Understanding and Accelerating Neural Architecture Search With Training-Free and Theory-Grounded Metrics

Wuyang Chen, Xinyu Gong, Junru Wu, Yunchao Wei, Humphrey Shi, *Senior Member, IEEE*, Zhicheng Yan, Yi Yang, *Senior Member, IEEE*, and Zhangyang Wang, *Senior Member, IEEE*

***Abstract*—This work targets designing a principled and unified training-free framework for Neural Architecture Search (NAS), with high performance, low cost, and in-depth interpretation. NAS has been explosively studied to automate the discovery of top-performer neural networks, but suffers from heavy resource consumption and often incurs search bias due to truncated training or approximations. Recent NAS works Mellor et al. 2021, Chen et al. 2021, Abdelfattah et al. 2021 start to explore indicators that can predict a network's performance without training. However, they either leveraged limited properties of deep networks, or the benefits of their training-free indicators were not applied to more extensive search methods. By rigorous correlation analysis, we present a unified framework to understand and accelerate NAS, by disentangling "TEG" characteristics of searched networks – *T*rainability, E*x*pressivity, G*eneralization* – all assessed in a training-free manner. The TEG indicators could be scaled up and integrated with various NAS search methods, including both supernet and single-path NAS approaches. Extensive studies validate the effective and efficient guidance from our TEG-NAS framework, leading to both improved search accuracy and over 56% reduction in search time cost. Moreover, we visualize search trajectories on three landscapes of "TEG" characteristics, observing that a good local minimum is easier to find on NAS-Bench-201 given its simple topology, whereas balancing "TEG" characteristics is much harder on the DARTS space due to its complex landscape geometry.**

***Index Terms*—Generalization, linear region, neural architecture search, neural tangent kernel.**

Manuscript received 29 December 2022; revised 2 September 2023; accepted 11 October 2023. Date of publication 1 December 2023; date of current version 8 January 2024. Recommended for acceptance by N. Vasconcelos. *(Wuyang Chen, Xinyu Gong, and Junru Wu contributed equally to this work.) (Corresponding author: Zhangyang Wang.)*

Wuyang Chen, Xinyu Gong, and Zhangyang Wang are with the Department of Electrical and Computer Engineering, The University of Texas at Austin, Austin, TX 78712 USA (e-mail: wuyang.chen@utexas.edu; xinyu.gong@utexas.edu; atlaswang@utexas.edu).

Junru Wu is with the Department of Computer Science and Engineering, Texas A&M University, College Station, TX 77843 USA (e-mail: sandboxmaster@tamu.edu).

Yunchao Wei is with the Institute of Information Science, Beijing Jiaotong University, Beijing 100044, China (e-mail: yunchao.wei@bjtu.edu.cn).

Humphrey Shi is with the School of Interactive Computing, Georgia Institute of Technology, Atlanta, GA 30332 USA (e-mail: shihonghui3@gmail.com).

Zhicheng Yan is with Meta Reality Labs, Burlingame, CA 94010 USA (e-mail: zhicheng.yan@live.com).

Yi Yang is with Zhejiang University, Hangzhou 310027, China (e-mail: yangyics@zju.edu.cn).

Digital Object Identifier 10.1109/TPAMI.2023.3328347

I. Introduction

THE development of deep convolutional neural networks significantly contributes to the success of computer vision tasks [4], [5], [6], [7]. However, manually designing new network architectures not only costs tremendous time and resources, but also requires a rich network training experience that can hardly scale up. Neural architecture search (**NAS**) is recently explored to remedy the human efforts and costs, benefiting automated discovery of architectures in a given search space [8], [9], [10], [11], [12], [13], [14], [15], [16].

Despite the principled automation, NAS still suffers from heavy consumption of computation time and resources. Most NAS methods mainly leverage the validation set and conduct accuracy-driven architecture optimization. Therefore, frequent training and evaluation of sampled architectures become a severe bottleneck that hinders both search efficiency and interpretation. A super-network is extremely slow to be trained until converge [17] even with many effective heuristics for channel approximations or architecture sampling [18], [19]. Approximated proxy inference such as truncated training/early stopping can accelerate the search, but is observed to introduce severe search bias [10], [20], [21].

People recently address this problem by proposing training-free NAS. Indicators like covariance of sample-wise Jacobian [1], Neural Tangent Kernel [2], and "synflow" [3] are found to highly correlate with network's accuracy even at initialization (i.e., no gradient descent). This significantly reduces the search cost. However, these works only validated a few highly customized search approaches, and leveraged limited properties of deep networks in an empirical or ad-hoc way. Mellor et al. [1] only considered the "local linear map" defined by the covariance of sample-wise Jacobian, and only studied the random search method. Abdelfattah et al. [3] mainly leveraged "synflow" proposed in previous pruning literature [22] while relying on a warm-up stage. Chen et al. [2] considered two aspects (trainability and expressivity) and integrated two indicators, but still have to leverage highly customized supernet pruning method and cannot extend to other non-supernet NAS search methods. Moreover, these training-free indicators still only pursue final search performance and provide limited benefit towards the interpretation and understanding of the search trajectory and different search spaces.

In contrast, we target on designing a *unified* and *visualizable* training-free NAS framework that is (i) "*search method agnostic*", i.e., can be scaled up to a broad variety of popular search algorithms; (ii) "*visualizable*", i.e., can help understand search behaviors on different landscapes of architecture spaces. Our core idea is to propose indicators that can rank the network's performance, and characterize the network's properties, while still incurring no training cost. More importantly, we aim to make our training-free indicators widely applicable to multiple popular NAS methods, and also to facilitate the understanding of NAS search process.

Specifically, We first propose to disentangle the network's characteristics into three distinct aspects: **T***rainability,* **E***xpressivity,* **G***eneralization*, or "**TEG**" for short (defined in Section III). All three could be assessed with training-free indicators, and our studies demonstrate their strong correlations with the network's training or test accuracy. Further, across various network operator types and topologies, they show complementary preferences, together leading to a comprehensive picture. Extensive studies validate the effective and efficient guidance from our TEG-NAS framework, with both improvements on search accuracy and over 56% reduction on search time cost. More importantly, we for the first time visualize the search trajectory on architecture landscapes from different search spaces, thanks to our proposed TEG dimensions that disentangle different aspects of the searched model performance and can be efficiently quantified. For example, we find that a good local optimum is easier to find on NAS-Bench-201 [23] which has simpler topologies. However, it is much harder for a search method to balance TEG properties on the DARTS space with complex architecture landscapes. We summarize our contributions as:

- We perform a rigorous correlation analysis of three disentangled "**TEG**" properties against the network's training and test accuracy, and how changes made to an architecture will affect these aspects. All three notions are measured in a training-free manner. Since the three properties are complementary, they can achieve a very high correlation with the network's performance if properly combined.
- We design a unified training-free framework to provide accurate yet extremely efficient guidance during NAS search. Our framework is generally applicable to various existing NAS methods, including both supernet and single-path approaches, in a plug-and-play fashion. In both NAS-Bench-101, NAS-Bench-201, and DARTS search spaces, we trim down the search time by over 56% while improving the searched model's accuracy.
- Beyond the final search performance, we for the first time visualize the search trajectory on the architecture landscapes from different search spaces, on how the search progresses along the TEG dimensions. That leads to a novel visualization of the NAS search process, as well as insightful comparisons among different search spaces.

Paper Organization. We first review recent advanced methods for efficient NAS and topics in Deep Learning theory in Section II. We present our methods in two steps: 1) what are training-free indicators for NAS (Section III); 2) how to use training-free indicators in NAS (Section IV). In Section III we first introduce our motivation and background in analyzing the trainability/expressivity/generalization of deep networks. Definitions and architecture inductive biases of three theory-grounded indicators will be explained, and we will demonstrate that disentangling different aspects of neural architectures leads to a better ranking prediction of networks from a search space. After validating different preferences of our three training-free indicators on network architectures, in Section IV we propose a unified and interpretable NAS framework that does not require any gradient descent training. Our NAS framework can not only be easily integrated into recent popular NAS methods (reinforcement learning, evolution, supernet), but also reflect a novel visualization of architecture landscapes. This contributes to both accelerated high-performance NAS methods and interpretable tools for analyzing NAS search space. We show our final results in Section V, where we studied the search accuracy and time cost on NAS-Bench-101 [24], NAS-Bench-201 [23] and DARTS space.

The preliminary version of this work has been published in [2], and we have made significant improvements over it. First, in the main method section, we will introduce a missing part in our ICLR version – a training-free indicator for the generalization (Section III-C). As generalization is a different property of deep networks besides trainability and expressivity, we will demonstrate its strong indication of network performance, its distinct architecture preference (Section IV-A), and its contribution to the final search results. Second, this version of training-free NAS is no longer a highly customized algorithm, but a unified and generally adaptable framework, which will be verified in three popular NAS search methods in our experiments (Section V). All three NAS methods will benefit from strong search guidance and significant time cost reduction after being integrated with our general framework. Finally, our new work will facilitate search space visualization and contribute to a novel visualization of the NAS search process. By tracking and projecting the search trajectory along the three proposed TEG dimensions, we can observe distinct landscape patterns from simple to complex search spaces, which will provide insights for understanding and designing NAS search spaces.

II. Related Works

A. Neural Architecture Search

Most NAS works suffer from heavy search costs. Sampling-based methods [10], [25], [26], [27], [28] achieve accurate network evaluations, but the truncated training imposes bias on the architecture rankings. The one-shot super network [17], [19], [29], [30], [31] can share parameters to sub-networks and greatly accelerate the evaluations, but it is hard to optimize [32] and suffers from poor correlation between supernet accuracy and its sub-networks' [33]. In all, there is no clear one-winner method across the variety.

B. Efficient and Training-Free NAS

Recent NAS works start focusing on reduced training or even training-free search. EcoNAS [34] investigated different ad-hoc proxies (input size, model size, training samples, epochs, etc.)

to reduce the training cost. Mellor et al. [1] for the first time proposed a training-free NAS framework, which empirically leverages the correlation between sample-wise Jacobian to rank architectures. However, why did the Jacobian work was not clearly explained and demonstrated. Abdelfattah et al. [3] studied different training-free indicators, and leveraged "synflow" from pruning [22] as the main ranking indicator. Park et al. [35] ranked the network's performance with NTK and NNGP. Chen et al. [2] studied two theory-inspired indicators and combined with supernet pruning for further efficiency. However, these methods either leveraged ad-hoc or limited theory-driven properties of deep networks, or the benefits of their training-free strategies are tied to some specific search methods. In contrast, we hope to explore a comprehensive set of deep network properties, and further propose a unified training-free framework for various existing NAS methods.

C. Trainability, Expressivity, and Generalization

Numerous indicators in the deep learning theory field have been proposed to study various aspects of deep networks. Neural tangent kernel (NTK) is proposed to characterize the gradient descent training dynamics of wide networks [36], [37]. It was also proved that wide networks evolve as linear models under gradient descent [38]. Xiao et al. [39] further propose to decouple the network's trainability and generalization. Meanwhile, a network's expressivity can be measured as the number of linear regions separated in the input space [40], [41], [42], [43]. Many works also try to directly probe the network's generalization from various training statistics or network parameters [44], [45], [46].

III. Disentangling Trainability, Expressivity, and Generalization of Deep Networks

Trainability, expressivity, and generalization are three important, distinct, and complementary properties to characterize and understand neural networks [39], [42], [44]. Specifically, the trainability is related to the convergence speed during optimization; the expressivity is related to the network's functional complexity; and the generalization indicates a model's error on unseen data. Typically, a deep network achieves high performance when: 1) it can produce a loss landscape that is easily trainable with gradient descent, 2) it can represent sufficiently complex functions, 3) it can learn representation transferable to unseen examples, instead of just memorizing training data. In this section, we will introduce what are these training-free indicators, and in Section IV we will introduce how to use them in NAS.

