A Generic Graph-based Neural Architecture Encoding Scheme with Multifaceted Information

Xuefei Ning, Yin Zheng, Zixuan Zhou, Tianchen Zhao, Huazhong Yang *Fellow, IEEE*, Yu Wang *Fellow, IEEE*

Abstract—Neural architecture search (NAS) can automatically discover well-performing architectures in a large search space and has been shown to bring improvements to various applications. However, the computational burden of NAS is huge, since exploring a large search space can need evaluating more than thousands of architecture samples. To improve the sample efficiency of search space exploration, predictor-based NAS methods learn a performance predictor of architectures, and utilize the predictor to sample worth-evaluating architectures. The encoding scheme of NN architectures is crucial to the predictor's generalization ability, and thus crucial to the efficacy of the NAS process. To this end, we have designed a generic Graph-based neural ArchiTecture Encoding Scheme (GATES), a more reasonable modeling of NN architectures that mimics their data processing. Nevertheless, GATES is unaware of the concrete computing semantic of NN operations or architectures. Thus, the learning of operation embeddings and weights in GATES can only exploit the information in architectures-performance pairs. We propose GATES++, which incorporates multifaceted information about NN's operation-level and architecture-level computing semantics into its construction and training, respectively. Experiments on benchmark search spaces show that both the operation-level and architecture-level information can bring improvements alone, and GATES++ can discover better architectures after evaluating the same number of architectures.

Index Terms—Neural Architecture Search, Predictor-based NAS, Graph-based Encoding, Zero-Shot Information, Ranking Loss

1 Introduction

The evolution of neural network (NN) architectures is one of the key driving forces of the rapid development of deep learning. In the early years, the design of NN architectures relies on solely manual knowledge and experiences [1], [2], [3], [4]. While recently, neural architecture search (NAS) [5], [6] emerged as an automatic design tool of NN architectures. NAS has received substantial attention and has brought improvements to various applications.

In the pioneering work of [5], the architecture evaluation is conducted by training every candidate architecture for 50 epochs, and thousands of architectures need to be evaluated to explore the large search space. As a result, the overall NAS process is extremely costly ($\sim$48k GPU hours). One direction to alleviate the computational burden of NAS is to improve the sample efficiency of the architecture searching module, so that fewer architectures need to be evaluated for discovering a good architecture [7], [8], [9], [10].

Along this direction, a promising idea is to learn an approximated performance predictor, and then utilize the predictor to sample architectures that are more worth evaluating. We refer to these NAS methods [8], [9], [10] as the predictor-based NAS methods, and we summarize their general flow in Sec. 3. The generalization ability of the predictor is crucial to the sample efficiency of the predictor-based NAS flow. Our work follows the line of research of predictor-based NAS, and focuses on improving both the construction and training of the performance predictor.

- *X. Ning, Z. Zhou, T. Zhao, H. Yang, Y Wang were with the Department of Electronic Engineering, Tsinghua University, China.*
  E-mail: foxdoraame@gmail.com (X. Ning), yanghz@tsinghua.edu.cn (H. Yang), yu-wang@tsinghua.edu.cn (Y. Wang)
- *Y. Zheng is with Wechat group, Tencent.*

From the construction aspect, a performance predictor needs to encode architectures into vectors in a continuous space. And existing neural architecture encoding schemes include the sequence-based scheme and the graph-based scheme. The sequence-based schemes [8], [9], [10] rely on some specific serialization of the architecture. They do not model the topological information explicitly, which deteriorates the representational power of the predictor. Existing graph-based schemes [11], [11], [12] apply graph convolutional networks (GCN) [13] to encode the neural architectures. These schemes embed the operations (e.g., `Conv3x3`, `MaxPool`) as node embeddings, aggregate these node embeddings on the directed acyclic graph (DAG) for several steps, and finally use the aggregated node embeddings to construct a global embedding as the architecture representation. In other words, these methods embed the NN operations as the information to be passed and aggregated on the graph. Note that these schemes neglect the fact that a neural architecture is a "data processing" graph, where the operations behave as data processing "functions" instead of the "data" itself.

Following this intuition, we have proposed a general encoding scheme: *Graph-based neural ArchiTecture Encoding Scheme (GATES)*, which is suitable for the representation learning of data processing graphs such as neural architectures. Specifically, to encode a neural architecture, GATES models the information flow of the actual data processing of the architecture. First, GATES models the input information as the attributes of the input nodes. And the input information will be propagated along the architecture DAG. GATES models the operations (e.g., `Conv3x3`, `MaxPool`) as transforms of the information, instead of the information itself. Finally, the output information is used as the embedding of the cell architecture. Since the encoding process of GATES mimics the actual computation flow of the architectures, GATES intrinsically maps isomorphic architectures to the same representation. Moreover, GATES can encode architectures from different search spaces in a consistent way.

From the training aspect, we realized that it is more important for the predictor to give out correct architecture ranking rather than absolute accurate predictions in the NAS process. Therefore, we have proposed to use ranking losses to train the predictor. Using ranking losses helps to improve the ranking quality of architecture predictor, since ranking losses are better surrogates of the ranking measures.

The design of GATES and the usage of training loss correspond to the characteristics of NN architectures and NAS problem. Both techniques significantly boost the generalization ability of the predictor to unseen architectures.

However, the construction and training of GATES are unaware of the concrete computing semantics of NN operations and architectures. That is to say, the learning of operation embeddings and weights in GATES can only exploit the information in the pairs of architecture descriptions and ground-truth performances. To further empower GATES, this paper proposes GATES++, which incorporates multifaceted information about NN's operation-level and architecture-level computing semantics into its construction and training. 1) Into the predictor construction: We concatenate operation-level zero-shot metrics onto the trainable operation embedding in the GATES construction. This information provides prior knowledge to the predictor regarding NN's actual operation-level computing semantics. For example, almost all types of zero-shot metrics can reflect that `Conv3x3` is more similar to `Conv5x5` than `MaxPool`. 2) Into the predictor training: We add an auxiliary regression head onto GATES to fit various zero-shot metrics. These metrics provide multifaceted architecture-level information of NN, and learning to fit this information might encourage the encoder to learn better architecture representation.

The contributions of this paper are as follows.

- We propose a Graph-based neural ArchiTecture Encoding Scheme (GATES), a more reasonable modeling of NN architectures that mimics their data processing. In contrast to existing GCN encoding schemes that model NN operations as node attributes (i.e., node information) to be propagated and aggregated, GATES models the NN operations as the transforms of node attributes.
- We propose to employ ranking losses to train the predictor to achieve better ranking quality, since it is more important for the predictor to give out correct architecture ranking rather than absolute accurate predictions.
- Based on GATES, we propose a Graph-based neural ArchiTecture Encoding Scheme++ (GATES++), which incorporates multifaceted information about NN's operation-level and architecture-level computing semantics into its construction and training. Specifically, operation-level zero-shot metrics are concatenated onto the trainable operation embedding of the GATES construction, and an auxiliary regression to fit architecture-level zero-shot metrics is added in the training process.
- Experimental results show that GATES and GATES++ consistently outperform baseline predictors when using different numbers of training architectures. And the multifaceted architectural information incorporated helps GATES++ to achieve a good ranking quality with a minimal number of training architectures.

This paper extends our preliminary version (GATES) [14] from several aspects. 1) GATES++ extends the encoding scheme in GATES by concatenating operation-level zero-shot metrics onto the original operation embeddings. This brings operation-level information into the encoding process. 2) GATES++ extends the training scheme in GATES by adding an auxiliary regression head and encouraging it to fit multiple architecture-level zero-shot metrics. This brings architecture-level information into the trained predictor through the training process. 3) We provide comprehensive ablation studies and experimental results to demonstrate the effectiveness of GATES++. 4) We provide experiments on a new benchmark search space, NAS-Bench-301 [15].

2 Background and Related Work

2.1 Neural Architecture Search

In recent years, neural architecture search (NAS) has been proposed as an automatic design tool of NN architectures. A pioneering work, Zoph et al. [5] use a recurrent neural network (RNN) controller to sample architectures, and employ reinforcement learning to learn the controller based on the validation accuracy of these architectures.

Generally speaking, there are three key components in a NAS framework [6], the *architecture search space*, the *architecture searching module* and the *architecture evaluation module*. Specifically, the architecture search space designates the architectural decisions to make, and the architecture searching module conducts exploration in the search space to sample architectures for evaluation. And the architecture evaluation module gives out the evaluation results of sampled architectures, e.g., accuracy, latency, etc., and feeds them back to the architecture searching module.

The vanilla NAS algorithm [5] is prohibitively costly, and there are two directions to alleviate the huge computational burden of NAS, which focus on improving the searching and evaluation module, respectively. 1) Evaluation: accelerating the evaluation of each individual architecture [16], [17], [18], [19]; 2) Searching: increasing the sample efficiency so that fewer architectures needed to be evaluated for discovering a good architecture [7], [8], [9], [10].

2.2 Architecture Evaluation Module

One commonly used technique to accelerate architecture evaluation is parameter sharing [17], [18], [19], where a super-net is constructed such that all architectures in the search space share a superset of weights. In this way, the training costs of architectures are amortized to an "one-shot" super-net training. Parameter sharing dramatically reduces the computational burden and is widely used by recent methods. However, recent studies [20], [21], [22] reveal that the ranking of architecture candidates with parameter sharing might fail to reflect their true rankings, which dramatically affects the effectiveness of the NAS algorithm.

Moreover, the parameter sharing technique is not generally applicable, since it is difficult to construct the super-net for some search spaces, for example, in NAS-Bench-101 [23], the output dimension of one operation can vary across candidate architectures. Due to these limitations, this work does not use parameter sharing, and focuses on improving the sample efficiency of the architecture searching module.

2.3 Architecture Searching Module

To improve the sample efficiency of the architecture search module, a variety of search strategies have been used, e.g., RL-based methods [5], [18], Evolutionary methods [7], [24], Monte Carlo Tree Search (MCTS) method [25], etc.

A promising direction to improve the sample efficiency of NAS is to utilize a performance predictor to sample new architectures, a.k.a. *predictor-based NAS*. An early study [8] trains a surrogate model (predictor) to identify promising architectures with increasing complexity. NASBot [26] design a distance metric in the architecture space and exploits Gaussian Process to get the posterior of the architecture performances. Then, it samples new architectures based on the acquisition function calculated using the posterior. NAO [9] trains an LSTM-based autoencoder together with a performance predictor based on the latent representation. After updating the latent representation following the predictor's gradients, NAO decodes the latent representation to sample new architectures. BRP-NAS [27] uses a GCN to predict the hardware latencies as well as the architecture accuracies. Arch-Graph [28] jointly encodes the task embedding and the architecture pair using a GCN-based predictor to support task-transferable search.

2.4 Neural Architecture Encoders

Existing neural architecture encoding schemes include sequence-based and graph-based ones. In the sequence based scheme, the neural architecture is *flattened* into a string encoding the architecture decisions, then encoded using either an LSTM [8], [9], [10] or a Multi-Layer Perceptron (MLP) [8], [10]. In these methods, the topological information could only be modeled implicitly, which deteriorates the encoder's representational ability. Also, the search efficiency would deteriorate since these encoders could not guarantee to map isomorphic architectures [23], [29] to the same representation, and data augmentation and regularization tricks are utilized to alleviate this issue [9].

