5. Conclusion

We propose an effective neural architecture representation learning framework that are consisted of linearly scaling network encoders, a transformers based representation learning model, and an effective model training method with data augmentations and assisted loss functions. Experiments show that our framework are capable of improving the accuracy of downstream prediction tasks while overcoming scale limitations on input architectures. Although not the scope of this work, we believe this framework can also be extended for other down stream tasks, such as predicting the quantization loss or searching for the best mixed precision model inference strategies.

# References

[1] Han Cai, Chuang Gan, Tianzhe Wang, Zhekai Zhang, and Song Han. Once-for-all: Train one network and specialize it for efficient deployment. *arXiv preprint arXiv:1908.09791*, 2019. 7, 8

[2] Nicolas Carion, Francisco Massa, Gabriel Synnaeve, Nicolas Usunier, Alexander Kirillov, and Sergey Zagoruyko. End-to-end object detection with transformers. In *European conference on computer vision*, pages 213–229. Springer, 2020. 3

[3] Tianqi Chen, Thierry Moreau, Ziheng Jiang, Lianmin Zheng, Eddie Yan, Haichen Shen, Meghan Cowan, Leyuan Wang, Yuwei Hu, Luis Ceze, et al. {TVM}: An automated {End-to-End} optimizing compiler for deep learning. In *13th USENIX Symposium on Operating Systems Design and Implementation (OSDI 18)*, pages 578–594, 2018. 1

[4] Yaofo Chen, Yong Guo, Qi Chen, Minli Li, Wei Zeng, Yaowei Wang, and Mingkui Tan. Contrastive neural architecture search with neural architecture comparators. In *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition*, pages 9502–9511, 2021. 1, 2, 6, 7

[5] Ziye Chen, Yibing Zhan, Baosheng Yu, Mingming Gong, and Bo Du. Not all operations contribute equally: Hierarchical operation-adaptive predictor for neural architecture search. In *Proceedings of the IEEE/CVF International Conference on Computer Vision*, pages 10508–10517, 2021. 1, 2

[6] Boyang Deng, Junjie Yan, and Dahua Lin. Peephole: Predicting network performance before training. *arXiv preprint arXiv:1712.03351*, 2017. 1, 2

[7] Jia Deng, Wei Dong, Richard Socher, Li-Jia Li, Kai Li, and Li Fei-Fei. Imagenet: A large-scale hierarchical image database. In *2009 IEEE conference on computer vision and pattern recognition*, pages 248–255. Ieee, 2009. 7

[8] Jacob Devlin, Ming-Wei Chang, Kenton Lee, and Kristina Toutanova. Bert: Pre-training of deep bidirectional transformers for language understanding. *arXiv preprint arXiv:1810.04805*, 2018. 3

[9] Terrance DeVries and Graham W Taylor. Improved regularization of convolutional neural networks with cutout. *arXiv preprint arXiv:1708.04552*, 2017. 7

[10] Xuanyi Dong and Yi Yang. Nas-bench-201: Extending the scope of reproducible neural architecture search. *arXiv preprint arXiv:2001.00326*, 2020. 2, 6, 7

[11] Alexey Dosovitskiy, Lucas Beyer, Alexander Kolesnikov, Dirk Weissenborn, Xiaohua Zhai, Thomas Unterthiner, Mostafa Dehghani, Matthias Minderer, Georg Heigold, Sylvain Gelly, et al. An image is worth 16x16 words: Transformers for image recognition at scale. *arXiv preprint arXiv:2010.11929*, 2020. 3

[12] Kaiming He, Xiangyu Zhang, Shaoqing Ren, and Jian Sun. Deep residual learning for image recognition. In *Proceedings of the IEEE conference on computer vision and pattern recognition*, pages 770–778, 2016. 7

[13] Gao Huang, Zhuang Liu, Laurens Van Der Maaten, and Kilian Q Weinberger. Densely connected convolutional networks. In *Proceedings of the IEEE conference on computer vision and pattern recognition*, pages 4700–4708, 2017. 7