A. Trainability

Training deep networks requires optimizing high-dimensional non-convex loss functions. In practice, gradient descent often finds the global or good local minimum. However, many expressible networks are not easily learnable. For example, a deep stack of convolutional layers (e.g., Vgg [4]) is much harder to train than networks with skip connections (ResNet [6], DenseNet [47], etc.), even the former could equip a larger number of parameters. The trainability of a neural network studies how effective it can be optimized by gradient descent [48], [49], [50].

Architecture Bias on Trainability: A network's architecture can control how effectively the gradient information can flow through it. These topological properties might control the amount of information that can be learned by networks. Preserving the gradient flow is found to be essential during network pruning, even at initialization [22], [51]. Skip connections also have a significant impact on the sharpness/flatness of the loss landscapes [52]. Therefore, we hypothesize that certain aspects of trainability can be characterized just by the architecture at its initialization.

Conditioning of NTK: To characterize the training dynamics of wide networks, Neural tangent kernel (NTK) is proposed [38], [53], [54], defined as:

$$\hat{\Theta}(\boldsymbol{x}, \boldsymbol{x}') = J(\boldsymbol{x})J(\boldsymbol{x}')^T, \tag{1}$$

where $J(\boldsymbol{x})$ is the Jacobian evaluated at a point $\boldsymbol{x}$. Xiao et al. [39] measures the trainability of networks by studying the spectrum and conditioning of $\hat{\Theta}$:

$$\mu_t(\boldsymbol{x}_{\text{train}}) = (\mathbf{I} - e^{-\eta\hat{\Theta}(\boldsymbol{x}_{\text{train}}, \boldsymbol{x}_{\text{train}})t})\boldsymbol{y}_{\text{train}} \tag{2}$$

$$\mu_t(\boldsymbol{x}_{\text{train}})_i = (\mathbf{I} - e^{-\eta\lambda_i t})\boldsymbol{y}_{\text{train},i}. \tag{3}$$

$\mu_t(\boldsymbol{x})$ is the expected outputs of a wide network, λ_i are the eigenvalues of $\hat{\Theta}(\boldsymbol{x}_{\text{train}}, \boldsymbol{x}_{\text{train}})$, $\boldsymbol{x}_{\text{train}}$ and $\boldsymbol{y}_{\text{train}}$ are drawn from the training set D_{train}. (3) indicates that different time is needed to learn the ith eigenmode, and thus we can conclude that the more diverse the learning speeds of different eigenmodes are, the more difficult the network can be optimized. We are therefore motivated to use the empirical condition number of NTK to represent trainability:

$$\hat{\kappa} = \mathop{\mathbb{E}}_{\substack{\boldsymbol{x}_{\text{train}} \sim D_{\text{train}} \\ \theta \sim \mathcal{N}(0, \frac{2}{N_l})}} \frac{\lambda_{\max}(\hat{\Theta}(\boldsymbol{x}_{\text{train}}, \boldsymbol{x}_{\text{train}}))}{\lambda_{\min}(\hat{\Theta}(\boldsymbol{x}_{\text{train}}, \boldsymbol{x}_{\text{train}}))}, \tag{4}$$

where network parameters θ are drawn from Kaiming normal initialization $\mathcal{N}(0, \frac{2}{N_l})$ (N_l is the width at layer l) [55], and thus $\hat{\kappa}$ is calculated at network's initialization. As shown in Fig. 1, $\hat{\kappa}$ is negatively correlated with both the network's training and test accuracy, with the Kendall-tau correlation as -0.59. Therefore, minimizing the $\hat{\kappa}$ during the search will encourage the discovery of architectures with high performance.

B. Expressivity

Recent works try to explain the success of deep networks by their ability to approximate complex functions, quantified by various complexity measures [56], [57]. The more expressible the network is, the more efficient it can fit the training data. In the case of ReLU networks that compute piecewise linear functions, the number of distinct linear regions is a natural measure of such expressivity. The composition of ReLU leads the input space partitioned into distinct pieces (i.e., linear regions). Therefore, the density of linear regions serves as a convenient proxy for the complexity of the network [40], [42], [43], [58].

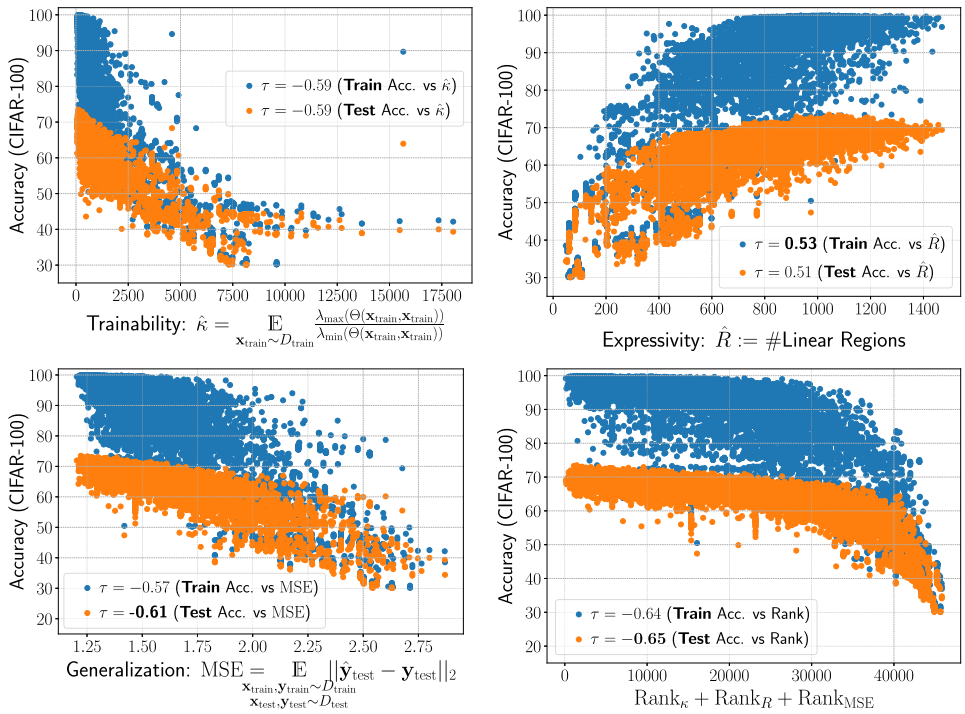


Fig. 1. From left to right: correlation of trainability, expressivity, generalization, and sum of rankings against accuracies on NAS-Bench-201 [23], all revealing strong correlations. Meanwhile, expressivity has a stronger correlation against training accuracy, and generalization has stronger correlation against test accuracy, which is aligned with their definitions.

Architecture Bias on Expressivity: It was proved that networks with random Gaussian initialization can embed the training data in a distance-preserving manner [59]. Hanin et al. [58] show that the number of activation patterns for ReLU networks is tightly bounded by the total number of neurons both at initialization and during training, and empirically showed that the number of regions stays roughly constant during training [42]. Therefore, network architecture itself has a strong inductive bias on its expressivity.

Complexity of Linear Regions: We first introduce the definition of activation patterns and see how it is connected to the number of linear regions in input space.

Definition 1 of [43] (Activation Patterns as the Linear Regions) Let $\mathcal{N}$ be a ReLU CNN. An activation pattern of $\mathcal{N}$ is a function $\boldsymbol{P}$ from the set of neurons to $\{1, -1\}$, i.e., for each neuron z in $\mathcal{N}$, we have $\boldsymbol{P}(z) \in \{1, -1\}$. Let θ be a fixed set of parameters (weights and biases) in $\mathcal{N}$, and $\boldsymbol{P}$ be an activation pattern. The region corresponding to $\boldsymbol{P}$ and θ is

$$\boldsymbol{R}(\boldsymbol{P}; \theta) := \{\boldsymbol{x}^0 \in \mathbb{R}^{C \times H \times W} : z(\boldsymbol{x}^0; \theta) \cdot \boldsymbol{P}(z) > 0, \quad \forall z \in \mathcal{N}\}, \tag{5}$$

*where $z(\boldsymbol{x}^0; \theta)$ is the pre-activation of a neuron z. Let $R_{\mathcal{N},\theta}$ denote the number of linear regions of $\mathcal{N}$ at θ, i.e., $R_{\mathcal{N},\theta} := \#\{\boldsymbol{R}(\boldsymbol{P}; \theta) : \boldsymbol{R}(\boldsymbol{P}; \theta) \neq \emptyset$˜ *for some activation pattern $\boldsymbol{P}$}.*

(5) tells us that a linear region in the input space is a set of input data $\boldsymbol{x}^0$ that satisfies a certain fixed activation pattern $\boldsymbol{P}(z)$, and therefore the number of linear regions $R_{\mathcal{N},\theta}$ measures how many unique activation patterns that can be divided by the network.

Since the input space is recursively partitioned by ReLU as the layers go deeper, and the composition of piecewise linear functions is still piecewise linear, each linear region in the input space can be uniquely represented with a set of affine parameters based on a combination of ReLU activation patterns. This means that, with given training examples and parameters, the number of linear regions $R(\boldsymbol{x}_{\text{train}}, \theta)$ can be approximated by the number of unique activation patterns combined from all ReLU layers in the whole network. We are therefore motivated to use the empirical number of linear regions to represent expressivity:

$$\hat{R} = \mathop{\mathbb{E}}_{\substack{\boldsymbol{x}_{\text{train}} \sim D_{\text{train}} \\ \theta \sim \mathcal{N}(0, \frac{2}{N_l})}} R(\boldsymbol{x}_{\text{train}}, \theta). \tag{6}$$

As shown in Fig. 1, $\hat{R}$ is positively correlated with both the network's training and test accuracy. Moreover, we observe that $\hat{R}$ has a stronger correlation with training over test accuracy, which validates that $\hat{R}$ indicates how well a network fits the training data, but not its generalizability.

C. Generalization

Typically, the generalization error[1] is defined as the risk of the model over the underlying data distribution D. A model chosen from a very complex family of functions can essentially fit all the training data, but memorization cannot guarantee the accurate

[1]Here we quantify the absolute generalization error, instead of the generalization gap.

association of unseen examples with seen ones. That makes generalization a distinct notion from trainability ("optimization") and expressivity ("memorization"), since generalization focuses on how well a model can transfer the information from seen to unseen data.

Architecture Bias on Generalization: With even random initialization, network architecture alone could have a strong inductive bias to its generalization error. Network architectures of different complexity or sparsity, without learning any weight parameters, are found to be able to encode solutions for a given task [60], [61]. Bhardwaj et al. [62] formally established a link between the structure of CNN architectures (depths, widths, number of skip connections, etc.) and their generalization errors. More importantly, inductive bias from certain architecture patterns (e.g., graph-based representation [63]) can even transfer across different types of networks (MLPs, CNNs, ResNets, etc.) and different tasks (CIFAR-10, ImageNet, etc.). Same in our work, we decouple the architecture from the network weights, and focus only on the aspect of "weight-agnostic" generalization, which is impacted by just the network architecture.