In this paper, we classify topological search spaces into two types, the "operation on node" (OON) search spaces, and the "operation on edge" (OOE) search spaces. The operations (e.g., `Conv3x3`) in the OON search spaces are on the nodes of the DAG. Some representatives include NAS-Bench-101 [23], and the randomly wired search space [30]. While in the OOE search spaces, the operations are on the edges of DAG. Representatives include NAS-Bench-201 [31], NAS-Bench-301 [15], ENAS [18] and so on. Figure 2 gives an illustration of the OON and OOE search spaces.

Recently, the graph-based encoding scheme that utilizes the topological information explicitly has been used to get better performance. In these graph-based schemes, graph convolutional networks (GCN) [13] are usually used to embed the graphs to fixed-length vector representations. For the "operation on node" search spaces, in which the operations (e.g., `Conv3x3`) are on the nodes of the DAG, GCN can be directly applied [12] to encode architectures, i.e., using adjacency matrix and operation embedding of each node as the input. However, for the "operation on edge" search spaces, in which the operations are on the edges, GCN cannot be applied directly. Zhang et al. [32] proposes an asynchronous message passing scheme to encode DAGs, and conducts NAS-related experiments on the ENAS search space, an OOE search space. The design intuition of this scheme is similar to ours. Another study [11] proposes an ad-hoc solution for the ENAS search space. They represent each node by the concatenation of the operation embeddings on the input edges. This solution is contrived and cannot generalize to search spaces where nodes have different input degrees. Moreover, since concatenations are not commutative, this encoding scheme could not handle isomorphic architectures correctly. In brief, existing graph-based encoding schemes are specific to different search spaces, and a generic approach for encoding the neural architectures is desirable in the literature.

Following the encoder design (GATES) in the preliminary version of this work [14], Chen et al. [33] consider the unequal contribution of the operations and use a set of weighting coefficients to aggregate information flow from different operations.

2.5 Zero-Shot Metrics of Architectures

Operation-Level Metrics NN pruning is a large research field that aims at pruning out redundant parameters from a neural network. Most pruning methods [34], [35], [36] require training before or along with pruning. While a special type of pruning-at-initialization methods [37], [38], [39], [40] propose various types of per-parameter saliency metrics, and utilize these metrics to prune out less-salient parameters without any training. Recently, Zhang et al. [41] have incorporated these operation-level metrics into DARTS to pick operations.

Architecture-Level Metrics Recently, Abdelfattah et al. [42] adapt zero-cost (i.e., zero-shot) saliency metrics from the pruning literature to score entire neural architectures. Specifically, they sum up the per-parameter saliency metrics of all parameters as the architecture score. There exist other studies that regard the neural network as a function mapping instead of a bulk of parameters, and extract certain properties of this function mapping as the architecture score. For example, Lin et al. [43] define the expected Gaussian complexity to measure the network expressivity. Mellor et al. [44] study the linear maps of NN that are identified by a binary code corresponding to the activation pattern of the rectified linear units (ReLUs). They define a kernel using the Hamming distance between these binary codes and use the log determinant of the kernel as the architecture score. A recent work [21] conducts a comparative study on these zero-shot metrics, and reveals that directly using them to rank architectures results in prominently biased ranking.

In this paper, rather than directly adopting architecture-level zero-shot metrics for ranking, we incorporate them into the construction and learning of architecture performance predictors. These operation-level and architecture-level zero-shot metrics contain information about NNs' concrete computing semantics, which all existing architecture performance predictors are unaware of. Incorporating this information can improve the predictors' ranking quality.

3 Predictor-Based NAS

The principle of predictor-based NAS is to increase the sample efficiency of the NAS process, by utilizing an approximated performance predictor to sample architectures that are more worth evaluating. We summarize the flow of predictor-based NAS as in Alg. 1 and Fig. 1. A recent work [45] presents a formulation of the predictor-based NAS that help justify the rationality of this widely-used flow.

In line 6 of Alg. 1, the architecture candidates are sampled based on the approximated evaluation of the predictor. Utilizing a more accurate predictor, we could choose better architectures for further evaluation. The better the generalization ability of the predictor is, the fewer architectures are needed to be exactly evaluated to get a highly accurate predictor. Therefore, the generalization ability of the predictor is crucial for the efficiency and effectiveness of NAS.

The model design (i.e., how to encode the neural architectures) of the predictor is crucial to its generalization ability. We'll introduce our main effort to improve the predictor from the "model design" aspect in the following section.

4 GATES: A Generic Neural Architecture Encoder

4.1 The Encoding Scheme

A performance predictor P is a model that takes a neural architecture a as input, and outputs a predicted score $\hat{s}$. Usually, the performance predictor is constructed by an encoder followed by an MLP, as shown in Eq. 1.

$$\hat{s} = \mathrm{P}(a) = \mathrm{MLP}(\mathrm{Enc}(a)). \tag{1}$$

The encoder Enc maps a neural architecture into a continuous embedding space, and its design is vital to the generalization ability of the performance predictor. Existing encoders include the sequence-based ones (e.g., MLP, LSTM) and the graph-based ones (e.g., GCN). We design a new graph-based neural architecture encoder GATES that is more suitable for modeling neural architectures.

To encode a cell architecture into an embedding vector, GATES follows the ideology of modeling the information flow in the architecture, and uses the output information as the embedding of the architecture. The notations are summarized in Table 1.

Specifically, we model the input information as the embedding of the input nodes $E \in \mathbb{R}^{n_i \times h_i}$, where n_i is the number of input nodes, and h_i is the embedding size of the information. The information (embedding of the input nodes) is then "processed" by the operations and "propagates" along the DAG.

The encoding process of GATES goes as follows: Upon each unary operation o (e.g., `Conv3x3`, `MaxPool`, etc.), the input information x_{in} of this operation is processed by a linear transform W_x and then elementwise multiplied with a soft attention mask $m = \sigma(\mathrm{EMB}(o)W_o) \in \mathbb{R}^{1\times h_i}$.

$$x_{\text{out}} = m \odot x_{\text{in}} W_x, \tag{2}$$

where $\odot$ denotes the elementwise multiplication. And the mask m is calculated from the operation embedding $\mathrm{EMB}(o) = \mathrm{onehot}(o)^T \mathrm{EMB} \in \mathbb{R}^{1\times h_o}$.

Algorithm 1 The flow of predictor-based neural architecture search.

```
1: $\mathcal{A}$: Architecture search space
2: P : $\mathcal{A} \to \mathbb{R}$: Performance predictor that outputs the predicted performance given the architecture
3: $N^{(k)}$: Number of architectures to sample in the $k$-th iteration

4: k = 1
5: while $k \le$ MAX_ITER do
6:     Sample a subset of architectures $S^{(k)} = \{a_j^{(k)}\}_{j=1,\cdots,N^{(k)}}$ from $\mathcal{A}$, utilizing P
7:     Evaluate architectures in $S^{(k)}$, get $\tilde{S}^{(k)} = \{(a_j^{(k)}, y_j^{(k)})\}_{j=1,\cdots,N^{(k)}}$ ($y$ is the performance)
8:     Optimizing P using the ground-truth architecture evaluation data $\tilde{S} = \cup_{i=1}^{k} \tilde{S}^{(i)}$
9: end while
10: Output $a_{j^*} \in \cup_{i=1}^{k} S^{(i)}$ with best corresponding $y_{j^*}$; Or, $a^* = \mathrm{argmax}_{a\in\mathcal{A}} \mathrm{P}(a)$
```

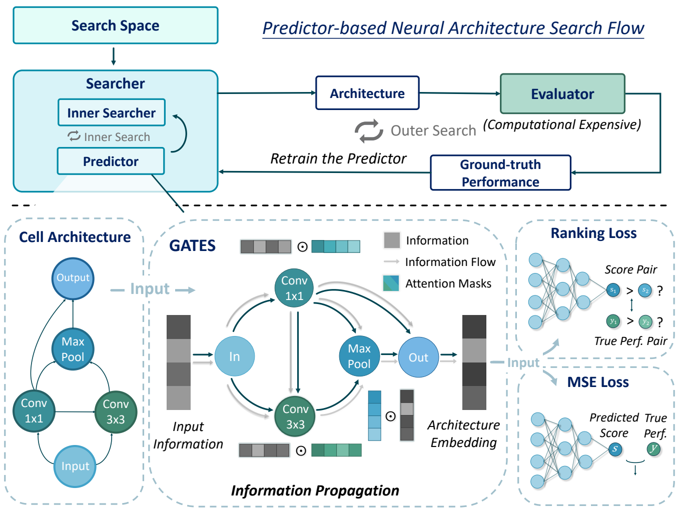
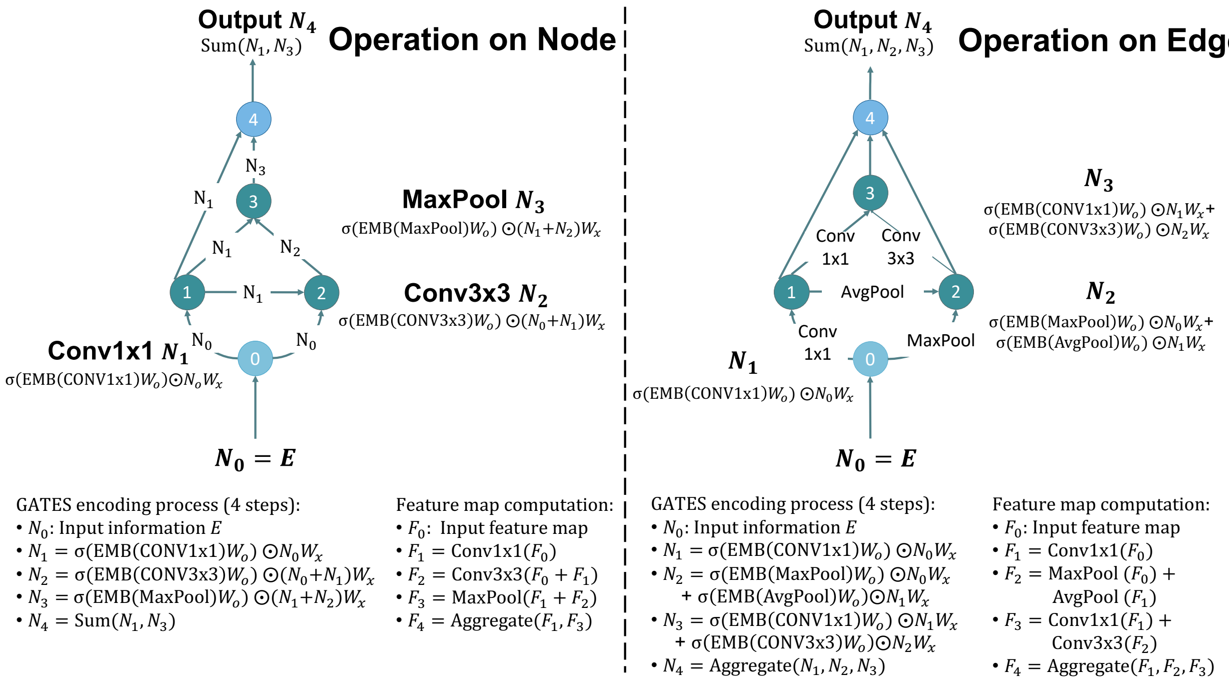


Fig. 1. The overview of GATES. Upper: The general flow of the predictor-based NAS. Lower: Illustration of GATES' encoding processes of an OON cell architecture and the usage of ranking losses.

Multiple pieces of information are aggregated at each node using summation. Finally, after obtaining the virtual information at all the nodes, the information at the output node is used as the embedding of the entire cell architecture. For search spaces with multiple cells (e.g., normal and reduce cells in ENAS), GATES encodes each cell independently, and concatenates the embeddings of cells as the embedding of the architecture.