[14] Sam Kaufman, Phitchaya Phothilimthana, Yanqi Zhou, Charith Mendis, Sudip Roy, Amit Sabne, and Mike Burrows. A learned performance model for tensor processing units. *Proceedings of Machine Learning and Systems*, 3:387–400, 2021. 2, 8

[15] Samuel Kaufman, Phitchaya Mangpo Phothilimthana, and Mike Burrows. Learned tpu cost model for xla tensor programs. In *Proc. Workshop ML Syst. NeurIPS*, pages 1–6, 2019. 2

[16] Alex Krizhevsky, Geoffrey Hinton, et al. Learning multiple layers of features from tiny images. 2009. 7

[17] Wei Li, Shaogang Gong, and Xiatian Zhu. Neural graph embedding for neural architecture search. In *Proceedings of the AAAI Conference on Artificial Intelligence*, volume 34, pages 4707–4714, 2020. 1, 2

[18] Chenxi Liu, Barret Zoph, Maxim Neumann, Jonathon Shlens, Wei Hua, Li-Jia Li, Li Fei-Fei, Alan Yuille, Jonathan Huang, and Kevin Murphy. Progressive neural architecture search. In *Proceedings of the European conference on computer vision (ECCV)*, pages 19–34, 2018. 1, 2, 7

[19] Hanxiao Liu, Karen Simonyan, and Yiming Yang. Darts: Differentiable architecture search. *arXiv preprint arXiv:1806.09055*, 2018. 2, 7

[20] Liang Liu, Mingzhu Shen, Ruihao Gong, Fengwei Yu, and Hailong Yang. Nnlqp: A multi-platform neural network latency query and prediction system with an evolving database. In *51 International Conference on Parallel Processing - ICPP*, ICPP '22. Association for Computing Machinery, 2022. 1, 2, 7, 8

[21] Yuqiao Liu, Yehui Tang, and Yanan Sun. Homogeneous architecture augmentation for neural predictor. In *Proceedings of the IEEE/CVF International Conference on Computer Vision*, pages 12249–12258, 2021. 5

[22] Ze Liu, Yutong Lin, Yue Cao, Han Hu, Yixuan Wei, Zheng Zhang, Stephen Lin, and Baining Guo. Swin transformer: Hierarchical vision transformer using shifted windows. In *Proceedings of the IEEE/CVF International Conference on Computer Vision*, pages 10012–10022, 2021. 3, 7

[23] Shun Lu, Jixiang Li, Jianchao Tan, Sen Yang, and Ji Liu. Tnasp: A transformer-based nas predictor with a self-evolution framework. *Advances in Neural Information Processing Systems*, 34:15125–15137, 2021. 1, 2, 3, 6, 7, 8

[24] Renqian Luo, Fei Tian, Tao Qin, Enhong Chen, and Tie-Yan Liu. Neural architecture optimization. *Advances in neural information processing systems*, 31, 2018. 1, 2, 6, 7

[25] Ben Mildenhall, Pratul P Srinivasan, Matthew Tancik, Jonathan T Barron, Ravi Ramamoorthi, and Ren Ng. Nerf: Representing scenes as neural radiance fields for view synthesis. *Communications of the ACM*, 65(1):99–106, 2021. 1, 4

[26] Xuefei Ning, Yin Zheng, Tianchen Zhao, Yu Wang, and Huazhong Yang. A generic graph-based neural architecture encoding scheme for predictor-based nas. In *European Conference on Computer Vision*, pages 189–204. Springer, 2020. 1, 2, 7

[27] Hieu Pham, Melody Guan, Barret Zoph, Quoc Le, and Jeff Dean. Efficient neural architecture search via parameters

sharing. In *International conference on machine learning*, pages 4095–4104. PMLR, 2018. 7

[28] Alec Radford, Karthik Narasimhan, Tim Salimans, and Ilya Sutskever. Improving language understanding by generative pre-training. `https://s3-us-west-2.amazonaws.com/openai-assets/research-covers/language-unsupervised/language_understanding_paper.pdf`, 2018. 3