NTK Kernel Regression: Previous works [38], [39] showed that at time t during gradient descent training with an MSE loss, the expected outputs of an infinite wide network evolve as:

$$\mu_t(\boldsymbol{x}_{\text{test}}) = \hat{\Theta}(\boldsymbol{x}_{\text{test}}, \boldsymbol{x}_{\text{train}})(\hat{\Theta}(\boldsymbol{x}_{\text{train}}, \boldsymbol{x}_{\text{train}}))^{-1}(\boldsymbol{I} - e^{-\eta\hat{\Theta}(\boldsymbol{x}_{\text{train}}, \boldsymbol{x}_{\text{train}})t})\boldsymbol{y}_{\text{train}}. \tag{7}$$

Studying the evolution of $\mu_t(\boldsymbol{x}_{\text{test}})$ in (7) along the training iteration t can characterize the generalization performance of deep networks. However, the infinite width is not directly applicable in real-life scenarios, and we want to estimate the generalization at a network's initialization. Therefore in our work, we choose to empirically estimate the generalization by calculating the test MSE error of a network's NTK kernel regression:

$$\hat{\boldsymbol{y}}_{\text{test}} = \hat{\Theta}^L(\boldsymbol{x}_{\text{test}}, \boldsymbol{x}_{\text{train}})(\hat{\Theta}^L(\boldsymbol{x}_{\text{train}}, \boldsymbol{x}_{\text{train}}))^{-1}\boldsymbol{y}_{\text{train}}, \tag{8}$$

$$\text{MSE} = \mathop{\mathbb{E}}_{\substack{\boldsymbol{x}_{\text{train}}, \boldsymbol{y}_{\text{train}} \sim D_{\text{train}} \\ \boldsymbol{x}_{\text{test}}, \boldsymbol{y}_{\text{test}} \sim D_{\text{test}}}} ||\hat{\boldsymbol{y}}_{\text{test}} - \boldsymbol{y}_{\text{test}}||_2. \tag{9}$$

$\hat{\Theta}^L$ indicates the NTK evaluated only for the last layer of the deep network. Eq 8 tries to associate $\boldsymbol{x}_{\text{test}}$ with $\boldsymbol{x}_{\text{train}}$ via NTK kernel regression, and transfer the given training labels $\boldsymbol{y}_{\text{train}}$ to the test data. A deep neural network will fail to generalize if its prediction $\hat{\boldsymbol{y}}_{\text{test}}$ becomes data-independent, and the MSE in (9) will become large.

Note that we are not directly predicting a network's converged generalization at its initialization. Instead, we use MSE to compare different networks and study how it would be affected by different architectures. Adopting MSE also follows the convention that NTK is also derived under the squared loss [36], [64].

As further demonstrated in Fig. 1, MSE shows a strong negative correlation with both the network's training accuracy and test accuracy. We also observe that both training and testing accuracy drop with the increase of MSE. This is because on the observation from NAS-Bench-201, all models' training and testing accuracies are positively correlated. More importantly, MSE has a stronger correlation with the test than the training accuracy. This precisely validates that MSE represents how well a network generalizes, but not memorization of the training data.

D. Comparison With Other Zero-Cost Proxies

We further compare our training-free indicators with other publicly available zero-cost proxies [65]. We follow code at https://github.com/automl/naslib/tree/zerocost and provide our results of Spearman correlations of our training-free indicators. From Fig. 2 we can see that our three training-free indicators show strong correlations across diverse benchmarks, compared with other proxies.

At this moment we disentangled the network's performance into three distinct properties. In the next section, we present our unified and interpretable TEG-NAS strategy.

IV. TEG-NAS: A Unified and Interpretable NAS Framework

In this section we will demonstrate how to use our three training-free indicators in NAS. Our core motivation is to provide a unified training-free framework for NAS of both high performance and low cost. We also enable the visualization of NAS search trajectory on the architecture landscapes.

A. How Architecture Affects $\hat{\kappa}$, $\hat{R}$, and MSE

Despite the strong correlations and different preferences over training or testing accuracy we observe in Fig. 1, it is still unknown whether each individual aspect of three – trainability, expressivity, generalization – is necessary for a deep network to be of high performance. This analysis is also missing in previous works [2], [39]. Before we directly adopt our disentangled **TEG** properties to NAS search, we must study how changes of $\hat{\kappa}$, $\hat{R}$, MSE could be reflected on network architectures, and how network's operator types or topology will affect its trainability, expressivity, and generalization. Otherwise, if they share the same preference on selecting architectures, picking any one of them will guide the search towards similar results.

Architecture Exclusively Selected by $\hat{\kappa}$, $\hat{R}$, MSE: Trainability, expressivity, and generalization may have different preferences over the network's operator types and topology. This motivates us to summarize architectures that are exclusively selected by $\hat{\kappa}$, $\hat{R}$, MSE. We first measure the thresholds T_κ, T_R, T_{MSE} that filter top 10% architectures out of the search space $\boldsymbol{A}$, ranked by $\hat{\kappa}$, $\hat{R}$, and MSE, respectively. We define the following three subsets of architectures, with any two out of three having an empty intersection:

$$\boldsymbol{A}_\kappa = \{a | a \in \boldsymbol{A}, \hat{\kappa}_a \leq T_\kappa, \hat{R}_a < T_R, \text{MSE}_a > T_{\text{MSE}}\}, \tag{10}$$

$$\boldsymbol{A}_R = \{a | a \in \boldsymbol{A}, \hat{\kappa}_a > T_\kappa, \hat{R}_a \geq T_R, \text{MSE}_a > T_{\text{MSE}}\}, \tag{11}$$

$$\boldsymbol{A}_{\text{MSE}} = \{a | a \in \boldsymbol{A}, \hat{\kappa}_a > T_\kappa, \hat{R}_a < T_R, \text{MSE}_a \leq T_{\text{MSE}}\}. \tag{12}$$

We study three subsets of architectures in terms of both operator and topology, shown in Fig. 3. For operator types, the ratio of

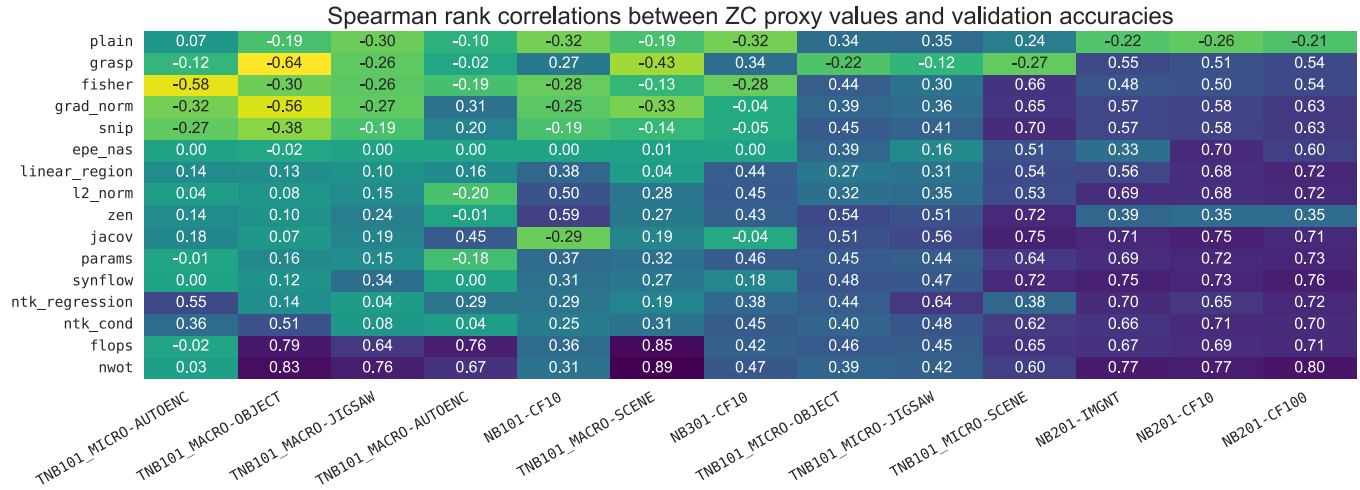


Fig. 2. Spearman rank correlation coefficients between our three training-free metrics (`ntk_cond` in Section III-A, `linear_region` in Section III-B, `ntk_regression` in Section III-C) and validation accuracies across different benchmarks, comparing with correlations of all other metrics included in Fig. 2 of [65]. The rows and columns are ordered based on the mean scores across columns and rows, respectively.

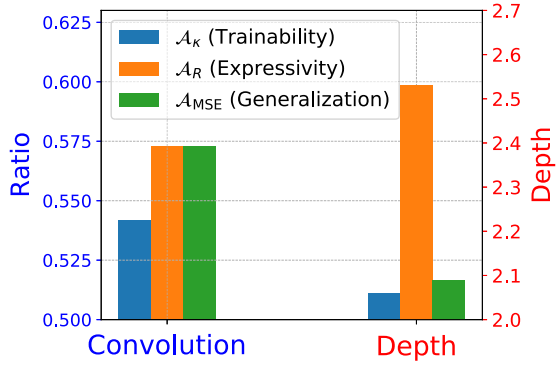


Fig. 3. Impact of network operators and topologies on $\hat{\kappa}$, $\hat{R}$, and MSE on NAS-Bench-201.

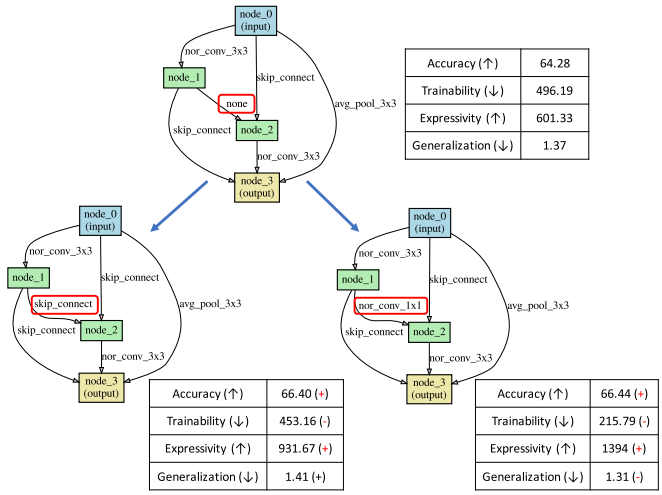


Fig. 4. "None" operator jeopardizes the architecture's trainability, expressivity, and generalization. By replacing "none" with "skip_connect" or "conv1 × 1", the bad trainability or expressivity can be addressed, leading to better test accuracy.

convolution (1×1 and 3×3) operators in $\boldsymbol{A}_\kappa$ is lower than those of $\boldsymbol{A}_R$ and $\boldsymbol{A}_{\text{MSE}}$, indicating operators with a heavy number of parameters may not friendly for optimization. In contrast, $\boldsymbol{A}_R$ and $\boldsymbol{A}_{\text{MSE}}$ favor more convolution layers, benefiting to data fitting. For network topology, the averaged depth[2] of architectures in $\boldsymbol{A}_\kappa$ is much lower than those in $\boldsymbol{A}_R$, since shallow networks are easier to train [39]. Depth from $\boldsymbol{A}_{\text{MSE}}$ is also low, contributing to better test accuracy.

Case Study: One failure reason for a bad architecture is the existence of "none" (or "zero") operator, which completely breaks the feed-forward and gradient flow. Xie et al. studied the role of the "None" operator in differentiable architecture search, highlighting the importance of the appearance and disappearance of the "None" operation during the evolution of the supernet topology [67]. In our work, we use "None" to demonstrate that our training-free metrics are sensitive to changes in a single-path subnetwork' topology.

[2]Depth of a cell is defined as the number of connections on the longest path from input to the output [66]

We show one case in Fig. 4. When there is a "none" exist, both trainability and expressivity are bad, leading to poor test accuracy. By switching into "skip_connect" or "conv1 × 1", the bad trainability or expressivity is addressed, leading to better test accuracy.

B. A Unified Training-Free NAS Framework

Different preferences of $\hat{\kappa}$, $\hat{R}$, and MSE on network's operators and topology validate their potential of guiding the NAS search. We now propose our unified training-free NAS framework (Algorithm 1). Existing NAS methods evaluate the accuracy or loss value of every single architecture via truncated training or shared supernet weights. The evaluated accuracy or

TABLE I
COMPARISON OF DIFFERENT NAS SEARCH METHODS STUDIED IN OUR EXPERIMENTS

NAS Method	NAS Formulation	Weight-sharing	Update Method	Search Stopping Criterion
REINFORCE	Single-path	No	policy gradients	policy entropy
Evolution	Single-path	No	mutation	population diversity
FP-NAS	Supernet	Yes	gradient descent	entropy of architecture parameters

Algorithm 1: Our unified training-free framework for different NAS methods.