Fig. 2 illustrates two examples of the encoding process in the OON and OOE search spaces. As can be seen, the encoding process of GATES mimics the actual feature map computation. For example, in the example of the OON search space, the actual feature map computation at node 2 is $F_2 = \mathrm{Conv3x3}(F_0 + F_1)$, where F_i is the feature map at node i. To model the information processing of this feature map computation, GATES calculates the information (node embedding) at node 2 by $N_2 = \sigma(\mathrm{EMB}(\mathrm{Conv3x3})W_o) \odot (N_0 + N_1)W_x$, where $\sigma(\cdot)$ is the sigmoid function, and


Output N_4
Sum(N_1, N_3)
Operation on Node
MaxPool N_3
$\sigma(\text{EMB(MaxPool)}W_o) \odot (N_1+N_2)W_x$
Conv3x3 N_2
$\sigma(\text{EMB(CONV3x3)}W_o) \odot (N_0+N_1)W_x$
Conv1x1 N_1
$\sigma(\text{EMB(CONV1x1)}W_o) \odot N_o W_x$
$N_0 = E$
Output N_4
Sum(N_1, N_2, N_3)
Operation on Edge
Conv 1x1
Conv 3x3
AvgPool
MaxPool
N_3
$\sigma(\text{EMB(CONV1x1)}W_o) \odot N_1 W_x +$
$\sigma(\text{EMB(CONV3x3)}W_o) \odot N_2 W_x$
N_2
$\sigma(\text{EMB(MaxPool)}W_o) \odot N_0 W_x +$
$\sigma(\text{EMB(AvgPool)}W_o) \odot N_1 W_x$
N_1
$\sigma(\text{EMB(CONV1x1)}W_o) \odot N_0 W_x$
$N_0 = E$

GATES encoding process (4 steps):
- N_0: Input information E
- $N_1 = \sigma(\text{EMB(CONV1x1)}W_o) \odot N_0 W_x$
- $N_2 = \sigma(\text{EMB(CONV3x3)}W_o) \odot (N_0+N_1)W_x$
- $N_3 = \sigma(\text{EMB(MaxPool)}W_o) \odot (N_1+N_2)W_x$
- $N_4 = \text{Sum}(N_1, N_3)$

Feature map computation:
- F_0: Input feature map
- $F_1 = \text{Conv1x1}(F_0)$
- $F_2 = \text{Conv3x3}(F_0 + F_1)$
- $F_3 = \text{MaxPool}(F_1 + F_2)$
- $F_4 = \text{Aggregate}(F_1, F_3)$

GATES encoding process (4 steps):
- N_0: Input information E
- $N_1 = \sigma(\text{EMB(CONV1x1)}W_o) \odot N_0 W_x$
- $N_2 = \sigma(\text{EMB(MaxPool)}W_o) \odot N_0 W_x$ $+ \sigma(\text{EMB(AvgPool)}W_o) \odot N_1 W_x$
- $N_3 = \sigma(\text{EMB(CONV1x1)}W_o) \odot N_1 W_x$ $+ \sigma(\text{EMB(CONV3x3)}W_o) \odot N_2 W_x$
- $N_4 = \text{Aggregate}(N_1, N_2, N_3)$

Feature map computation:
- F_0: Input feature map
- $F_1 = \text{Conv1x1}(F_0)$
- $F_2 = \text{MaxPool}(F_0) + \text{AvgPool}(F_1)$
- $F_3 = \text{Conv1x1}(F_1) + \text{Conv3x3}(F_2)$
- $F_4 = \text{Aggregate}(F_1, F_2, F_3)$

Fig. 2. Feature map (F_i) computation and GATES encoding process (N_i). Left: The "operation on node" cell search space, where operations (e.g., `Conv3x3`) are on the nodes of the DAG (e.g., NAS-Bench-101 [23], randomly wired search space [30]). Right: The "operation on edge" cell search space, where operations are on the edges of the DAG. (e.g., NAS-Bench-201 [31], NAS-Bench-301 [15], ENAS [18]).

TABLE 1
Notations to illustrate the basic methodology of GATES. E, EMB, W_o and W_x are all trainable parameters.

n_i	number of input nodes: 1, 1, 2 for NAS-Bench-101, NAS-Bench-201 and ENAS, respectively
N_o	number of operation primitives
h_o	embedding size of operation
h_i	embedding size of information
$E \in \mathbb{R}^{n_i \times h_i}$	the embedding of the information at the input nodes
$\text{EMB} \in \mathbb{R}^{N_o \times h_o}$	the operation embeddings
$W_o \in \mathbb{R}^{h_o \times h_i}$	the transformation matrix on the operation embedding
$W_x \in \mathbb{R}^{h_i \times h_i}$	the transformation matrix on the information

$W_o \in \mathbb{R}^{h_o \times h_i}$ is a transformation matrix that transforms the h_o-dim operation embedding into a h_i-dim feature. That is to say, the summation of feature maps $F_0 + F_1$ corresponds to the summation of the virtual information $N_0 + N_1$, and the data processing function $o(\cdot)$ (`Conv3x3`) corresponds to a transform $f(\cdot)$ that processes the information $x = N_0 + N_1$ by $f_o(x) = \sigma(\text{EMB}(o)W_o) \odot xW_x$.

Intuitively, to encode a cell architecture, GATES models the operations in the architecture as the "soft gates" that control the flow of virtual information, and the output information is used as the embedding of the cell architecture. The key difference between GATES and GCN is: In GATES, the operations (e.g., `Conv3x3`) are modeled as the processing of the node attributes (i.e., virtual information), whereas GCN models them as the node attributes themselves.

To summarize, the representational power of GATES for neural architectures comes from two aspects: 1) The more reasonable modeling of operations in data-processing DAGs. 2) The intrinsic proper handling of DAG isomorphism. The following section provides discussions on GATES's ability to handle isomorphic architectures.

4.2 Discussion on Isomorphism Handling

4.2.1 GATES correctly maps ismorphic architectures

The encoding process of GATES mimics the actual computation flow: GATES uses multiplicative transforms to mimic the forward process of operations (e.g., `Conv3x3`), and uses commutative aggregation to mimic actual commutative aggregation of the feature maps. Naturally, GATES would encode two architectures that give out the same feature map results into the same representation. That is to say, the embedding space of GATES is more meaningful. However, GATES might fail to map non-isomorphic architectures to different representations. And we leave it to future work to further increase the discriminative power of GATES.

In the search spaces which we have experimented with (i.e., NAS-Bench-101, NAS-Bench-201, and ENAS), the combination of feature maps at internal nodes is done via

TABLE 2
Kendall's Tau on 1) NAS-Bench-101 test set 2) the 7-vertex subset of test set 3) all isomorphic counterparts of the 7-vertex subset (w.o. de-duplication). The last column shows the sum of variances of predicted scores in every isomorphic architecture group. All predictors are trained using the hinge pairwise ranking loss on 0.1% training data.

Encoders	test set (42362)	7-vertex test set (36064)	7-vertex test set w.o. de-dup. (116102)	
	τ	τ	τ	Total Var.
MLP [10]	0.5272	0.5143	0.4729	43.58
LSTM [10]	0.5993	0.5877	0.5656	18.80
GCN [12]	0.5790	0.5876	0.6169	1.16E-11
GATES	**0.7789**	**0.7724**	**0.7758**	9.24E-12

addition operation, which is commutative. Therefore, for encoding the architecture, GATES also uses commutative addition to combine the "virtual information". Note that if the feature map aggregation at some internal node is not commutative (e.g., concatenation), we should use a non-commutative aggregation of the virtual information too.

We conduct a simple experiment to verify GATES's ability to map isomorphic architectures to the same representation on NAS-Bench-101. After splitting the train and test sets, there are 36064, 6037, 256, 5 testing architectures with 7, 6, 5, 4 vertices, and 323018, 55973, 2185, 79, 6, 1 training architectures with 7 6, 5, 4, 3, 2 vertices respectively. Since all isomorphic architectures are already removed in NAS-Bench-101, we generate the isomorphic architectures for the 36064 unique testing architectures with 7 vertices, and get 116102 architectures. Among the 36064 architectures, there are 20994 architectures that have isomorphic counterparts. Table 2 shows the test results of different predictors trained with 0.1% training samples on these 116k architectures. Since sequence-based encoding schemes cannot map isomorphic architectures to the same representation, the ranking correlation decreases if no de-duplication procedure is carried out. The last column shows the sum of variance of predicted scores in every isomorphic architecture group. We can see that GATES and GCN can map isomorphic architectures to the same representation (a variance of 0 with negligible numeric errors), since only isomorphism-invariant aggregation is used in the encoding process.

4.2.2 Two counter-examples of an ad-hoc GCN encoder

Since GCN cannot be directly applied to encoding architectures from the OOE search spaces, a recent study [11] proposes an ad-hoc solution for the ENAS search space. They represent each node by the concatenation of the operation embeddings on the input edges. This solution cannot generalize to search spaces where nodes could have different input degrees. What's more, since the concatenation operation is not commutative, this encoding scheme could not map isomorphic architectures to the same representation correctly. Fig. 3 illustrates two minimal counter-examples.

4.3 Implementation of GATES

In practice, to calculate the information propagation following the topological order of different graphs in a batched manner, we use a stack of GATES layers. In the forward process of each GATES layer, one step of information propagation is taken place at every node. That is to say, if a graph is fed into a GATES encoder with N layers, the information is propagated and aggregated for N steps along the graph.

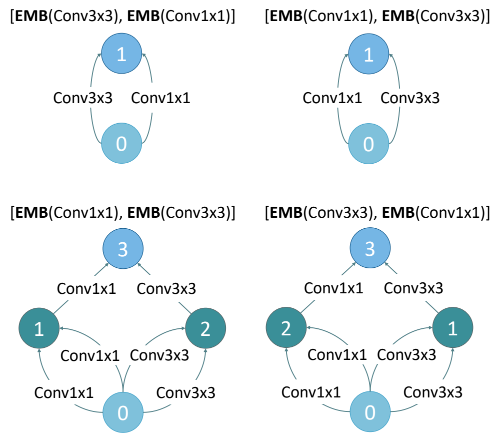

[EMB(Conv3x3), EMB(Conv1x1)]
[EMB(Conv1x1), EMB(Conv3x3)]
[EMB(Conv1x1), EMB(Conv3x3)]
[EMB(Conv3x3), EMB(Conv1x1)]
Conv3x3
Conv1x1
0
1
2
3

Fig. 3. An ad-hoc graph-based solution [11] for encoding the architecture (in an OOE search space) fails to map isomorphic architectures to the same representation. In the upper case, the two architectures are the same graph, but the embeddings of Node 1 differ. In the lower case, these two architecture are isomorphic, since the feature map aggregation at Node 3 is a commutative element-wise addition. However, this encoding scheme cannot guarantee to map these two architectures to the same representation, since the original node embeddings already differ at Node 3. The failure to handle the isomorphism is because that the non-commutative characteristics of the concatenation operation causes the encoding process to be not permutation-invariant to node and edge order.

The detailed formulas and implementations of one GATES layer for OON and OOE search spaces are described as follows, and the notations are summarized in Table 3.