[29] Esteban Real, Alok Aggarwal, Yanping Huang, and Quoc V Le. Regularized evolution for image classifier architecture search. In *Proceedings of the aaai conference on artificial intelligence*, volume 33, pages 4780–4789, 2019. 7

[30] Pranab Kumar Sen. Estimates of the regression coefficient based on kendall's tau. *Journal of the American statistical association*, 63(324):1379–1389, 1968. 6, 7

[31] Han Shi, Renjie Pi, Hang Xu, Zhenguo Li, James Kwok, and Tong Zhang. Bridging the gap between sample-based and one-shot neural architecture search with bonas. *Advances in Neural Information Processing Systems*, 33:1808–1819, 2020. 1, 2

[32] Mingxing Tan and Quoc Le. Efficientnet: Rethinking model scaling for convolutional neural networks. In *International conference on machine learning*, pages 6105–6114. PMLR, 2019. 7

[33] Ashish Vaswani, Noam Shazeer, Niki Parmar, Jakob Uszkoreit, Llion Jones, Aidan N Gomez, Łukasz Kaiser, and Illia Polosukhin. Attention is all you need. *Advances in neural information processing systems*, 30, 2017. 1, 2

[34] Linnan Wang, Yiyang Zhao, Yuu Jinnai, Yuandong Tian, and Rodrigo Fonseca. Alphax: exploring neural architectures with deep neural networks and monte carlo tree search. *arXiv preprint arXiv:1903.11059*, 2019. 1

[35] Chen Wei, Chuang Niu, Yiping Tang, Yue Wang, Haihong Hu, and Jimin Liang. Npenas: Neural predictor guided evolution for neural architecture search. *IEEE Transactions on Neural Networks and Learning Systems*, 2022. 7

[36] Wei Wen, Hanxiao Liu, Yiran Chen, Hai Li, Gabriel Bender, and Pieter-Jan Kindermans. Neural predictor for neural architecture search. In *European Conference on Computer Vision*, pages 660–676. Springer, 2020. 1, 2, 6, 7, 8

[37] Colin White, Willie Neiswanger, and Yash Savani. Bananas: Bayesian optimization with neural architectures for neural architecture search. In *Proceedings of the AAAI Conference on Artificial Intelligence*, volume 35, pages 10293–10301, 2021. 2

[38] Xiangning Xie, Yuqiao Liu, Yanan Sun, Mengjie Zhang, and Kay Chen Tan. Architecture augmentation for performance predictor based on graph isomorphism. *arXiv preprint arXiv:2207.00987*, 2022. 5, 8

[39] Yixing Xu, Yunhe Wang, Kai Han, Yehui Tang, Shangling Jui, Chunjing Xu, and Chang Xu. Renas: Relativistic evaluation of neural architecture search. In *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition*, pages 4411–4420, 2021. 2, 6

[40] Chris Ying, Aaron Klein, Eric Christiansen, Esteban Real, Kevin Murphy, and Frank Hutter. Nas-bench-101: Towards reproducible neural architecture search. In *International Conference on Machine Learning*, pages 7105–7114. PMLR, 2019. 2, 6, 7

[41] Li Lyna Zhang, Shihao Han, Jianyu Wei, Ningxin Zheng, Ting Cao, Yuqing Yang, and Yunxin 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*, pages 81–93, 2021. 2

[42] Zizhao Zhang, Han Zhang, Long Zhao, Ting Chen, Sercan Ö Arik, and Tomas Pfister. Nested hierarchical transformer: Towards accurate, data-efficient and interpretable visual understanding. In *Proceedings of the AAAI Conference on Artificial Intelligence*, volume 36, pages 3417–3425, 2022. 7

[43] Chen Zhu, Renkun Ni, Zheng Xu, Kezhi Kong, W Ronny Huang, and Tom Goldstein. Gradinit: Learning to initialize neural networks for stable and efficient training. *Advances in Neural Information Processing Systems*, 34:16410–16422, 2021. 7

[44] Barret Zoph and Quoc V Le. Neural architecture search with reinforcement learning. *arXiv preprint arXiv:1611.01578*, 2016. 7