1: **Input:** architecture search space $\boldsymbol{A}$, NAS search method $\mathcal{M}$, step $t = 0$.
2: **while not** Search Stopping Criterion of $\mathcal{M}$ satisfied **do**
3: Sample architecture: $a_t = \mathcal{M}.\text{sample}(\boldsymbol{A})$
4: Calculate $\hat{\kappa}_t$, $\hat{R}_t$, MSE_t for a_t
5: Update NAS method: $\mathcal{M}.\text{update}(a_t, \hat{\kappa}_t, \hat{R}_t, \text{MSE}_t)$
6: $t = t + 1$
7: **end while**
8: **Return** Searched architecture $\mathcal{M}$.derive().

loss is also leveraged as feedback to update the NAS method itself. Instead, we leverage the disentangled trainability, expressivity, and generalization during the search. For each architecture sampled by the NAS search method, we average three repeated calculations of $\hat{\kappa}_t$ / $\hat{R}_t$ / MSE_t, by using three independent mini-batches of training data. They will be leveraged as feedback to guide the update of the search method. For different search stopping criteria and update manners of different NAS methods, please refer to Section V-A and Table I.

Comparison With Prior Works

- Mellor et al. [1] only leveraged sample-wise correlation of Jacobian, with no detailed explanation of which aspect (trainability/expressivity/generalization) this indicator represents. Moreover, they only leveraged Random Search on NAS-Bench-201, without studying more NAS methods and search spaces.
- Abdelfattah et al. [3] mainly leveraged "synflow" indicator equipped with "warm-up" or "move proposal" search strategy, which is related to trainability. However, they still have to use trained models for proxy inference during the search.
- Chen et al. [2] built their framework on top of a super-net based approach, and strongly rely on a highly customized super-net pruning strategy. We evaluate their method without pruning (shown in Table III) and observed inferior performance.

C. Visualizing Search Process on Different Architecture Landscapes

It has been a missing part in the NAS community to visualize the search process on architecture landscapes from different search spaces. Several bottlenecks hinder this analysis: 1) evaluation via truncated training still suffers from heavy computation cost, making the architecture landscape intractable to characterize; 2) the truncated accuracy or loss value is noisy, making the trade-off of exploration-exploitation of the search process hard to observe.

We leverage Reinforcement Learning (RL) as the example, and take pioneering steps to conduct such analysis:

- To explore the global and local geometry of the architecture landscapes, at search step t we spawn a parent architecture into two children, and proceed the search of these two children with different randomness.
- We collect the trajectory of architectures from two children, and project the high-dimensional architecture space (represented by the categorical policy distribution of the RL agent) into a 2D plane via PCA.

We perform these analyses at early and late search steps on both NAS-Bench-201 space [23] and DARTS search space [17] (see search space details in Section V), shown in Fig. 5. Our observations are summarized as follows:

- On NAS-Bench-201, the search can land in areas where $\hat{\kappa}_t$, $\hat{R}_t$, and MSE_t are all good. Although some areas (e.g., area "A" in early NAS-Bench-201) enjoy local minima on one of the three aspects, the search will proceed beyond it due to its inferiority on the other two properties.
- Spawning at both early and late search stages, two children from NAS-Bench-201 land in the same area in the end. This is probably because of the simple operator types and topology from the design of NAS-Bench-201 (see details in Section V-D and original paper [23]).
- DARTS space is associated with much more complex architecture landscapes. Children spawned from both early and late stages may land in different areas, with a barrier (or a valley) on their interpolation (orange dashed line). This complex landscape introduces a significant challenge to balance and trade-off $\hat{\kappa}_t$, $\hat{R}_t$, and MSE_t for NAS search, with even noisy signals.

In general, our landscape analysis on the architecture space can be analogized to the counterpart on the parameter space [52]. The architectural landscape can influence the behavior of the architecture search. Our visualizations help explore the sharpness/flatness of architectural minimizers found by different search methods, in different architecture spaces, and the choices of different architectural compositions (skip connections, channel numbers, network depths, etc.). Specifically, the usage and impact of our landscape analysis are explained below:

1) Compare different search spaces: given the same search method, the search trajectory on different search spaces will lead to different behaviors. If a search space incurs plenty of barriers or valleys on the trajectory, that means it poses challenges for the search method to converge.

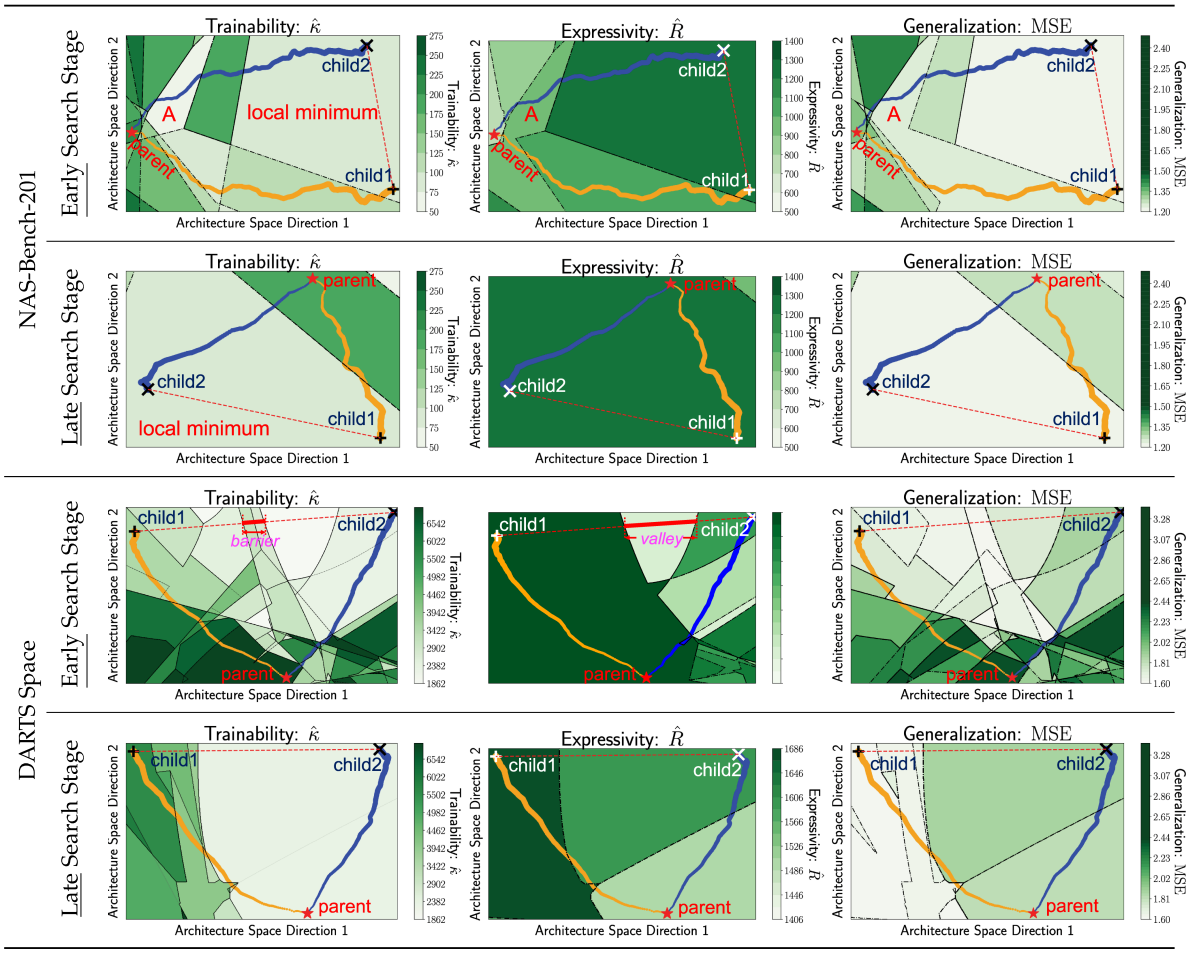


Fig. 5. Architecture landscape with respect to trainability (left), expressivity (middle), and generalization (right) on a 2D plane projected via PCA from the search space. The red star indicates the parent of a search by Reinforcement Learning, and two crosses represent two children spawned from the same parent (at "Early" or "Late" search stage), but searched with different randomness. The simple geometry of NAS-Bench-201 is easier to search, whereas the DARTS space is much more complex and hard to explore.

2) Compare different search algorithms: given the same search space, besides the final search performance, a good search algorithm should be able to explore a vast amount of areas in a search space, instead of being trapped in local regions.
3) Design search space: [68], [69], visualizing the architecture landscapes can help study the complexity and geometry of the search space and avoid rough architecture landscapes. Specifically, we can evaluate compositions in a search space, by comparing the change in the landscape when we add different operators or connections.
4) Design search algorithm: visualizing the search process can reveal the quality and stability of the search via different spawning and randomness.

V. Experiments

Following the experimental setting in [1], [2], [3], [79], [80], [81], [82], [83], in this section, we evaluate our TEG-NAS framework on three commonly used search spaces: NAS-Bench-101 [24], NAS-Bench-201 [23], and DARTS [17]. For DARTS space, we conduct experiments on both CIFAR-10 and ImageNet (Section V-E). For NAS-Bench-201, we test all three supported datasets (CIFAR-10, CIFAR-100, ImageNet-16-120 [85]) in Section V-D.

A. Studied Search Methods

Reinforce [10], [76]: Reinforcement learning (RL) treats the NAS search process as a sequential decision-making process. The policy agent formulates a single-path architecture by choosing a sequence of operators as actions, and uses the accuracy, loss value, or our training-free indicators of sampled architecture as the reward to update its internal policy distribution.

Evolution [25]: Starting from a randomly initialized pool of architectures, the evolution keeps updating the population by mutating the high-ranked architectures. The ranking criteria could be the accuracy, loss value, or our training-free indicators of sampled architectures.

Fast Probabilistic NAS: FP-NAS views search evaluations from an underlying distribution over architectures [77]. It constructs a supernet and corresponding architecture

parameters. Leveraging Importance Weighted Monte-Carlo EB algorithm [86], architecture parameters are optimized to maximize the model likelihoods of sampled architectures, which are weighted by a proxy architecture performance indicator, like accuracy, loss value, or training-free indicators.

B. Implementation Details

In this section, we include more details regarding Algorithm 1 in terms of different search methods.

1) Reinforcement Learning: The policy agent maintains an internal state to represent the architecture search space, denoted as $\boldsymbol{\theta}^{\boldsymbol{A}}$. This internal state can be converted to a categorical distribution of the architectures ($\boldsymbol{\mathcal{A}}$) via softmax: $\boldsymbol{\mathcal{A}} = \sigma(\boldsymbol{\theta}^{\boldsymbol{A}})$.

Stopping Criterion: We stop the RL search when the entropy of $\boldsymbol{\mathcal{A}}$ stops decreasing (total iterations $T = 500$ in our work). We train the RL agent with a learning rate as $\eta = 0.04$ on NAS-Bench-201 and $\eta = 0.07$ on DARTS space.

Architecture Sampling: In each iteration, the agent samples one architecture a_t from $\boldsymbol{\mathcal{A}}$.

Update: We update the RL agent via policy gradients.

$$\boldsymbol{\theta}^{\boldsymbol{A}}_{t+1} = \boldsymbol{\theta}^{\boldsymbol{A}}_{t} - \eta \cdot \nabla_{\boldsymbol{\theta}^{\boldsymbol{A}}} f(\boldsymbol{\theta}^{\boldsymbol{A}}_{t}) \quad t = 1, \ldots, T \tag{13}$$

$$f(\boldsymbol{\theta}^{\boldsymbol{A}}_{t}) = -\log(\sigma(\boldsymbol{\theta}^{\boldsymbol{A}})) \cdot (r_t - b_t) \tag{14}$$

$$b_t = \gamma b_{t-1} + (1-\gamma) r_t \quad (b_0 = 0, \gamma = 0.9) \tag{15}$$

r stands for reward, and b for an exponential moving average of reward for the purpose of variance reduction. For the baseline method, the reward is taken from the proxy inference, i.e., the test accuracy by 1-epoch truncated training. For our TEG-NAS, the reward is composited of three parts: $r = r^{\kappa} + r^{R} + r^{\text{MSE}}$, and we show the justification for how we combine our indicators in Table VI. Taking r^{κ} (reward from trainability) as the example:

$$r^{\kappa}_{t} = \frac{\hat{\kappa}_t - \hat{\kappa}_{t-1}}{\hat{\kappa}_{\max,t} - \hat{\kappa}_{\min,t}} \tag{16}$$

$$\hat{\kappa}_{\max,t} = \max(\hat{\kappa}_1, \hat{\kappa}_2, \ldots, \hat{\kappa}_t) \tag{17}$$

$$\hat{\kappa}_{\min,t} = \min(\hat{\kappa}_1, \hat{\kappa}_2, \ldots, \hat{\kappa}_t) \tag{18}$$

where $\hat{\kappa}_t$ is the evaluated trainability of the architecture sampled at step t. We calculate r^{R} (expressivity) and r^{MSE} (generalization) in the same way.