4.3.1 Operation On Node (OON) Search Space

For the OON case, we take the NAS-Bench-101 search space as an example. In the cell architecture, there is $n_i = 1$ input node, and at most $V = 7$ nodes. For batch computation, we pad zero columns and rows into the adjacent matrix to ensure that all adjacent matrices are of size 7×7, and add "none" operations into the corresponding positions in the operation list. The calculation of the k-th GATES layer is

$$\begin{aligned} X^{(0)} &= \text{CONCAT}(\tilde{E}, \mathbf{0}_{b \times V - n_i \times h_i^{(0)}}, \text{dim=1}), \\ X^{(k)} &= \sigma(\text{EMB}(o) W_o^{(k)}) \odot (A X^{(k-1)} W_x^{(k)}), \end{aligned} \tag{3}$$

where $\tilde{E} = \text{repeat}(E, [b, 1, 1]) \in \mathbb{R}^{b \times n_i \times h_i^{(0)}}$, and $E, \text{EMB}, W_o^{(k)}, W_x^{(k)}$ are trainable parameters.

In practice, we found that for the OON search space, adding a self-loop of the information propagation would lead to slightly better performance.

$$\begin{aligned} X^{(k)} &= \sigma(\text{EMB}(o) W_o^{(k)}) \odot (\tilde{A} X^{(k-1)} W_x^{(k)}), \\ \tilde{A} &= A + I. \end{aligned} \tag{4}$$

TABLE 3
Notations used in the implementation of GATES with support for batched computation.

V	maximum number of nodes: 7, 4, 6 for NAS-Bench-101 [23], NAS-Bench-201 [31] and ENAS [18], respectively
n_i	number of input nodes: 1, 1, 2 for NAS-Bench-101, NAS-Bench-201 and ENAS, respectively
N_o	number of operation primitives
h_o	embedding size of operation
$h_i^{(k)}$	embedding size of information in the k-th layer
$E \in \mathbb{R}^{n_i \times h_i^{(0)}}$	the embedding of the information at the input nodes
$\text{EMB} \in \mathbb{R}^{N_o \times h_o}$	the operation embeddings
$W_o^{(k)} \in \mathbb{R}^{h_o \times h_i^{(k)}}$	the transformation matrix on the operation embedding (the k-th layer)
$W_x^{(k)} \in \mathbb{R}^{h_i^{(k-1)} \times h_i^{(k)}}$	the transformation matrix on previous layer's output information (the k-th layer)
b	batch size
$A \in \mathbb{R}^{b \times V \times V}$	adjacency matrix
$X^{(k)} \in \mathbb{R}^{b \times V \times h_i^{(k)}}$	the output virtual information of the k-th layer
$\text{EMB}(o) \in \mathbb{R}^{b \times V \times h_o}$	(NAS-Bench-101) the embeddings of the operations on nodes
$\text{EMB}(o) \in \mathbb{R}^{b \times V \times V \times h_o}$	(NAS-Bench-201) the embeddings of the operations on edges
n_d	(ENAS) maximum input degree of nodes
$\text{EMB}(o_d) \in \mathbb{R}^{b \times V \times V \times h_o}$	(ENAS) the embeddings of operations on the d-th input edge for nodes

4.3.2 *Operation On Edge (OOE) Search Space*

For the OOE search spaces, the calculation of a GATES layer could be written as

$$\begin{aligned} X^{(0)} &= \text{CONCAT}(\tilde{E}, \mathbf{0}_{b \times V - n_i \times h_i^{(0)}}, \text{dim=1}), \\ S &= \text{EXPAND}(X^{(k-1)} W_x^{(k)}, 1), \\ X^{(k)} &= \text{SUM}(\sum_{d=1}^{n_d} \text{EXPAND}(A, 3) \odot \sigma(\text{EMB}(o_d) W_o^{(k)}) \odot S, \\ &\qquad \text{dim=2}), \end{aligned} \tag{5}$$

where $\tilde{E} = \text{repeat}(E, [b, 1, 1]) \in \mathbb{R}^{b \times n_i \times h_i^{(0)}}$, and $\text{EXPAND}(A, \text{dim})$ denotes the operation to insert a new dimension as the "dim"-th dimension.

For the search spaces where there is at most one edge between each pair of nodes (e.g., NAS-Bench-201), the above calculation could be simplified to

$$\begin{aligned} X^{(0)} &= \text{CONCAT}(\tilde{E}, \mathbf{0}_{b \times V - n_i \times h_i^{(0)}}, \text{dim=1}), \\ S &= \text{EXPAND}(X^{(k-1)} W_x^{(k)}, 1), \\ X^{(k)} &= \text{SUM}(\text{EXPAND}(A, 3) \odot \sigma(\text{EMB}(o) W_o^{(k)}) \odot S, \\ &\qquad \text{dim=2}), \end{aligned} \tag{6}$$

4.4 The Training Scheme

The most common practice [8], [9] to train the predictors is to minimize the Mean Squared Error (MSE) between the predicted scores and the true performances.

$$L(\{a_j, y_j\}_{j=1,\cdots,N}) = \sum_{j=1}^{N} (\text{P}(a_j) - y_j)^2, \tag{7}$$

where N denotes the number of training architectures, a_j denotes one architecture, and y_j denotes the true performance of a_j.

Nevertheless, in NAS applications, the relative ranking order of architectures is more important in guiding the search process rather than the absolute performance values. And since ranking losses are better surrogate losses [46], [47] for the ranking quality than the regression loss, we propose to employ ranking losses to train the predictors.[1] Moreover, using ranking losses also open up possibilities for utilizing supervisory signals of different fidelities to train a predictor. For example, a work [49] following our conference version uses ranking losses to leverage supervisory signals from different training epochs.

We utilize different pairwise and listwise ranking losses for training the predictor [50], [51], [52]. The pairwise ranking loss could be written as

$$L^p(\tilde{S}) = \sum_{i=1}^{N} \sum_{j \in \{j | y_i < y_j\}} \phi(P(a_j), P(a_i)). \tag{8}$$

We experiment with two different choices of ϕ. 1) The binary cross entropy function $\phi(s_j, s_i) = \log(1 + e^{(s_j - s_i)})$; 2) The hinge loss function $\phi(s_j, s_i) = \max(0, m - (s_j - s_i))$, where m is a positive margin.

We also experiment with a pairwise comparator: We construct an MLP that takes the concatenation of two architecture embeddings as input and outputs a score: $s = \text{MLP}([E(a_j), E(a_i)]$, and a positive s indicates that a_j is better than a_i. Note that the total-orderness of the architectures is not guaranteed using this comparator. So, we add a simple anti-symmetry regularization term in the training of the comparator. The loss for training the comparator is:

$$\begin{aligned} L^p(\tilde{S}) = &\sum_{i=1}^{N} \sum_{j \in \{j | y_i < y_j\}} \max(0, m - \text{MLP}([E(a_j), E(a_i)]) \\ &+ \max(0, m + \text{MLP}([E(a_i), E(a_j)])). \end{aligned} \tag{9}$$

1. A concurrent work [48] with our conference version also proposes to use ranking loss to train the predictor.

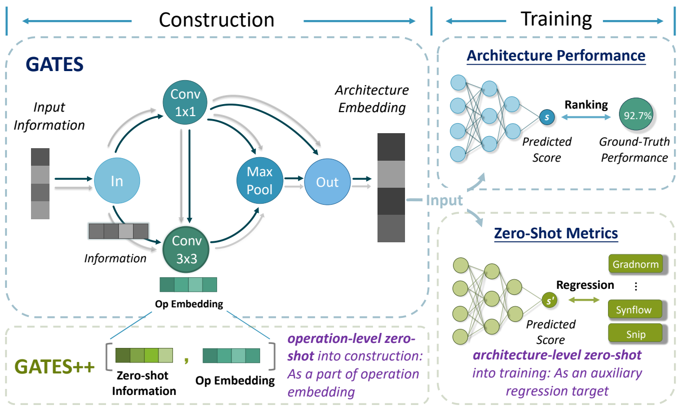


Fig. 4. The overview of GATES++.

We also use a listwise ranking loss, ListMLE [52]:

$$L^l(\tilde{S}) = \sum_{U \subset \tilde{S}} \sum_{i=1}^{|U|} \{-P(a^{(i),U}) + \log \sum_{j=i}^{|U|} \exp(P(a^{(j),U}))\}, \tag{10}$$

where U are subsets of $\tilde{S}$, $|U|$ denotes the size of U, $a^{(i),U}$ denotes the architecture whose true performance $y^{(i),U}$ is the i-th best in the subset U.

5 GATES++: Incorporating Multifaceted Architectural Information into GATES

GATES is reasonable modeling of data-processing DAGs, since it models and learns the **virtual information processing** of data-processing operations. However, its construction and training are completely unaware of the **concrete computing semantics** of NNs. At the operation level, the GATES construction initializes the embeddings of different operations indistinguishably, and is unaware of which operations are parameterized, and which two operations should be closer. At the architecture level, GATES is only trained with the pairs of descriptions and ground-truth performances of architectures, without other relatively low-level information related to the actual computation of these NN architectures.

We propose GATES++, an enhanced encoder for NN DAGs that incorporates multifaceted architectural information about NN's operation-level and architecture-level computing semantics into its construction and training, respectively. Sec. 5.1 describes how we incorporate the operation-level information, which provides prior knowledge to the predictor. And Sec. 5.2 describes how we incorporate the architecture-level information, which adds additional observation information to better train the predictor. These two techniques of GATES++ are illustrated in Fig. 4.

5.1 The Encoding Scheme

As we have introduced in Sec. 2.5, the NN pruning literatures have proposed various per-parameter saliency metrics. We use four types of zero-shot per-parameter saliency metrics, *grad_norm*, *plain* [39], *snip* [37], *synflow* [40], which are computed with the gradients obtained through a single forward and backward pass of the actual NN architecture, and two complexity metrics *params*, *flops*. The four saliency metrics are computed as follows.

$$\begin{aligned}
&grad\_norm: \mathcal{S}(\theta) = |\frac{\partial \mathcal{L}}{\partial \theta}|_2, \\
&plain: \mathcal{S}(\theta) = \frac{\partial \mathcal{L}}{\partial \theta} \odot \theta, \\
&snip: \mathcal{S}(\theta) = |\frac{\partial \mathcal{L}}{\partial \theta} \odot \theta|_1, \\
&synflow: \mathcal{S}(\theta) = \frac{\partial \mathcal{R}}{\partial \theta} \odot \theta; \mathcal{R} = \mathbb{1}^T (\prod_{\theta_i \in \theta} |\theta_i|)\mathbb{1},
\end{aligned} \tag{11}$$

where $\mathcal{L}$ denotes the loss function, and θ denotes the parameters, and $\odot$ denotes the Hadamard product.

GATES++ encodes an architecture α as follows. We firstly random initialize a candidate model with architecture α, and conduct a single forward and backward pass on a data batch. Then we can compute six parameter-level metrics for all parameters. Note that as we are encoding architecture descriptions from cell-based search spaces, an operation in the cell architecture description stands for multiple parameters in the candidate model. For example, on NAS-Bench-201, each operation in the cell architecture description corresponds to 15 actual NN operations, since the cell architecture is stacked 15 times to construct a candidate model. Therefore, for an operation o in the cell architecture description, we calculate the average of metrics of its corresponding parameters in the candidate model, and get a 6-dim vector $z^o(o) \in \mathbb{R}^6$ (the superscript o denotes "operation-level"), with each dimension standing for one metric type. For nonparametrized operations such as MaxPool, we just set the 6-dim vector to $\vec{0}$. Finally, we concatenate $z^o(o)$ onto the trainable operation embedding of the operation EMB(o). That is to say, for an operation o in the architecture description, GATES++ calculate its attention mask as $\sigma(\text{CONCAT}[\text{EMB}(o), z^o(o)]W_o)$.

Jumping out of the details, these zero-shot metrics provides prior knowledge regarding NN's actual operation-level computing semantics. For example, almost all types of zero-shot metrics can reflect that `Conv3x3` is more similar to `Conv5x5` than `MaxPool`. To give some quantitive evidence for this claim, we provide some analyses in the appendix. This prior knowledge helps GATES++ model NN architectures better, especially when there are only a small number of training architectures.

5.2 The Training Scheme

Apart from incorporating operation-level information into the construction of GATES++, we would also like to incorporate more architecture-level information into the training of GATES++. We use five types of architecture-level zero-shot metrics, including *relu_logdet* [44], and four other architecture-level metrics [42] summing up per-parameter saliency metrics (i.e., *grad_norm*, *plain*, *snip*, *synflow*).

Among these zero-shot metrics, the most special one is *relu_logdet* [44], which is a zero-shot measure of the architecture discriminability. Instead of simply aggregating per-parameter gradients, *relu_logdet* measures the activation differences at all ReLU layers between different input images:

$$s = \log ||K_H||$$
$$K_H = \begin{pmatrix} N_A - d_H(c_1, c_1) & \cdots & N_A - d_H(c_1, c_N) \\ \vdots & \ddots & \vdots \\ N_A - d_H(c_N, c_1) & \cdots & N_A - d_H(c_N, c_N) \end{pmatrix}, \tag{12}$$

where c_i is a binary mask indicating whether each feature value is larger than 0 at all ReLU layers for input data i. And $d_H(c_i, c_j)$ is the Hamming distance between the binarized activation code of the i-th and the j-th data, and N_A is the number of rectified linear units.

To train GATES++, we add an auxiliary regression head to fit these architecture-level zero-shot metrics. The regression target is a 5-dim vector z^a (the superscript a denotes "architecture-level"). And an auxiliary loss term is added onto the loss function introduced in Sec. 4.4:

$$\mathcal{L}_{\text{aux}} = \sum_{j=1}^{N} (\text{MLP}_{\text{reg}}(\text{Enc}(\alpha_j)) - z_j^a)^2, \tag{13}$$

where MLP_{reg} is an auxiliary MLP head with an output dimension of 5, and Enc is the shared GATES++ encoder.

From a high-level viewpoint, these metrics provide multifaceted architecture-level information of NN, and learning to fit this information could encourage the encoder to learn better architecture representation. Note that we use the regression loss for the auxiliary head instead of the ranking loss. This is because we neither use the output of the auxiliary head to rank architecture nor seek to provide the correct ranking of these zero-shot metrics. Thus it is not necessary to use the ranking loss for the auxiliary head. Another minor consideration is that more architecture-level information can be brought into the encoder by the regression objective than by a ranking objective, as the ranking objective only preserves the ranking information and discards the absolute values and differences.

6 Experiments

This section is organized as follows. Firstly, Sec. 6.1 describes the evaluation benchmarks and criteria. Sec. 6.2 gives out experimental setup regarding predictor the training process and encoder construction. Then, Sec. 6.3 compares GATES, GATES++, and baseline encoders, including MLP, LSTM, and other GCN encoders. And Sec. 6.4 and Sec. 6.5 give out abalation studies of techniques in GATES and GATES++, respectively. Next, we compare architecture search results with different encoders in Sec. 6.6. Finally, Sec. 6.7 shows the perfomances of discovered architecture performances on CIFAR-10 and ImageNet. Code is available at https://github.com/walkerning/aw_nas.

6.1 Benchmarks and Criteria

Evaluation Benchmarks Experiments in Sec. 6.3, Sec. 6.4 and Sec. 6.6 are carried out on benchmark search spaces, including an OON search space, NAS-Bench-101 [23], and one of its sub search space, NAS-Bench-1shot-3 [53], and two OOE search spaces, NAS-Bench-201 [31] and NAS-Bench-301 [15]. We introduce these benchmarks and the available baseline encoders on them as follows.

- NAS-Bench-101 provides the performances of the 423k unique architectures in an OON search space. When evaluating the ranking qualities of predictors, we use the first 90% (381262) architectures as the training data, and the other 42362 architectures as the testing data. In our experiments, we use the first 50% (7813) as the training data, and the remaining 7812 architectures as the testing data. Sequence-based encoding schemes [10], and graph-based encoding schemes [12] are proposed for encoding architectures on NAS-Bench-101.
- NAS-Bench-1shot-3 is a sub search space of NAS-Bench-101, in which all architectures have 5 internal nodes and the node degrees are designated. We conduct GATES++ experiments on NAS-Bench-1shot-3. NAS-Bench-1shot-3 has about 14k architectures, in which the first 50% (7290) architectures are used as the training data, and the other 7290 are used as the testing data.
- NAS-Bench-201 provides the performances of all the 15625 architectures in an OOE search space. The first 50% (7813) architectures are used as the training data, and the other 7812 architectures are used as the testing data. Since the vanilla GCN encoder cannot be directly applied to OOE search spaces, we adopt a line graph solution [12], [54] for applying GCN to encode OOE architectures following three steps: 1) Convert the graph to a line graph; 2) Apply a vanilla GCN; 3) Concatenate the node embeddings as the graph representation.
- Compared to the previous two benchmarks, NAS-Bench-301 is a surrogate-based benchmark on a much larger OOE search space. It provides the tabular performances of $\sim$60k anchor architectures, and predicts performances of other architectures using these anchor performances. For a fair evaluation of predictors, we only use the tabular performances, in which 5896 architecture performance pairs are used as the training data, and the remaining anchor data are used as the testing set. Sequence-based encoding schemes [9], and graph-based encoding schemes [11] are proposed for encoding architectures on NAS-Bench-301.

Evaluation Criteria We use three ranking measures to evaluate the trained architecture performance predictors. The first one is the commonly used Kendall's Tau ranking correlation [55]. In Kendall's Tau , all discordant pairs are treated equally. However, in NAS applications, the relative rankings among the poorly performing architectures are not of concern. Therefore, we design another two ranking measures that have a more direct correspondence with the NAS flow: 1) N@K: The best true ranking among the top-K architectures selected according to the predicted scores. 2) Precision@K: The proportion of true top-K architectures among the top-K predicted architectures.

6.2 Training and Construction Setup

Predictor Training For each configuration of training ratio (i.e., training set size), we randomly sample 3 different training sets, and train the predictor on each training set with 3

TABLE 4
Kendall's Tau of using different encoders on NAS-Bench-1shot-3, NAS-Bench-201, and NAS-Bench-301 benchmarks.

	Encoder	Proportions of 7290 training samples									
		0.1%	0.25%	0.5%	0.75%	1%	2.5%	5%	10%	50%	100%
NAS-Bench-1shot-3	MLP [10]	0.0772	0.2384	0.3116	0.3413	0.3937	0.4988	0.5318	0.5703	0.6225	0.6307
	LSTM [10]	0.1614	0.3843	0.4588	0.5055	0.5476	0.5826	0.5876	0.6040	0.6196	0.6131
	GCN (global node) [12]	0.1692	0.1983	0.2595	0.3081	0.3668	0.4898	0.5973	0.6927	0.7520	0.7689
	ReNAS [48]	0.1969	0.3589	0.4346	0.4899	0.5131	0.5792	0.6151	0.6250	0.6422	0.6473
	NAOCE [33]	0.1511	0.3635	0.3823	0.5094	0.5542	0.5845	0.5985	0.6083	0.6780	0.7123
	GATES	0.1744	0.4580	**0.6168**	0.6309	0.6557	0.7260	0.7494	0.7742	0.8102	0.8165
	GATES_Large	0.1816	**0.4595**	0.6117	**0.6450**	0.6735	0.7357	0.7448	0.7860	**0.8180**	**0.8221**
	GATES++	**0.2014**	0.4517	0.6001	0.6260	**0.6965**	**0.7365**	**0.7555**	**0.7904**	0.8016	0.8085
	Encoder	Proportions of 7813 training samples									
		0.1%	0.25%	0.5%	0.75%	1%	2.5%	5%	10%	50%	100%
NAS-Bench-201	MLP [10]	0.0162	0.0703	0.0863	0.1613	0.1756	0.2514	0.3885	0.5492	0.8198	0.8728
	LSTM [10]	0.1935	0.4348	0.5079	0.5181	0.5691	0.6025	0.669	0.7395	0.8757	0.9011
	Line-Graph GCN [12]	0.2461	0.2071	0.3113	0.3536	0.4080	0.4797	0.5461	0.6095	0.7733	0.8257
	NAOCE [33]	0.4249	0.4890	0.5040	0.5452	0.5673	0.6565	0.6904	0.7240	0.7774	0.7991
	GATES	0.4309	0.6087	0.6702	0.7328	0.7571	0.8195	0.8583	0.8823	0.9189	0.9258
	GATES_Large	0.3543	0.5852	0.6653	0.7261	0.7546	0.8288	0.8640	0.8824	**0.9192**	**0.9265**
	GATES++	**0.5020**	**0.6284**	**0.7021**	**0.7404**	**0.7729**	**0.8331**	**0.8664**	**0.8848**	0.9116	0.9210
	Encoder	Proportions of 5896 training samples									
		0.1%	0.25%	0.5%	0.75%	1%	2.5%	5%	10%	50%	100%
NAS-Bench-301	MLP [10]	0.1346	0.1901	0.2758	0.3549	0.4018	0.4997	0.5373	0.5687	0.6237	0.6485
	LSTM [9]	**0.3958**	0.4235	0.5161	0.5539	0.5689	0.6513	0.6893	0.7144	0.7531	0.7665
	GCN [11]	0.0336	0.0927	0.0951	0.1313	0.1280	0.2436	0.2673	0.2835	0.3146	0.3242
	NAOCE [33]	0.1257	0.3915	0.4617	0.4928	0.5270	0.6259	0.6959	0.7265	0.7672	0.7866
	GATES	0.1020	0.3144	0.4889	0.5209	0.5590	0.6452	**0.7142**	**0.7430**	0.7695	**0.7887**
	GATES_Large	0.1782	0.4346	0.4966	0.5253	0.5632	0.6664	0.7169	0.7494	0.7685	0.7810
	GATES++	0.2285	**0.4330**	**0.5329**	**0.5713**	**0.5833**	**0.6545**	0.7048	0.7333	**0.7785**	0.7858

random seeds (21, 2021, 202121). The average Kendall's Tau on the testing set across these nine experiments is reported.

In each predictor training experiment, the predictor is trained with an ADAM optimizer with a learning rate 1e-3 and a batch size of 512 for 200 epochs. And the average of testing Kendall's Taus in the last five epochs is recorded.

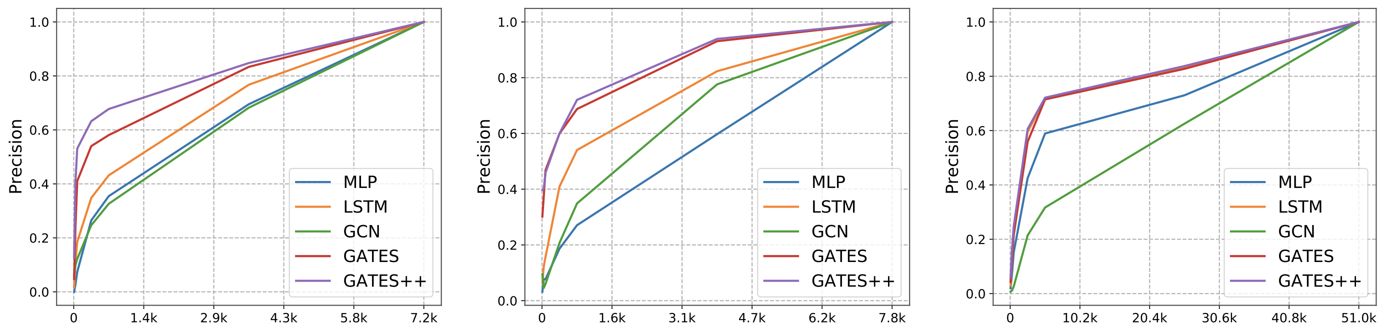
In Sec. 6.4, we find that the performances of different ranking losses are close, and the pairwise hinge loss is a good choice. Therefore, in our experiments, *the pairwise hinge loss with a margin 0.1 is used to train all the predictors unless otherwise stated*.