Architecture Deriving: To derive the final searched network, the agent chooses the architecture that has the highest probability, i.e., $a^* = \text{argmax}_a \sigma(\boldsymbol{\theta}^{\boldsymbol{A}})(a)$.

2) Evolution: The evolution search is first initialized with a population of 256 architectures by random sampling. We choose this size of the population based on Fig. A-1(a) from [25].

Stopping Criterion: We stop the Evolution search when the population diversity stops decreasing (1000 iterations in our work). Population diversity is calculated as the averaged pairwise architecture difference in their operator types.

Architecture Sampling: Following Real et al. [25], in each iteration, the evolution search first randomly samples a subset of 64 architectures out of the population. We choose this sampling size based on Fig. A-1(a) from [25]. Next, the best architecture (a_t) from this subset is selected. For the baseline method, the best architecture is the top1 ranked by the proxy inference, i.e., the test accuracy by 1-epoch truncated training. For our TEG-NAS, the best architecture is the top1 by the sum of three rankings by trainability, expressivity, and generalization: $\text{rank}^{\kappa} + \text{rank}^{R} + \text{rank}^{\text{MSE}}$.

Update: Following Real et al. [25], in each iteration the population is updated by adding a new architecture and popping out the oldest architecture (the one that stays in the population for the longest time). The new architecture is generated by mutating the sampled one mentioned above. We follow the same mutation strategy from Real et al. [25].

Architecture Deriving: To derive the final searched network, the best architecture from the population is selected, where the criterion is the same as we choose a_t (see above "Architecture Sampling").

3) Fast Probabilistic NAS: The Fast Probabilistic NAS (FP-NAS) formulates the search space as a supernet and shares its parameters to its sub-networks. The original FP-NAS search performs alternative optimization between network parameters and the architecture parameters (denoted as $\boldsymbol{\theta}^{\boldsymbol{A}}$). This architecture parameter can be converted to a categorical distribution of the architectures ($\boldsymbol{\mathcal{A}}$) via softmax: $\boldsymbol{\mathcal{A}} = \sigma(\boldsymbol{\theta}^{\boldsymbol{A}})$.

Stopping Criterion: We stop the FP-NAS search when the entropy of $\boldsymbol{\mathcal{A}}$ stops decreasing (total epochs $T = 100$ in our work). We update the architecture parameters with a learning rate as $\eta = 0.1$.

Architecture Sampling: In each step, the FP-NAS samples a subset $\boldsymbol{A}_t$ of $\lambda H(\text{Prob}(a_i|\boldsymbol{\mathcal{A}}))$ architectures from $\boldsymbol{\mathcal{A}}$. Here H denotes the distribution entropy and λ is a pre-defined scaling factor where we set it to 0.25.

Update: We update the architecture parameters by stochastic gradient descent.

$$\boldsymbol{\theta}^{\boldsymbol{A}}_{t+1} = \boldsymbol{\theta}^{\boldsymbol{A}}_{t} - \eta \cdot \nabla_{\boldsymbol{\theta}^{\boldsymbol{A}}} f(\boldsymbol{\theta}^{\boldsymbol{A}}_{t}) \quad t = 1, \ldots, T \tag{19}$$

$$f(\boldsymbol{\theta}^{\boldsymbol{A}}_{t}) = -\sum_{i=1}^{|\boldsymbol{A}_t|} \log(\text{Prob}(a_i|\boldsymbol{\mathcal{A}})) \cdot \frac{e^{r_i}}{\sum_{j=1}^{|\boldsymbol{A}_t|} e^{r_j}} \tag{20}$$

r stands for reward, calculated in the same way as we did for Reinforcement Learning (Section V-B1).

Architecture Deriving: To derive the final searched network, the FP-NAS chooses the architecture that has the highest probability, i.e., $a^* = \text{argmax}_a \sigma(\boldsymbol{\theta}^{\boldsymbol{A}})(a)$.

C. Results on NAS-Bench-101

NAS-Bench-101 [24] contains 423,624 unique neural architectures exhaustively generated and evaluated from a fixed graph-based search space. The search space is extremely diverse yet expressive, due to its general encoding scheme, consisting of an adjacency matrix and its corresponding operations at each vertex. Specifically, the adjacency matrix is represented by a 7×7 upper-triangular binary matrix, while the operation at each vertex could be any of three operator types: *conv*1×1, *conv*3×3 *convolution*, and *average pooling* 3×3. Each network is trained for 108 epochs and the network's accuracy at intermediate epoch(s) is also provided. For the baseline methods,

TABLE II
SEARCH PERFORMANCE ON NAS-BENCH-101

Method	GPU Hours	#Queries	Test Acc.(%)	STD(%)	Test Regret(%)	Avg. Rank	Search Method
LaNAS [70]	107.3†	200	93.90	-	0.42	168.1	Sample-based
BONAS [71]	107.3†	200	94.09	-	0.23	18.0	Sample-based
NASBOWLr [72]	80.5†	150	94.09	-	0.23	18.0	Sample-based
CATE (cate-DNGO-LS) [73]	80.5†	150	94.10	-	0.22	12.3	Sample-based
WeakNAS [74]	80.5†	150	94.10	0.19	0.22	12.3	Sample-based
ZERO-COST NAS [3]‡	27.3†	51	94.22	-	0.10	3.0	Training-Free + Sample-based
Synflow [22]	-	-	91.31	0.02	3.01	156663.0	Training-Free
NASWOT [1]	0.006	-	91.77	0.05	2.55	118291.0	Training-Free
AREA (Evolution + NASWOT) [1]	3.33	-	93.91	0.29	0.41	153.0	Training-Free
GenNAS-N [75]	5.75	-	93.92	0.004	0.40	135.0	Training-Free
Evolution	2.22	170	92.17	2.19	2.15	85891.0	Sample-based
Evolution + TEG (ours)	0.78	250	92.52	1.30	1.80	59676.0	Training-Free
REINFORCE	2.77	200	93.80	0.12	0.52	441.0	Sample-based
REINFORCE + TEG (ours)	0.24	250	94.11	0.11	0.21	12.0	Training-Free
Optimal	-	-	94.32	-	0.00	1.0	-

† Estimated results via the number of queries, where each query in NAS-Bench-101 takes an average of 1932s to train from scratch.
‡ ZERO-COST NAS [3] use training-free metrics to warm-up and initialize the sampled-based search algorithm, thus is considered a hybrid of both.
We ran teg-nas for 10 times and report the mean test accuracy and STD.

TABLE III
SEARCH PERFORMANCE FROM NAS-BENCH-201

Architecture	**CIFAR-10**	**CIFAR-100**	**ImageNet-16-120**	**Search Cost (GPU sec.)**	**Search Method**
ResNet [6]	93.97	70.86	43.63	-	-
RSPS [78]	87.66(1.69)	58.33(4.34)	31.14(3.88)	8007.13	random
ENAS [10]	54.30(0.00)	15.61(0.00)	16.32(0.00)	13314.51	RL
DARTS (1st) [17]	54.30(0.00)	15.61(0.00)	16.32(0.00)	10889.87	gradient
DARTS (2nd) [17]	54.30(0.00)	15.61(0.00)	16.32(0.00)	29901.67	gradient
GDAS [19]	93.61(0.09)	70.70(0.30)	41.84(0.90)	28925.91	gradient
DrNAS [79]	94.36(0.00)	73.51(0.00)	46.34(0.00)	-	gradient
RLNAS [80]	93.45	70.71	43.70	-	gradient
G-EA [81]	93.98(0.18)	72.12(0.35)	45.94(0.71)	18567	gradient
β-DARTS [82]	94.36(0.00)	73.51(0.00)	46.34(0.00)	11520	gradient
Single-DARTS [83]	94.36(0.00)	73.51(0.00)	46.34(0.00)	-	gradient
GM + DARTS [84]	93.72(0.12)	71.83(0.97)	42.60(0.00)	-	gradient
NASWOT ($N = 1000$) [1]	92.96(0.81)	69.98(1.22)	44.44(2.10)	306.19	training-free
TE-NAS [2]	93.9(0.47)	71.24(0.56)	42.38(0.46)	1558	training-free
TE-NAS + TEG (ours)	93.94(0.2)	71.44(0.81)	44.11(0.88)	3330.5	training-free
REINFORCE [76]	90.00(1.16)	68.40(5.93)	44.78(1.20)	33.9k/35.9k/63.5k	RL
REINFORCE + TEG (ours)	90.21(0.67)	70.42(0.36)	44.88(0.91)	3668.5	training-free
Evolution [25]	90.92(0.31)	69.32(3.31)	44.33(1.81)	33.7k/38.1k/148.3k	evolution
Evolution + TEG (ours)	91.00(0.33)	70.10(1.47)	44.45(0.75)	9939.5	training-free
FP-NAS [77]	55.38(1.52)	22.30(15.45)	16.96(6.43)	3.7k/6.6k/17.2k	gradient
FP-NAS + TEG (ours)	93.73(0.50)	70.36(0.44)	46.03(0.10)	641.67	training-free
Optimal	94.37	73.51	47.31	-	-

Test accuracy with mean and deviation are reported. "Optimal" indicates the best test accuracy achievable in the space. The search time cost of our TEG-NAS is agnostic to the size of the dataset (section iv-b). For REINFORCE [76], evolution [25], and FP-NAS [77], three search costs are listed for CIFAR-10/CIFAR-100/ImageNet-16-120.

the RL agent and Evolution use the validation accuracy after 2-epoch training as the reward or ranking criteria. For all results on NAS-Bench-101, we run for 10 independent times with different random seeds and the mean and standard deviation of test accuracy are reported. Due to the slight difference in test accuracies of architectures, we also include test regret (the absolute accuracy gap to global optimal) and average rank (the ranking distance to global optimal) for a clearer comparison across different search methods.

As shown in Table II, combing our TEG-NAS with REINFORCE or Evolution, we achieve better performance over the baseline. We significantly boost the searched test accuracy (over 0.3%+) while reducing more than 64% search time cost. Note that we did not evaluate supernet-based NAS methods (FP-NAS, TE-NAS, gradient-based NAS) on NAS-Bench-101, since the general graph-based encoding scheme in NAS-Bench-101 makes it incompatible with weight-sharing supernet, which is required in gradient-based NAS.