Encoder Construction On NB101, we follow the serialization method and the model settings in [10] for the construction of the MLP and LSTM encoders. Specifically, the MLP is constructed by 4 fully-connected layers with 512, 2048, 2048, and 512 nodes, and the output of dimension 512 is used as the cell's embedding. The embedding and hidden sizes of the LSTM are both set to 100, and the final hidden state is used as the cell's embedding. For the GCN and GATES encoders, we construct the encoder by stacking five 128-dim GCN or GATES layers. All the embedding sizes are set to 48, including the operation embedding in GCN, and the operation and information embedding in GATES. For GCN, the average of all the nodes' features is used as the cell's embedding. In GCN with global node [12], the feature of the global node is used as the architecture embedding. For ReNAS [48], we reproduce the encoder based on their open-source code, and use the same configuration in the original paper.

On NB201, we use the 6 elements of the lower triangular matrix (excluding the diagonal ones) as the input of sequence-based baselines (MLP and LSTM). Four fully-connected layers with 512, 2048, 2048, 512 nodes are used in the MLP encoder. The embedding and hidden size of the 1-layer LSTM is set to 100, and the final hidden state is used as the architecture embedding. And we use five layers without self-loop for line-graph GCN, GATES, and GATES++.

On NB301, we concatenate the lists of nodes and operations as the input of sequence-based encoders (MLP and LSTM). Specifically, the MLP encoder is constructed with three 128-dim fully-connected layers and the output dim is set to 32. The embedding and hidden size of the 1-layer LSTM are set to 48 and 128, respectively, and the final hidden state is used as the architecture embedding. For GCN and GATES encoders, we construct the encoder by stacking three 32-dim GCN layers and four 64-dim GATES layers, respectively. The average embeddings of all internal nodes are used as the architecture embedding. All the embedding sizes are set to 32, including the operation embedding in GCN, and the operation and information embedding in GATES. We construct the GCN encoder following [11].

Besides, a recent work, NAOCE [33], adopts GATES as the base design, and uses a weighted aggregation of information flow from different operations. Since the authors do



(a) NAS-Bench-1shot-3 (b) NAS-Bench-201 (c) NAS-Bench-301

Fig. 5. Precision@K of different encoders on three benchmarks.

not provide the code, we carefully follow their descriptions in the paper to reproduce the code base on our GATES implementation. The configurations are aligned with the ones of GATES in our experiments.

6.3 Comparison of Different Encoders

Table 4 shows the comparison of GATES, GATES++ encoder, and various baseline encoders trained using different proportions of training data on three benchmarks. Note that we use the same training loss (the pairwise hinge loss with a margin of 0.1) for training all these encoders. As can be seen, on all three benchmarks, GATES could achieve higher Kendall's Taus on the testing architectures than the baseline encoders consistently with different training proportions.

Note that when there are few training architectures, the advantages of GATES and GATES++ over other baselines are especially significant, and the advantages of GATES++ over GATES is also larger in this case. For example, on NAS-Bench-201, when only 7 (0.1%) architectures are seen by the performance predictor, utilizing the same training settings, GATES and GATES++ achieve a test Kendall's Tau of 0.4309 and 0.5020, respectively, whereas Kendall's Taus achieved by MLP, LSTM, and Line-Graph GCN variant are 0.0162, 0.1935, and 0.2461, respectively.

The concatenation of zero-shot metrics in GATES++ enlarges the embedding dimension. To make the comparison more rigorous, we enlarge the embedding dimension of GATES by 6 to be the same as that of GATES++, and denote this encoder as GATES_Large. GATES_Large increases the encoder capacity but does not incorporate new zero-shot information. We can see that increasing the embedding dimension does bring improvements, but GATES++ still outperforms GATES in most cases. More specifically, in the cases that GATES outperforms GATES++, GATES_Large performs better. While in the cases that GATES++ outperforms GATES, GATES with increased embedding dimension (i.e., GATES_Large) still gets inferior performances than GATES++.

We also show the comparison of Precision@K and N@K measures in Fig. 5 and Tab. 5. We can see that GATES and GATES++ achieve consistently better performances than other encoders across different Ks on all three benchmarks. We show the scatter plot of the predicted scores and ground-truth performances and the scatter plot of the predicted ranking and ground-truth ranking in the appendix.

The results indicate that the proper modeling of data-processing DAGs in GATES and GATES++ enables us to learn a good performance predictor for unseen architectures after evaluating only a small set of architectures. Also, when there are only a few architecture-performance pairs for training the predictor, the prior operation-level information and extra architecture-level information in GATES++ are indeed beneficial.

6.4 Ablation Studies of GATES

We compare the results of using four types of ranking losses and the regression loss to train the GATES predictor on NAS-Bench-101 in Table 6. The four types of ranking losses are 1) Pairwise loss with binary cross-entropy ϕ. 2) Pairwise loss with a hinge loss function ϕ. 3) Pairwise comparator loss. 4) Listwise (ListMLE). We can see that compared with using the regression loss, using ranking losses bring consistent improvements. In our experiments, we also find that training with regression loss requires a smaller learning rate and longer time to converge, and does not work well with deep GCN or GATES models.

Evaluation Details of the Comparator Note that the evaluation of the comparator-based ranking loss is a little different than the evaluation of other ranking losses. For other ranking losses, we first compute the predicted score $\text{P}(a)$ for each test architecture, and then calculate the ranking correlation between all predicted scores and all true accuracies. However, a comparator trained using the comparator-based ranking loss must take a pair of architectures as the input and outputs a comparison result. Therefore, for evaluating the performance of a comparator, we run the randomized quick-sort procedure with the comparator to get the predicted rankings of the testing architectures. Since the comparator might not be a proper total order operator, different choices of the random pivots in the randomized quick-sort could lead to different sorted sequences. Therefore, we run randomized quick-sort with 3 different random seeds, and report the average Kendall's Tau. In practice, we find that Kendall's Taus calculated using different random seeds are very close.

TABLE 5
N@K on three benchmarks. All predictors are trained with 1% (72 architectures), 2.5% (195 architectures), 2.5% (147 architectures) of the training data on NAS-Bench-1shot-3, NAS-Bench-201, and NAS-Bench-301, respectively. The percentage in the bracket is the normalized N@K calculated by dividing the testing set size.

Encoder	NAS-Bench-1shot-3		NAS-Bench-201		NAS-Bench-301	
	N@5	N@10	N@5	N@10	N@5	N@10
MLP [10]	140 (1.92%)	98 (1.34%)	107 (1.37%)	89 (1.14%)	139 (0.27%)	139 (0.27%)
LSTM [9], [10]	155 (2.13%)	45 (0.62%)	9 (0.12%)	9 (0.12%)	260 (0.51%)	247 (0.48%)
GCN [11], [12]	146 (2.00%)	111 (1.52%)	77 (0.99%)	77 (0.99%)	1991 (3.90%)	477 (0.93%)
GATES	49 (0.67%)	23 (0.32%)	4 (0.05%)	3 (0.04%)	247 (0.48%)	102 (0.20%)
GATES++	**9 (0.12%)**	**6 (0.08%)**	**1 (0.01%)**	**1 (0.01%)**	**120 (0.23%)**	**13 (0.03%)**

TABLE 6
GATES ablation study: Kendall's Tau of using different loss functions on the NAS-Bench-101 benchmark. All experiments except "Regression (MSE) + GCN" use the GATES encoder.

Loss	Proportions of 381262 training samples							
	0.05%	0.1%	0.5%	1%	5%	10%	50%	100%
Regression (MSE) + GCN†	0.4536	0.5058	0.5587	0.5699	0.5846	0.5871	0.5901	0.5941
Pairwise (Hinge) + GCN	0.5343	0.5790	0.7915	0.8277	0.8641	0.8747	0.8918	0.8950
Regression (MSE) + GATES†	0.4935	0.5425	0.5739	0.6323	0.7439	0.7849	0.8247	0.8352
Pairwise (BCE) + GATES	0.7460	0.7696	0.8352	0.8550	0.8828	0.8913	**0.9006**	**0.9042**
Pairwise (Comparator) + GATES	0.7250	0.7622	0.8367	0.8540	0.8793	0.8891	0.8987	0.9011
Pairwise (Hinge) + GATES	**0.7634**	**0.7789**	**0.8434**	**0.8594**	0.8841	**0.8922**	0.9001	0.9030
Listwise (ListMLE) + GATES	0.7359	0.7604	0.8312	0.8558	**0.8852**	0.8897	0.9003	0.9009

†: For evaluating regression loss, we use a 1-layer GCN, and a 3-layer GATES encoder rather than 5-layer models, since we find that training deep GCN or GATES with the regression loss is unstable, and often fails to learn anything meaningful. With MSE loss, 1 layer of GCN and 3 layers of GATES achieve the best results among all layer number choices using 0.1% training data.

TABLE 7
GATES++ ablation study: Kendall's Tau of using the EES (zero-shot as a part of operation embedding) and ETS (zero-shot as an auxiliary regression target) techniques in GATES++.

Encoder	NAS-Bench-1shot-3 (7290 training)				NAS-Bench-201 (7812 training)				NAS-Bench-301 (5896 training)			
	0.5%	0.75%	1%	2.5%	0.5%	0.75%	1%	2.5%	0.5%	0.75%	1%	2.5%
GATES	0.6168	0.6309	0.6557	0.7260	0.6702	0.7328	0.7571	0.8195	0.4889	0.5209	0.5590	0.6452
GATES with EES	0.5886	0.6383	0.6796	0.7405	0.6913	0.7322	0.7651	0.8310	0.5339	0.5660	0.5815	0.6483
GATES with ETS	0.6053	0.6447	0.6922	0.7289	0.6918	0.7389	0.7770	0.8349	0.4998	0.5276	0.5616	0.6572
GATES++	0.6001	0.6260	0.6965	0.7365	0.7021	0.7404	0.7729	0.8331	0.5329	0.5713	0.5833	0.6545

6.5 Ablation Study of GATES++

GATES++ incorporates multifaceted architectural information into GATES by the enhanced encoding scheme (EES) and the enhanced training scheme (ETS). We conduct an ablation study on the usage of these two schemes. Table 7 shows the results on NAS-Bench-1shot-3, NAS-Bench-201, and NAS-Bench-301. We can see that both the enhanced encoding scheme and the enhanced training scheme improve the ranking quality of GATES. And in most cases, GATES++ equipped with these two schemes in the meantime obtain the best performance. For example, on NAS-Bench-201, when utilizing 39 (0.5%) architectures to train, GATES with EES, GATES with ETS, and GATES++ can obtain a relative improvement of 0.0211, 0.0216, and 0.0319 in Kendall's Tau, respectively.

Furthermore, to understand which metrics are useful, we conduct an item-wise ablation analysis to explore the truly useful zero-shot metrics for EES and ETS. Specifically, we separately use each metric in the EES or ETS, and demonstrate the ranking quality in Tab. 8 and Tab. 9, respectively. The results show that, in most cases, all the chosen zero-shot metrics are beneficial to the ranking quality. Among these metrics, *grad_norm* and *snip* help to obtain the largest improvement in both EES and ETS. Besides, we note that although *relu_logdet* itself can rank the architectures more accurately [21], it obtains a relatively minor improvement in ETS for GATES++. Based on the above analysis, all the chosen zero-shot metrics are used to improve the ranking quality of GATES++.

TABLE 8
GATES++ ablation study: Kendall's Tau of using the EES (zero-shot as a part of operation embedding) with each of the metrics in GATES++.

Metric	NAS-Bench-1shot-3 (7290 training)				NAS-Bench-201 (7812 training)				NAS-Bench-301 (5896 training)			
	0.5%	0.75%	1%	2.5%	0.5%	0.75%	1%	2.5%	0.5%	0.75%	1%	2.5%
None	0.6168	0.6309	0.6557	0.7260	0.6702	0.7328	0.7571	0.8195	0.4889	0.5209	0.5590	0.6452
grad_norm	0.6052	0.6378	0.6816	0.7429	0.6906	0.7406	0.7691	0.8365	0.5474	0.5664	0.6031	0.6611
snip	0.5945	0.6403	0.6796	0.7415	0.6868	0.7380	0.7686	0.8352	0.5419	0.5677	0.6015	0.6620
plain	0.6036	0.6290	0.6684	0.7312	0.6889	0.7432	0.7696	0.8363	0.5444	0.6090	0.6025	0.6613
synflow	0.6111	0.6396	0.6709	0.7328	0.6848	0.7386	0.7702	0.8345	0.5579	0.5814	0.6090	0.6622
flops	0.6058	0.6292	0.6672	0.7312	0.6872	0.7411	0.7679	0.8353	0.5451	0.5722	0.6054	0.6622
params	0.6041	0.6431	0.6753	0.7298	0.6865	0.7396	0.7591	0.8349	0.5406	0.5710	0.6021	0.6588

TABLE 9
GATES++ ablation study: Kendall's Tau of using the ETS (zero-shot as an auxiliary regression target) with each of the metrics in GATES++.

Metric	NAS-Bench-1shot-3 (7290 training)				NAS-Bench-201 (7812 training)				NAS-Bench-301 (5896 training)			
	0.5%	0.75%	1%	2.5%	0.5%	0.75%	1%	2.5%	0.5%	0.75%	1%	2.5%
None	0.6168	0.6309	0.6557	0.7260	0.6702	0.7328	0.7571	0.8195	0.4889	0.5209	0.5590	0.6452
grad_norm	0.6267	0.6389	0.6768	0.7363	0.6777	0.7426	0.7746	0.8385	0.4958	0.5203	0.5639	0.6650
snip	0.6307	0.6394	0.6735	0.7398	0.6726	0.7434	0.7689	0.8359	0.4995	0.5182	0.5675	0.6670
plain	0.6304	0.6490	0.6864	0.7447	0.6848	0.7273	0.7608	0.8234	0.4915	0.5220	0.5569	0.6666
synflow	0.6155	0.6350	0.6859	0.7270	0.6918	0.7278	0.7657	0.8230	0.4953	0.5202	0.5772	0.6615
relu_logdet	0.6064	0.6424	0.6813	0.7146	0.6909	0.7260	0.7661	0.8162	0.4934	0.5251	0.5804	0.6583

TABLE 10
Accuracy comparison of the discovered architectures by using GATES, GATES++ and baseline encoders on three benchmarks. For each configuration, the mean and standard deviation of 27 results are averaged (3 training seeds, 3 training sets, 3 sample seeds).

Encoder	NAS-Bench-1shot-3		NAS-Bench-201		NAS-Bench-301	
Training Proportion	1%	2.5%	1%	2.5%	1%	2.5%
MLP [10]	0.9330±0.0104	0.9401±0.0030	0.8820±0.0736	0.9120±0.0460	0.9442±0.0021	0.9443±0.0025
LSTM [9], [10]	0.9388±0.0052	0.9397±0.0048	0.9334±0.0045	0.9317±0.0042	0.9440±0.0022	0.9452±0.0016
GCN [11], [12]	0.9026±0.1119	0.9360±0.0069	0.9244±0.0151	0.9255±0.0107	0.9214±0.0185	0.9321±0.0087
GATES	0.9420±0.0025	0.9416±0.0020	0.9341±0.0102	0.9361±0.0051	0.9454±0.0538	0.9468±0.0024
GATES++	**0.9424±0.0028**	**0.9424±0.0020**	**0.9352±0.0080**	**0.9371±0.0045**	**0.9468±0.0060**	**0.9468±0.0025**

TABLE 11
Comparison of NAS-discovered architectures on CIFAR-10.

Method	Test Error (%)	#Params (M)	#Archs Evaluated
NASNet-A [5]	2.65	3.3	20000
AmoebaNet-B [7]	2.55	2.8	27000
NAONet [9]	2.98	28.6	1000
PNAS [8]	3.41	3.2	1160
NAONet-WS† [9]	3.53	2.5	-
DARTS† [56]	2.76	3.3	-
ENAS† [18]	2.89	4.6	-
GATES	2.58	4.1	800
GATES++	2.47	4.0	600

TABLE 12
Comparison of NAS-discovered architectures on ImageNet.

Method	Top-1 Test Error (%)	#Params (M)
NASNet-A [5]	26.0	5.3
AmoebaNet-B [7]	27.2	5.3
PNAS [8]	25.8	5.1
DARTS [56]	26.9	4.9
GHN [57]	27.0	6.1
GATES	24.1	5.6
GATES++	24.1	5.6

6.6 Architecture Search with Different Encoders

We conduct predictor-based architecture search using the GATES and GATES++ on three benchmarks, and show the results in Table 10. Specifically, when utilizing the trained predictor to sample architectures, we randomly sample 200 architectures from the search space and pick the architecture with the highest predicted score. Table 10 shows that after evaluating the same number of architectures, GATES and GATES++ can help discover architectures with higher accuracies than using the baseline encoders.

6.7 Performances on CIFAR-10 and ImageNet

We show the comparison of the test errors of different architectures in Table 11, and our discovered architectures are shown in the appendix. We can see that the architectures discovered by GATES and GATES++ can achieve a competitive test error rate of 2.58% and 2.47%, respecitvely.

We transfer the discovered architecture to ImageNet and show the training results in Tab. 12. Specifically, we increase the base channel number to 48 and stack 14 cells to construct the model. The augmented model is trained for 300 epochs with batch size 256, and the learning rate is decayed from 0.1 to 0 following a cosine schedule. We use a weight decay of 3e-5, and auxiliary towers with weight 0.4, and no dropout.

7 Conclusion

This paper proposes GATES, a graph-based neural architecture encoder with better representation ability for neural architectures. Due to its reasonable modeling of the neural architectures and intrinsic ability to handle DAG isomorphism, GATES significantly improves the predictor fitness on various cell-based search spaces.

Moreover, we propose an enhanced version of GATES, GATES++, by incorporating multifaceted architectural information about NN's operation-level and architecture-level computing semantics into the construction and training of the predictor. Specifically, operation-level zero-shot metrics are concatenated onto the trainable operation embedding of GATES, and architecture-level zero-shot metrics are used as the auxiliary regression targets when training GATES. Utilizing GATES and GATES++ in predictor-based NAS could improve the search results. Extensive experiments on three benchmark search spaces and DARTS demonstrate the effectiveness of the proposed methods.

Acknowledgments

This work was supported by National Natural Science Foundation of China (No. U19B2019, 62171313, 61832007), Beijing National Research Center for Information Science and Technology (BNRist), Tsinghua EE Xilinx AI Research Fund, and Beijing Innovation Center for Future Chips.

References

[1] A. Krizhevsky, I. Sutskever, and G. E. Hinton, "Imagenet classification with deep convolutional neural networks," in *Annual Conference on Neural Information Processing Systems (NIPS)*, vol. 25. Curran Associates, Inc., 2012.

[2] K. Simonyan and A. Zisserman, "Very deep convolutional networks for large-scale image recognition," *arXiv preprint arXiv:1409.1556*, 2014.

[3] K. He, X. Zhang, S. Ren, and J. Sun, "Deep residual learning for image recognition," in *IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, 2016, pp. 770–778.

[4] A. G. Howard, M. Zhu, B. Chen, D. Kalenichenko, W. Wang, T. Weyand, M. Andreetto, and H. Adam, "Mobilenets: Efficient convolutional neural networks for mobile vision applications," *arXiv preprint arXiv:1704.04861*, 2017.

[5] B. Zoph and Q. V. Le, "Neural architecture search with reinforcement learning," in *International Conference on Learning Representations (ICLR)*, 2017.

[6] T. Elsken, J. H. Metzen, F. Hutter *et al.*, "Neural architecture search: A survey." *Journal of Machine Learning Research (JMLR)*, vol. 20, no. 55, pp. 1–21, 2019.

[7] E. Real, A. Aggarwal, Y. Huang, and Q. V. Le, "Regularized evolution for image classifier architecture search," in *AAAI Conference on Artificial Intelligence*, vol. 33, 2019, pp. 4780–4789.

[8] 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 *European Conference on Computer Vision (ECCV)*, 2018, pp. 19–34.

[9] R. Luo, F. Tian, T. Qin, E. Chen, and T.-Y. Liu, "Neural architecture optimization," in *Annual Conference on Neural Information Processing Systems (NIPS)*. Curran Associates, Inc., 2018, pp. 7816–7827.

[10] L. Wang, Y. Zhao, Y. Jinnai, and R. Fonseca, "Alphax: exploring neural architectures with deep neural networks and monte carlo tree search," *arXiv preprint arXiv:1805.07440*, 2018.

[11] Y. Guo, Y. Zheng, M. Tan, Q. Chen, J. Chen, P. Zhao, and J. Huang, "Nat: Neural architecture transformer for accurate and compact architectures," in *Annual Conference on Neural Information Processing Systems (NeurIPS)*, 2019, pp. 735–747.

[12] H. Shi, R. Pi, H. Xu, Z. Li, J. Kwok, and T. Zhang, "Bridging the gap between sample-based and one-shot neural architecture search with bonas," *Annual Conference on Neural Information Processing Systems (NeurIPS)*, vol. 33, 2020.

[13] T. N. Kipf and M. Welling, "Semi-supervised classification with graph convolutional networks," in *International Conference on Learning Representations (ICLR)*, 2017.

[14] X. Ning, Y. Zheng, T. Zhao, Y. Wang, and H. Yang, "A generic graph-based neural architecture encoding scheme for predictor-based nas," in *European Conference on Computer Vision (ECCV)*, 2020, pp. 189–204.

[15] J. Siems, L. Zimmer, A. Zela, J. Lukasik, M. Keuper, and F. Hutter, "Nas-bench-301 and the case for surrogate benchmarks for neural architecture search," *arXiv preprint arXiv:2008.09777*, 2020.

[16] B. Baker, O. Gupta, R. Raskar, and N. Naik, "Accelerating neural architecture search using performance prediction," in *International Conference on Learning Representations Workshop (ICLRW)*, 2018.

[17] G. Bender, P.-J. Kindermans, B. Zoph, V. Vasudevan, and Q. Le, "Understanding and simplifying one-shot architecture search," in *International Conference on Machine Learning (ICML)*. PMLR, 2018, pp. 550–559.

[18] H. Pham, M. Guan, B. Zoph, Q. Le, and J. Dean, "Efficient neural architecture search via parameters sharing," in *International Conference on Machine Learning (ICML)*. PMLR, 2018, pp. 4095–4104.

[19] 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 *European Conference on Computer Vision (ECCV)*. Springer, 2020, pp. 544–560.