TABLE IV
SEARCH PERFORMANCE FROM DARTS SPACE ON CIFAR-10

Architecture	Test Error (%)	Params (M)	Search Cost (GPU days)	Search Method
AmoebaNet-A [25]	3.34	3.2	3150	evolution
PNAS [11]*	3.41	3.2	225	SMBO
ENAS [10]	2.89	4.6	0.5	RL
NASNet-A [87]	2.65	3.3	2000	RL
DARTS (1st) [17]	3.00	3.3	0.4	gradient
SNAS [88]	2.85	2.8	1.5	gradient
GDAS [19]	2.82	2.5	0.17	gradient
BayesNAS [89]	2.81	3.4	0.2	gradient
ProxylessNAS [90]†	2.08	5.7	4.0	gradient
NASP [91]	2.83 (0.09)	3.3	0.1	gradient
P-DARTS [92]	2.50	3.4	0.3	gradient
PC-DARTS [18]	2.57	3.6	0.1	gradient
R-DARTS (L2) [93]	2.95 (0.21)	-	1.6	gradient
SGAS (Cri 1. avg) [31]	2.66 (0.24)	3.7	0.25	gradient
SDARTS-ADV [94]	2.61	3.3	1.3	gradient
DrNAS [79]	2.46 (0.03)	4.1	0.6‡	gradient
β-DARTS [82]	2.53 (0.08)	3.75 (0.15)	0.4	gradient
Single-DARTS [83]	2.46	3.3	-	gradient
Few-shot DARTS-Small [95]	2.31 (0.08)	3.8	1.35	gradient
GM + DARTS (1st) [84]	2.46 (0.07)	3.7	1.1	gradient
TE-NAS [2]	2.63	3.8	0.05‡	training-free
TE-NAS + TEG (ours)	2.58(0.01)	4.8 (0.1)	0.05‡	training-free
REINFORCE [76]	3.25(0.43)	2.4 (0.3)	1.1‡	RL
REINFORCE + TEG (ours)	2.87(0.04)	4.0 (0.3)	0.15‡	training-free
Evolution [25]	3.24(0.17)	2.1 (0.1)	2.6‡	evolution
Evolution + TEG (ours)	3.16(0.35)	3.3 (0.5)	0.4‡	training-free
FP-NAS [77]	4.61(0.56)	2.2 (0.1)	0.3‡	gradient
FP-NAS + TEG (ours)	2.74(0.18)	4.4 (0.2)	0.13‡	training-free

* No cutout augmentation.
† Different space: PyramidNet [96] as the backbone.
‡ Recorded on a single GTX 1080Ti GPU.
Our results are averaged over three searched architectures under different random seeds, with standard deviations in parentheses.

D. Results on NAS-Bench-201

NAS-Bench-201 [23] provides a cell-based search space and the performance of all 15,625 networks it contains using a unified protocol. The network's accuracy is directly available by querying the database, benefiting the study of NAS methods without network evaluation. It contains five operator types: *none* (*zero*), *skip connection*, *conv*1×1, *conv*3×3 *convolution*, and *average pooling* 3×3. We refer to their paper for details of the space. For the baseline methods, the RL agent and Evolution use the test accuracy after 1-epoch training as the reward or ranking criteria. The FP-NAS uses alternative training between architecture parameters and supernet parameters with stochastic gradient descent. For all results we report, we run for four independent times with different random seeds, and report the mean and standard deviation in Table III.

We can see that for all three NAS methods (REINFORCE, Evolution, FP-NAS), our TEG-NAS framework boosts the search performance while significantly reducing the search time cost. Moreover, by adopting our unified framework, the accuracy of TE-NAS can be further improved.

E. Results on DARTS Search Space

Architecture Space: The DARTS space contains eight operator types: *none* (*zero*), *skip connection*, *separable convolution* 3×3 and 5×5, *dilated separable convolution* 3×3 and 5×5, *max pooling* 3×3, *average pooling* 3×3. We stack 20 cells to compose the network and set the initial channel number as 36 [17], [18], [92]. We place the reduction cells at the 1/3 and 2/3 of the network. Each cell contains six nodes.

The architecture for ImageNet is slightly different: the network is stacked with 14 cells with the initial channel number set to 48 [18], [92]. The spatial resolution is downscaled from 224×224 to 28×28 with the first three convolution layers of stride 2.

TABLE V
SEARCH PERFORMANCE FROM DARTS SPACE ON IMAGENET

Architecture	Test Error(%)		Params	Search Cost	Search
	top-1	top-5	(M)	(GPU days)	Method
NASNet-A [87]	26.0	8.4	5.3	2000	RL
AmoebaNet-C [25]	24.3	7.6	6.4	3150	evolution
PNAS [11]	25.8	8.1	5.1	225	SMBO
MnasNet-92 [97]	25.2	8.0	4.4	-	RL
DARTS (2nd) [17]	26.7	8.7	4.7	4.0	gradient
SNAS (mild) [88]	27.3	9.2	4.3	1.5	gradient
GDAS [19]	26.0	8.5	5.3	0.21	gradient
BayesNAS [89]	26.5	8.9	3.9	0.2	gradient
P-DARTS (CIFAR-10) [92]	24.4	7.4	4.9	0.3	gradient
P-DARTS (CIFAR-100) [92]	24.7	7.5	5.1	0.3	gradient
PC-DARTS (CIFAR-10) [18]	25.1	7.8	5.3	0.1	gradient
ProxylessNAS (GPU) [90]†	24.9	7.5	7.1	8.3	gradient
OFA w/ PS [98]†	24.0	-	-	1.67	gradient
PC-DARTS (ImageNet) [18]†	24.2	7.3	5.3	3.8	gradient
SGAS (Cri 1. avg) [31]	24.42 (0.16)	7.29 (0.09)	5.3	0.25	gradient
DrNAS [79]†	23.7	7.1	5.7	4.6	gradient
RLNAS [80]	24.0	7.1	5.7	-	gradient
β-DARTS [82]	23.9	7.0	5.5	0.4	gradient
Single-DARTS [83]	23.0	-	6.6	-	gradient
GM + DARTS (2nd) [84]	26.7	8.7	4.7	1.0	gradient
TE-NAS [2]	26.2	8.3	6.3	0.05	training-free
TE-NAS + TEG (ours)	23.6 (0.1)	7.1 (0.03)	6.6 (0.1)	0.05	training-free
REINFORCE [76]	28.2 (1.8)	9.6 (1.1)	3.8 (0.4)	1.1	RL
REINFORCE + TEG (ours)	25.1 (0.2)	7.7 (0.1)	5.6 (0.3)	0.15	training-free
Evolution [25]	29.1 (0.8)	10.2 (0.5)	3.3 (0.1)	2.6	evolution
Evolution + TEG (ours)	26.3 (0.6)	8.4 (0.3)	4.8 (0.6)	0.4	training-free
FP-NAS [77]	31.2 (1.0)	11.4 (0.7)	3.4 (0.1)	0.3	gradient
FP-NAS + TEG (ours)	23.7 (0.2)	7.0 (0.1)	6.1 (0.2)	0.13	training-free

† Architecture searched on ImageNet, otherwise searched on CIFAR-10 or CIFAR-100.
Our results are averaged over three searched architectures under different random seeds, with standard deviations in parentheses.

TABLE VI
ABLATION STUDY OF DIFFERENT COMBINATIONS OF TRAINING-FREE INDICATORS FOR REINFORCE NAS METHOD ON NAS-BENCH-201 CIFAR-100

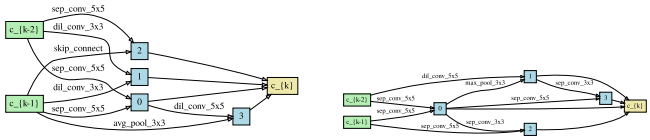
Indicators	Accuracy	GPU secs.
Baseline (1-epoch training)	68.4(5.93)	19786
$\hat{R}$	69.06(2.15)	254.6
$\hat{\kappa}$	69.27(0.73)	1574
MSE	69.68(0.88)	2447.7
$\hat{\kappa}$, $\hat{R}$	69.89(0.98)	1716.1
$\hat{R}$, MSE	70.00(0.42)	2667.4
$\hat{\kappa}$, MSE	70.18(0.04)	2704
$\hat{\kappa}$, $\hat{R}$, MSE	70.42(0.36)	3668.5

Test accuracy with mean and deviation are reported.

Evaluation Protocols: We follow previous NAS works [18], [92], [94] to evaluate architectures after search. On CIFAR-10, we train the searched network with cutout regularization of length 16, drop-path [87] with probability as 0.3, and an auxiliary tower of weight 0.4. On ImageNet, we also use label smoothing during training. On both CIFAR-10 and ImageNet, the network is optimized by an SGD optimizer with cosine annealing, with a learning rate initialized as 0.025 and 0.5, respectively.

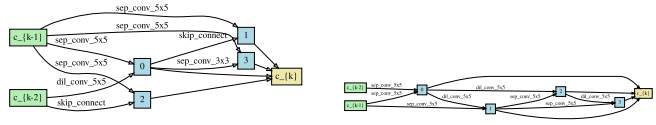
Results: For example, on ImageNet, our TEG brings improvements for: REINFORCE +3.1% top-1, -86.4% time cost; Evolution +2.8% top-1, -84.6% time cost; FP-NAS +7.5% top-1, -56.7% time cost. All training-free versions of three NAS methods can now complete the search with less than a half GPU day. These search improvements on the large-scale DARTS space and datasets validate the effectiveness and efficiency of our unified TEG-NAS framework.

We also notice that FP-NAS benefits the most by equipping our TEG method ($+1.87\%$ on CIFAR-10 and $+7.5\%$ on ImageNet). The underlying problem of ProbNAS is similar to DARTS. As a weight-sharing NAS method, skip-connection favors the gradient flow during search, which introduces a strong bias in the supernet parameters. At the end of search the supernet's accuracy can not faithfully represent the ranking of single-path networks. This problem is pointed out in recent NAS works [32], [94]. In contrast, our training-free method can address this problem: we avoid any gradient descent, and the shared weight (at its initialization) will not be affected by any inductive bias during training, thus unleashing more power of weight-sharing NAS methods.



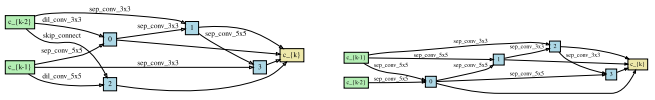
(a) Normal Cell (b) Reduction Cell

Fig. 6. Normal and reduction cells discovered by RL + TEG-NAS on DARTS space on CIFAR-10.



(a) Normal Cell (b) Reduction Cell

Fig. 7. Normal and reduction cells discovered by evolution + TEG-NAS on DARTS space on CIFAR-10.



(a) Normal Cell (b) Reduction Cell

Fig. 8. Normal and reduction cells discovered by FP-NAS + TEG-NAS on DARTS space on CIFAR-10.

F. Searched Architecture on DARTS Search Space

We visualize the searched normal and reduction cells on DARTS space, by Reinforcement Learning (Fig. 6), Evolution (Fig. 7), and FP-NAS (Fig. 8)).

G. Ablation Study on $\hat{\kappa}$, $\hat{R}$, and MSE

To validate the necessity of considering all of the trainability, expressivity, and generalization, we conduct an ablation study in Table VI using Reinforcement Learning on Cifar100 on NAS-Bench-201. This ablation study is conducted under the same settings as in Section V-D. As the baseline method, the RL agent uses the test accuracy after 1-epoch training as the reward. We can see that the guidance from every single indicator outperforms the truncated training, with much less search time cost. Finally, we achieve the best search performance once equipped with all $\hat{\kappa}$, $\hat{R}$, and MSE.

VI. Conclusion

We proposed a *unified* and *visualizable* NAS framework that benefits both various popular search methods and search interpretation. We successfully disentangle the network's characteristics into three distinct aspects: **T***rainability,* **E***xpressivity,* **G***eneralization*, or "**TEG**" for short, and leverage all of them to provide effective and efficient guidance for NAS search. Extensive studies on different NAS search methods validate the superior performance of our TEG-NAS framework. More importantly, we for the first time visualize the search trajectory on architecture landscapes from different search spaces, contributing to a better understanding of both the search and geometry of architecture space. We hope our work encourages the community to further explore NAS methods that benefit from extremely low cost, and provide a better understanding of the architectures and complexity of different search spaces.