[20] K. Yu, C. Sciuto, M. Jaggi, C. Musat, and M. Salzmann, "Evaluating the search phase of neural architecture search," in *International Conference on Learning Representations (ICLR)*, 2020.

[21] X. Ning, C. Tang, W. Li, Z. Zhou, S. Liang, H. Yang, and Y. Wang, "Evaluating efficient performance estimators of neural architectures," in *Annual Conference on Neural Information Processing Systems (NeurIPS)*, vol. 34, 2021, pp. 12 265–12 277.

[22] Z. Zhou, X. Ning, Y. Cai, J. Han, Y. Deng, Y. Dong, H. Yang, and Y. Wang, "Close: Curriculum learning on the sharing extent towards better one-shot nas," *arXiv preprint arXiv:2207.07868*, 2022.

[23] C. Ying, A. Klein, E. Christiansen, E. Real, K. Murphy, and F. Hutter, "Nas-bench-101: Towards reproducible neural architecture search," in *International Conference on Machine Learning (ICML)*. PMLR, 2019, pp. 7105–7114.

[24] H. Liu, K. Simonyan, O. Vinyals, C. Fernando, and K. Kavukcuoglu, "Hierarchical representations for efficient architecture search," in *International Conference on Learning Representations (ICLR)*, 2018.

[25] R. Negrinho and G. Gordon, "Deeparchitect: Automatically designing and training deep architectures," *arXiv preprint arXiv:1704.08792*, 2017.

[26] K. Kandasamy, W. Neiswanger, J. Schneider, B. Poczos, and E. P. Xing, "Neural architecture search with bayesian optimisation and optimal transport," in *Annual Conference on Neural Information Processing Systems (NeurIPS)*, 2018, pp. 2016–2025.

[27] L. Dudziak, T. Chau, M. Abdelfattah, R. Lee, H. Kim, and N. Lane, "Brp-nas: Prediction-based nas using gcns," *Annual Conference on Neural Information Processing Systems (NeurIPS)*, vol. 33, pp. 10 480–10 490, 2020.

[28] M. Huang, Z. Huang, C. Li, X. Chen, H. Xu, Z. Li, and X. Liang, "Arch-graph: Acyclic architecture relation predictor for task-transferable neural architecture search," in *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition*, 2022, pp. 11 881–11 891.

[29] P. Stagge and C. Igel, "Neural network structures and isomorphisms: Random walk characteristics of the search space," in *IEEE Symposium on Combinations of Evolutionary Computation and Neural Networks*. IEEE, 2000, pp. 82–90.

[30] S. Xie, A. Kirillov, R. Girshick, and K. He, "Exploring randomly wired neural networks for image recognition," in *IEEE International Conference on Computer Vision (ICCV)*, 2019, pp. 1284–1293.

[31] X. Dong and Y. Yang, "Nas-bench-201: Extending the scope of reproducible neural architecture search," in *International Conference on Learning Representations (ICLR)*, 2020.

[32] M. Zhang, S. Jiang, Z. Cui, R. Garnett, and Y. Chen, "D-vae: a variational autoencoder for directed acyclic graphs," in *Annual Conference on Neural Information Processing Systems (NeurIPS)*, 2019, pp. 1588–1600.

[33] Z. Chen, Y. Zhan, B. Yu, M. Gong, and B. Du, "Not all operations contribute equally: Hierarchical operation-adaptive predictor for neural architecture search," in *IEEE International Conference on Computer Vision (ICCV)*, 2021, pp. 10 508–10 517.

[34] S. Han, J. Pool, J. Tran, and W. Dally, "Learning both weights and connections for efficient neural network," *Annual Conference on Neural Information Processing Systems (NIPS)*, vol. 28, 2015.

[35] W. Wen, C. Wu, Y. Wang, Y. Chen, and H. Li, "Learning structured sparsity in deep neural networks," in *Annual Conference on Neural Information Processing Systems (NIPS)*, 2016, pp. 2074–2082.

[36] X. Ning, T. Zhao, W. Li, P. Lei, Y. Wang, and H. Yang, "Dsa: More efficient budgeted pruning via differentiable sparsity allocation," in *European Conference on Computer Vision (ECCV)*. Springer, 2020, pp. 592–607.

[37] N. Lee, T. Ajanthan, and P. H. Torr, "Snip: Single-shot network pruning based on connection sensitivity," in *International Conference on Learning Representations (ICLR)*, 2019.

[38] C. Wang, G. Zhang, and R. Grosse, "Picking winning tickets before training by preserving gradient flow," in *International Conference on Learning Representations (ICLR)*, 2020.

[39] M. C. Mozer and P. Smolensky, "Skeletonization: A technique for trimming the fat from a network via relevance assessment," in *Annual Conference on Neural Information Processing Systems (NIPS)*, 1989, pp. 107–115.

[40] H. Tanaka, D. Kunin, D. L. Yamins, and S. Ganguli, "Pruning neural networks without any data by iteratively conserving synaptic flow," *Annual Conference on Neural Information Processing Systems (NeurIPS)*, vol. 33, 2020.

[41] M. Zhang, S. Su, S. Pan, X. Chang, W. Huang, and G. Haffari, "Differentiable architecture search without training nor labels: A pruning perspective," *arXiv preprint arXiv:2106.11542*, 2021.

[42] M. S. Abdelfattah, A. Mehrotra, Ł. Dudziak, and N. D. Lane, "Zero-Cost Proxies for Lightweight NAS," in *International Conference on Learning Representations (ICLR)*, 2021.

[43] M. Lin, P. Wang, Z. Sun, H. Chen, X. Sun, Q. Qian, H. Li, and R. Jin, "Zen-nas: A zero-shot nas for high-performance deep image recognition," in *IEEE International Conference on Computer Vision (ICCV)*, 2021, pp. 347–356.

[44] J. Mellor, J. Turner, A. Storkey, and E. J. Crowley, "Neural architecture search without training," in *International Conference on Machine Learning (ICML)*, 2021.

[45] J. Wu, X. Dai, D. Chen, Y. Chen, M. Liu, Y. Yu, Z. Wang, Z. Liu, M. Chen, and L. Yuan, "Stronger nas with weaker predictors," *Annual Conference on Neural Information Processing Systems (NeurIPS)*, vol. 34, pp. 28 904–28 918, 2021.

[46] W. Chen, T. yan Liu, Y. Lan, Z. ming Ma, and H. Li, "Ranking measures and loss functions in learning to rank," in *Annual Conference on Neural Information Processing Systems (NIPS)*. Curran Associates, Inc., 2009, pp. 315–323.

[47] T.-Y. Liu *et al.*, "Learning to rank for information retrieval," *Foundations and Trends® in Information Retrieval*, vol. 3, no. 3, pp. 225–331, 2009.

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

[49] R. Wang, X. Chen, M. Cheng, X. Tang, and C.-J. Hsieh, "Rank-nosh: Efficient predictor-based architecture search via non-uniform successive halving," in *IEEE International Conference on Computer Vision (ICCV)*, 2021, pp. 10 377–10 386.

[50] C. Burges, T. Shaked, E. Renshaw, A. Lazier, M. Deeds, N. Hamilton, and G. Hullender, "Learning to rank using gradient descent," in *International Conference on Machine Learning (ICML)*, 2005, pp. 89–96.

[51] A. Shashua and A. Levin, "Ranking with large margin principle: Two approaches," in *Annual Conference on Neural Information Processing Systems (NIPS)*, 2003, pp. 961–968.

[52] F. Xia, T.-Y. Liu, J. Wang, W. Zhang, and H. Li, "Listwise approach to learning to rank: theory and algorithm," in *International Conference on Machine Learning (ICML)*, 2008, pp. 1192–1199.

[53] A. Zela, J. Siems, and F. Hutter, "Nas-bench-1shot1: Benchmarking and dissecting one-shot neural architecture search," in *International Conference on Learning Representations (ICLR)*, 2019.

[54] Wikipedia contributors, "Line graph — Wikipedia, the free encyclopedia," 2004, [Online; accessed 22-July-2004]. [Online]. Available: https://en.wikipedia.org/wiki/Line_graph

[55] P. K. Sen, "Estimates of the regression coefficient based on kendall's tau," *Journal of the American Statistical Association*, vol. 63, no. 324, pp. 1379–1389, 1968.

[56] H. Liu, K. Simonyan, and Y. Yang, "DARTS: Differentiable architecture search," in *International Conference on Learning Representations (ICLR)*, 2019.

[57] C. Zhang, M. Ren, and R. Urtasun, "Graph hypernetworks for neural architecture search," in *International Conference on Learning Representations (ICLR)*, 2019.

Xuefei Ning received her B.S. and Ph.D. degrees in electronic engineering from Tsinghua University, in 2016 and 2021. Xuefei's research mainly focuses on efficient deep learning algorithm design and neural architecture search.

Yin Zheng is currently a senior researcher with Weixin Group, Tencent. He received his Ph.D degree from Tsinghua University in 2015. He serves as a PC member or reviewer for many journals and conferences in his area.

Zixuan Zhou received his B.S. degree in electronic engineering from Tsinghua University in 2021. He is currently pursuing his Ms. degree at EE Department, Tsinghua University. His research interest mainly focuses on neural architecture search.

Tianchen Zhao received his B.S. degree in electronic engineering from Beihang University in 2020. He is currently pursuing his Ms. degree at EE Department, Beihang University, and is a visiting student at Tsinghua University. His research interest mainly focuses on efficient deep learning algorithm design.

Huazhong Yang (M'97-SM'00-F'20) received B.S. degree in microelectronics in 1989, M.S. and Ph.D. degree in electronic engineering in 1993 and 1998, respectively, from Tsinghua University, Beijing. In 1993, he joined the Department of Electronic Engineering, Tsinghua University, Beijing, where he has been a Professor since 1998. Prof. Yang was awarded the Distinguished Young Researcher by NSFC in 2000, Cheung Kong Scholar by the Chinese Ministry of Education (CME) in 2012, science and technology award first prize by China Highway and Transportation Society in 2016, and technological invention award first prize by CME in 2019. He has been in charge of projects sponsored by the national science and technology major project, 863 program, NSFC, and several international research projects. Prof. Yang has authored and co-authored over 500 technical papers, 7 books, and over 180 granted Chinese patents. His current research interests include wireless sensor networks, data converters, energy-harvesting circuits, nonvolatile processors, and brain inspired computing. He has also served as the chair of Northern China ACM SIGDA Chapter science 2014, general co-chair of ASPDAC'20, navigating committee member of AsianHOST'18, and TPC member for ASP-DAC'05, APCCAS'06, ICCCAS'07, ASQED'09, and ICGCS'10.

Yu Wang (M'07-SM'14-F'22) received the B.S. and Ph.D. (with honor) degrees from Tsinghua University, Beijing, in 2002 and 2007. He is currently a tenured professor with the Department of Electronic Engineering, Tsinghua University. His research interests include application specific accelerators, brain inspired computing, parallel circuit analysis, and power/reliability aware system design. He has authored and coauthored more than 300 papers in refereed journals and conferences. He has received Best Paper Award in ASPDAC 2019, FPGA 2017, NVMSA 2017, ISVLSI 2012, and Best Poster Award in HEART 2012 with 10 Best Paper Nominations. He is a recipient of DAC under 40 innovator award (2018), IBM X10 Faculty Award (2010). He served as the TPC chair, finance chair, track chair, PC member and editors for top EDA and FPGA conferences and journals. He is the co-founder of Deephi Tech (acquired by Xilinx in 2018), which is a leading deep learning computing platform provider.