References

[1] J. Mellor, J. Turner, A. Storkey, and E. J. Crowley, "Neural architecture search without training," in *Proc. Int. Conf. Mach. Learn.*, 2021, pp. 7588–7598.
[2] W. Chen, X. Gong, and Z. Wang, "Neural architecture search on imagenet in four GPU hours: A theoretically inspired perspective," in *Proc. Int. Conf. Learn. Representations*, 2021. [Online]. Available: https://arxiv.org/pdf/2102.11535.pdf
[3] M. Abdelfattah, A. Mehrotra, L. Dudziak, and D. N. Lane, "Zero-cost proxies for lightweight NAS," in *Proc. Int. Conf. Learn. Representations*, 2021. [Online]. Available: https://arxiv.org/pdf/2101.08134.pdf
[4] K. Simonyan and A. Zisserman, "Very deep convolutional networks for large-scale image recognition," 2014, *arXiv:1409.1556*.
[5] C. Szegedy et al., "Going deeper with convolutions," in *Proc. IEEE Conf. Comput. Vis. Pattern Recognit.*, 2015, pp. 1–9.
[6] K. He, X. Zhang, S. Ren, and J. Sun, "Deep residual learning for image recognition," in *Proc. IEEE Conf. Comput. Vis. Pattern Recognit.*, 2016, pp. 770–778.
[7] S. Xie, R. Girshick, P. Dollár, Z. Tu, and K. He, "Aggregated residual transformations for deep neural networks," in *Proc. IEEE Conf. Comput. Vis. Pattern Recognit.*, 2017, pp. 1492–1500.
[8] B. Zoph and Q. V. Le, "Neural architecture search with reinforcement learning," 2016, *arXiv:1611.01578*.
[9] A. Brock, T. Lim, J. M. Ritchie, and N. Weston, "SMASH: One-shot model architecture search through hypernetworks," 2017, *arXiv: 1708.05344*.
[10] H. Pham, M. Y. Guan, B. Zoph, Q. V. Le, and J. Dean, "Efficient neural architecture search via parameter sharing," 2018, *arXiv: 1802.03268*.
[11] C. Liu et al., "Progressive neural architecture search," in *Proc. Eur. Conf. Comput. Vis.*, 2018, pp. 19–34.
[12] L.-C. Chen et al., "Searching for efficient multi-scale architectures for dense image prediction," in *Proc. Adv. Neural Inf. Process. Syst.*, 2018, pp. 8699–8710.
[13] G. Bender, P.-J. Kindermans, B. Zoph, V. Vasudevan, and Q. Le, "Understanding and simplifying one-shot architecture search," in *Proc. Int. Conf. Mach. Learn.*, 2018, pp. 549–558.
[14] X. Gong, S. Chang, Y. Jiang, and Z. Wang, "AutoGAN: Neural architecture search for generative adversarial networks," in *Proc. IEEE Int. Conf. Comput. Vis.*, 2019, pp. 3223–3233.
[15] Y. Fu, W. Chen, H. Wang, H. Li, Y. Lin, and Z. Wang, "AutoGAN-distiller: Searching to compress generative adversarial networks," in *Proc. Int. Conf. Mach. Learn.*, 2020, pp. 3292–3303.
[16] W. Chen, X. Gong, X. Liu, Q. Zhang, Y. Li, and Z. Wang, "FasterSeg: Searching for faster real-time semantic segmentation," in *Proc. Int. Conf. Learn. Representations*, 2019. [Online]. Available: https://arxiv.org/pdf/1912.10917.pdf
[17] H. Liu, K. Simonyan, and Y. Yang, "Darts: Differentiable architecture search," 2018, *arXiv:1806.09055*.
[18] Y. Xu et al., "Pc-darts: Partial channel connections for memory-efficient architecture search," in *Proc. Int. Conf. Learn. Representations*, 2019. [Online]. Available: https://arxiv.org/pdf/1907.05737.pdf
[19] X. Dong and Y. Yang, "Searching for a robust neural architecture in four GPU hours," in *Proc. IEEE Conf. Comput. Vis. Pattern Recognit.*, 2019, pp. 1761–1770.
[20] H. Liang et al., "DARTS: Improved differentiable architecture search with early stopping," 2019, *arXiv: 1909.06035*.
[21] M. Tan, R. Pang, and Q. V. Le, "EfficientDet: Scalable and efficient object detection," in *Proc. IEEE/CVF Conf. Comput. Vis. Pattern Recognit.*, 2020, pp. 10778–10787.
[22] H. Tanaka, D. Kunin, D. L. Yamins, and S. Ganguli, "Pruning neural networks without any data by iteratively conserving synaptic flow," 2020, *arXiv: 2006.05467*.
[23] X. Dong and Y. Yang, "NAS-bench-102: Extending the scope of reproducible neural architecture search," 2020, *arXiv: 2001.00326*.
[24] C. Ying, A. Klein, E. Christiansen, E. Real, K. Murphy, and F. Hutter, "NAS-bench-101: Towards reproducible neural architecture search," in *Proc. 36th Int. Conf. Mach. Learn.*, 2019, pp. 7105–7114. [Online]. Available: http://proceedings.mlr.press/v97/ying19a.html
[25] E. Real, A. Aggarwal, Y. Huang, and Q. V. Le, "Regularized evolution for image classifier architecture search," in *Proc. AAAI Conf. Artif. Intell.*, 2019, pp. 4780–4789.

[26] Z. Li, T. Xi, J. Deng, G. Zhang, S. Wen, and R. He, "GP-NAS: Gaussian process based neural architecture search," in *Proc. IEEE/CVF Conf. Comput. Vis. Pattern Recognit.*, 2020, pp. 11933–11942.

[27] Z. Yang et al., "HourNAS: Extremely fast neural architecture search through an hourglass lens," 2020, *arXiv: 2005.14446*.

[28] J. Xu, L. Zhao, J. Lin, R. Gao, X. Sun, and H. Yang, "KNAS: Green neural architecture search," in *Proc. Int. Conf. Mach. Learn.*, 2021, pp. 11613–11625.

[29] K. Kandasamy, W. Neiswanger, J. Schneider, B. Poczos, and E. P. Xing, "Neural architecture search with Bayesian optimisation and optimal transport," in *Proc. Adv. Neural Inf. Process. Syst.*, 2018. [Online]. Available: https://arxiv.org/pdf/1802.07191.pdf

[30] J. Yu et al., "BigNAS: Scaling up neural architecture search with big single-stage models," 2020, *arXiv: 2003.11142*.

[31] G. Li, G. Qian, I. C. Delgadillo, M. Muller, A. Thabet, and B. Ghanem, "SGAS: Sequential greedy architecture search," in *Proc. IEEE/CVF Conf. Comput. Vis. Pattern Recognit.*, 2020, pp. 1620–1630.

[32] K. Yu, R. Ranftl, and M. Salzmann, "How to train your super-net: An analysis of training heuristics in weight-sharing NAS," 2020, *arXiv: 2003.04276*.

[33] K. Yu, C. Sciuto, M. Jaggi, C. Musat, and M. Salzmann, "Evaluating the search phase of neural architecture search," in *Proc. Int. Conf. Learn. Representations*, 2020. [Online]. Available: https://arxiv.org/pdf/1902.08142.pdf

[34] D. Zhou et al., "EcoNAS: Finding proxies for economical neural architecture search," in *Proc. IEEE/CVF Conf. Comput. Vis. Pattern Recognit.*, 2020, pp. 11393–11401.

[35] D. S. Park, J. Lee, D. Peng, Y. Cao, and J. Sohl-Dickstein, "Towards NNGP-guided neural architecture search," 2020, *arXiv: 2011.06006*.

[36] A. Jacot, F. Gabriel, and C. Hongler, "Neural tangent kernel: Convergence and generalization in neural networks," in *Proc. Adv. Neural Inf. Process. Syst.*, 2018, pp. 8571–8580.

[37] B. Hanin and M. Nica, "Finite depth and width corrections to the neural tangent kernel," 2019, *arXiv: 1909.05989*.

[38] J. Lee et al., "Wide neural networks of any depth evolve as linear models under gradient descent," in *Proc. Adv. Neural Inf. Process. Syst.*, 2019, pp. 8572–8583.

[39] L. Xiao, J. Pennington, and S. S. Schoenholz, "Disentangling trainability and generalization in deep learning," 2019, *arXiv: 1912.13053*.

[40] M. Raghu, B. Poole, J. Kleinberg, S. Ganguli, and J. Sohl-Dickstein, "On the expressive power of deep neural networks," in *Proc. Int. Conf. Mach. Learn.*, 2017, pp. 2847–2854.

[41] T. Serra, C. Tjandraatmadja, and S. Ramalingam, "Bounding and counting linear regions of deep neural networks," in *Proc. Int. Conf. Mach. Learn.*, 2018, pp. 4558–4566.

[42] B. Hanin and D. Rolnick, "Complexity of linear regions in deep networks," 2019, *arXiv: 1901.09021*.

[43] H. Xiong, L. Huang, M. Yu, L. Liu, F. Zhu, and L. Shao, "On the number of linear regions of convolutional neural networks," 2020, *arXiv: 2006.00978*.

[44] Y. Jiang, B. Neyshabur, H. Mobahi, D. Krishnan, and S. Bengio, "Fantastic generalization measures and where to find them," 2019, *arXiv: 1912.02178*, 2019.

[45] Y. Lee, J. Lee, S. J. Hwang, E. Yang, and S. Choi, "Neural complexity measures," *Adv. Neural Inf. Process. Syst.*, vol. 33, pp. 9713–9724, 2020.

[46] T. Unterthiner, D. Keysers, S. Gelly, O. Bousquet, and I. Tolstikhin, "Predicting neural network accuracy from weights," 2020, *arXiv: 2002.11448*.

[47] G. Huang, Z. Liu, L. Van Der Maaten, and K. Q. Weinberger, "Densely connected convolutional networks," in *Proc. IEEE Conf. Comput. Vis. Pattern Recognit.*, 2017, pp. 2261–2269.

[48] R. Burkholz and A. Dubatovka, "Initialization of relus for dynamical isometry," in *Proc. Adv. Neural Inf. Process. Syst.*, 2019, pp. 2385–2395.

[49] S. Hayou, A. Doucet, and J. Rousseau, "On the impact of the activation function on deep neural networks training," 2019, *arXiv: 1902.06853*.

[50] Y. Shin and G. E. Karniadakis, "Trainability of relu networks and data-dependent initialization," *J. Mach. Learn. Model. Comput.*, vol. 1, no. 1, pp. 39–74, 2020.

[51] C. Wang, G. Zhang, and R. Grosse, "Picking winning tickets before training by preserving gradient flow," 2020, *arXiv: 2002.07376*.

[52] H. Li, Z. Xu, G. Taylor, C. Studer, and T. Goldstein, "Visualizing the loss landscape of neural nets," 2017, *arXiv: 1712.09913*.

[53] A. Jacot, F. Gabriel, and C. Hongler, "Neural tangent kernel: Convergence and generalization in neural networks," in *Proc. Adv. Neural Inf. Process. Syst.*, 2018. [Online]. Available: https://arxiv.org/pdf/1806.07572.pdf

[54] L. Chizat, E. Oyallon, and F. Bach, "On lazy training in differentiable programming," *Adv. Neural inf. Process. Syst.*, vol. 32, 2019.

[55] K. He, X. Zhang, S. Ren, and J. Sun, "Delving deep into rectifiers: Surpassing human-level performance on imagenet classification," in *Proc. IEEE Int. Conf. Comput. Vis.*, 2015, pp. 1026–1034.

[56] N. Cohen, O. Sharir, and A. Shashua, "On the expressive power of deep learning: A tensor analysis," in *Proc. Conf. Learn. Theory*, 2016, pp. 698–728.

[57] F. Croce, M. Andriushchenko, and M. Hein, "Provable robustness of relu networks via maximization of linear regions," in *Proc. 22nd Int. Conf. Artif. Intell. Statist.*, 2019, pp. 2057–2066.

[58] B. Hanin and D. Rolnick, "Deep relu networks have surprisingly few activation patterns," in *Proc. Adv. Neural Inf. Process. Syst.*, 2019, pp. 361–370.

[59] R. Giryes, G. Sapiro, and A. M. Bronstein, "Deep neural networks with random Gaussian weights: A universal classification strategy?," *IEEE Trans. Signal Process.*, vol. 64, no. 13, pp. 3444–3457, Jul. 2016.

[60] A. Gaier and D. Ha, "Weight agnostic neural networks," 2019, *arXiv: 1906.04358*.

[61] V. Ramanujan, M. Wortsman, A. Kembhavi, A. Farhadi, and M. Rastegari, "What's hidden in a randomly weighted neural network?," in *Proc. IEEE/CVF Conf. Comput. Vis. Pattern Recognit.*, 2020, pp. 11893–11902.

[62] K. Bhardwaj and R. Marculescu, "Towards unifying neural architecture space exploration and generalization," 2019.

[63] J. You, J. Leskovec, K. He, and S. Xie, "Graph structure of neural networks," 2020, *arXiv: 2007.06559*.

[64] S. Arora, S. S. Du, W. Hu, Z. Li, R. Salakhutdinov, and R. Wang, "On exact computation with an infinitely wide neural net," 2019, *arXiv: 1904.11955*.

[65] A. Krishnakumar, C. White, A. Zela, R. Tu, M. Safari, and F. Hutter, "Nas-bench-suite-zero: Accelerating research on zero cost proxies," in *Proc. Adv. Neural Inf. Process. Syst.*, 2022, pp. 28037–28051.

[66] Y. Shu, W. Wang, and S. Cai, "Understanding architectures learnt by cell-based neural architecture search," in *Proc. Int. Conf. Learn. Representations*, 2019. [Online]. Available: https://arxiv.org/pdf/1909.09569.pdf

[67] S. Xie et al., "Understanding the wiring evolution in differentiable neural architecture search," in *Proc. Int. Conf. Artif. Intell. Statist.*, 2021, pp. 874–882.

[68] I. Radosavovic, J. Johnson, S. Xie, W.-Y. Lo, and P. Dollár, "On network design spaces for visual recognition," in *Proc. IEEE/CVF Int. Conf. Comput. Vis.*, 2019, pp. 1882–1890.

[69] I. Radosavovic, R. P. Kosaraju, R. Girshick, K. He, and P. Dollár, "Designing network design spaces," in *Proc. IEEE/CVF Conf. Comput. Vis. Pattern Recognit.*, 2020, pp. 10428–10436.

[70] L. Wang, S. Xie, T. Li, R. Fonseca, and Y. Tian, "Sample-efficient neural architecture search by learning actions for Monte Carlo tree search," *IEEE Trans. Pattern Anal. Mach. Intell.*, vol. 44, no. 9, pp. 5503–5515, Sep. 2022.

[71] 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," in *Proc. Adv. Neural Inf. Process. Syst.*, 2020. [Online]. Available: https://arxiv.org/pdf/1911.09336.pdf

[72] B. Ru, X. Wan, X. Dong, and M. Osborne, "Interpretable neural architecture search via Bayesian optimisation with Weisfeiler-Lehman kernels," in *Proc. Int. Conf. Learn. Representations*, 2021. [Online]. Available: https://arxiv.org/pdf/2006.07556.pdf

[73] S. Yan, K. Song, F. Liu, and M. Zhang, "CATE: Computation-aware neural architecture encoding with transformers," 2021, *arXiv:2102.07108*.

[74] J. Wu et al., "Stronger NAS with weaker predictors," in *Proc. Adv. Neural Inf. Process. Syst.*, 2021, pp. 28904–28918.

[75] Y. Li, C. Hao, P. Li, J. Xiong, and D. Chen, "Generic neural architecture search via regression," in *Proc. Adv. Neural Inf. Process. Syst.*, 2021, pp. 20476–20490.

[76] R. J. Williams, "Simple statistical gradient-following algorithms for connectionist reinforcement learning," *Mach. Learn.*, vol. 8, no. 3/4, pp. 229–256, 1992.

[77] Z. Yan, X. Dai, P. Zhang, Y. Tian, B. Wu, and M. Feiszli, "FP-NAS: Fast probabilistic neural architecture search," in *Proc. IEEE/CVF Conf. Comput. Vis. Pattern Recognit.*, 2021, pp. 15134–15143.

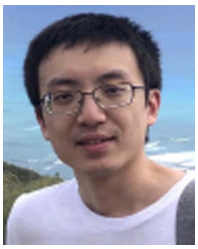
[78] L. Li and A. Talwalkar, "Random search and reproducibility for neural architecture search," in *Proc. Uncertainty Artif. Intell.*, 2020, pp. 367–377.

[79] X. Chen, R. Wang, M. Cheng, X. Tang, and C.-J. Hsieh, "DrNAS: Dirichlet neural architecture search," 2020, *arXiv: 2006.10355*.

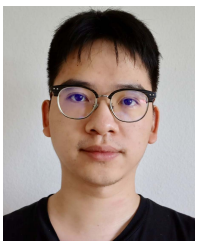
[80] X. Zhang, P. Hou, X. Zhang, and J. Sun, "Neural architecture search with random labels," in *Proc. IEEE/CVF Conf. Comput. Vis. Pattern Recognit.*, 2021, pp. 10902–10911.

[81] V. Lopes, M. Santos, B. Degardin, and L. A. Alexandre, "Guided evolution for neural architecture search," 2021, *arXiv:2110.15232*.

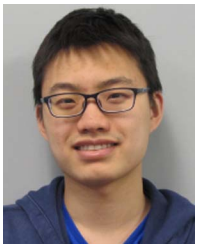
[82] P. Ye, B. Li, Y. Li, T. Chen, J. Fan, and W. Ouyang, "β-darts: Beta-decay regularization for differentiable architecture search," 2022, *arXiv:2203.01665*.
[83] P. Hou, Y. Jin, and Y. Chen, "Single-DARTS: Towards stable architecture search," in *Proc. IEEE/CVF Int. Conf. Comput. Vis.*, 2021, pp. 373–382.
[84] S. Hu, R. Wang, L. Hong, Z. Li, C.-J. Hsieh, and J. Feng, "Generalizing few-shot NAS with gradient matching," 2022, *arXiv:2203.15207*.
[85] P. Chrabaszcz, I. Loshchilov, and F. Hutter, "A downsampled variant of imagenet as an alternative to the CIFAR datasets," 2017, *arXiv:1707.08819*.
[86] B. P. Carlin and T. A. Louis, "Empirical bayes: Past, present and future," *J. Amer. Stat. Assoc.*, vol. 95, no. 452, pp. 1286–1289, 2000.
[87] B. Zoph, V. Vasudevan, J. Shlens, and Q. V. Le, "Learning transferable architectures for scalable image recognition," in *Proc. IEEE Conf. Comput. Vis. Pattern Recognit.*, 2018, pp. 8697–8710.
[88] S. Xie, H. Zheng, C. Liu, and L. Lin, "SNAS: Stochastic neural architecture search," 2018, *arXiv: 1812.09926*.
[89] H. Zhou, M. Yang, J. Wang, and W. Pan, "BayesNAS: A Bayesian approach for neural architecture search," 2019, *arXiv: 1905.04919*.
[90] H. Cai, L. Zhu, and S. Han, "ProxylessNAS: Direct neural architecture search on target task and hardware," 2018, *arXiv: 1812.00332*.
[91] Q. Yao, J. Xu, W.-W. Tu, and Z. Zhu, "Efficient neural architecture search via proximal iterations," in *Proc. AAAI Conf. Artif. Intell.*, 2020, pp. 6664–6671.
[92] X. Chen, L. Xie, J. Wu, and Q. Tian, "Progressive differentiable architecture search: Bridging the depth gap between search and evaluation," in *Proc. IEEE Int. Conf. Comput. Vis.*, 2019, pp. 1294–1303.
[93] A. Zela, T. Elsken, T. Saikia, Y. Marrakchi, T. Brox, and F. Hutter, "Understanding and robustifying differentiable architecture search," 2019, *arXiv: 1909.09656*.
[94] X. Chen and C.-J. Hsieh, "Stabilizing differentiable architecture search via perturbation-based regularization," 2020, *arXiv: 2002.05283*.
[95] Y. Zhao, L. Wang, Y. Tian, R. Fonseca, and T. Guo, "Few-shot neural architecture search," in *Proc. Int. Conf. Mach. Learn.*, 2021, pp. 12707–12718.
[96] D. Han, J. Kim, and J. Kim, "Deep pyramidal residual networks," in *Proc. IEEE Conf. Comput. Vis. Pattern Recognit.*, 2017, pp. 5927–5935.
[97] M. Tan et al., "MnasNet: Platform-aware neural architecture search for mobile," in *Proc. IEEE Conf. Comput. Vis. Pattern Recognit.*, 2019, pp. 2815–2823.
[98] H. Cai, C. Gan, T. Wang, Z. Zhang, and S. Han, "Once-for-all: Train one network and specialize it for efficient deployment," 2019, *arXiv: 1908.09791*.



Wuyang Chen received received BS degree from the University of Science and Technology of China, in 2014, the MS degree in computer science from Rice University, in 2016, and the PhD degree in electrical and computer engineering with the University of Texas at Austin. His research focuses on addressing domain adaptation/generalization, self-supervised learning, and AutoML.

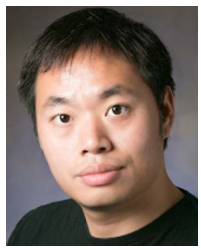


Xinyu Gong received the bachelor's degree in computer science from the University of Electronic Science and Technology of China, in 2018 and the PhD degree in electrical and computer engineering with the University of Texas at Austin. His research interests are broadly in computer vision and machine learning, with a recent focus on neural architecture search.



Junru Wu received the BS degree from Tongji University and the PhD degree from the Department of Computer Science and Engineering at Texas, A&M University. He's interned at industry research labs including Google Research, Microsoft Research, NEC Labs America, and ByteDance AI Lab. His research interests lie in the intersection of computer vision and machine learning. In particular, he is interested in enabling efficient machine learning in a broad spectrum of computer vision problems, which includes Low-level Vision, Neural Architecture Search, and Multimodal Understanding.

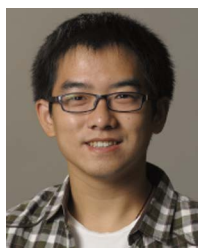
Yunchao Wei received the PhD degree from Beijing Jiaotong University, Beijing, China, in 2016. He is currently a professor with the Center of Digital Media Information Processing, Institute of Information Science, Beijing Jiaotong University. He was a Postdoctoral Researcher at Beckman Institute, UIUC, from 2017 to 2019. He is ARC Discovery Early Career Researcher Award Fellow from 2019 to 2021. His current research interests include computer vision and machine learning.



Humphrey Shi (Senior Member, IEEE) is currently an associate professor of interactive computing with Georgia Institute of Technology. He is also a graduate faculty member of computer science with the University of Oregon and the Department of Electrical and Computer Engineering, University of Illinois at Urbana-Champaign. Outside of academia, he is the chief scientist of Picsart AI Research (PAIR). His research interests include computer vision, machine learning, AI systems and applications, and creative, efficient, and responsible multimodal AI.

Zhicheng Yan received the PhD from the Department of Computer Science, University of Illinois at Urbana-Champaign, in 2016. He is a senior staff research scientist with Meta Reality Labs. He has been building and delivering cutting-edge on-device solutions to power the perception stack for Meta MR/VR products. Before, Zhicheng was a Senior Manager supporting an applied research team to develop a deep and personalized understanding of the objects and scenes in the egocentric data for next-generation Meta AR products. In the early stage of his career at Facebook, he worked on large-scale image and video understanding platform.

Yi Yang (Senior Member, IEEE) received the PhD degree from Zhejiang University, Hangzhou, China, in 2010. He is currently a professor with the Zhejiang University. He was a post-doctoral researcher with the School of Computer Science, Carnegie Mellon University. His current research interests include machine learning and its applications to multimedia content analysis and computer vision, such as multimedia retrieval and video content understanding.



Zhangyang Wang (Senior Member, IEEE)is a tenured associate professor and holds the Temple Foundation Endowed Faculty Fellowship, in the Chandra Family Department of Electrical and Computer Engineering, The University of Texas at Austin. He has broad research interests spanning from the theory to the application aspects of machine learning. He has received many research awards, including an NSF CAREER Award, an ARO Young Investigator Award, an IEEE AI's 10 To Watch Award, an INNS Aharon Katzir Young Investigator Award, a Google Research Scholar award, an IBM Faculty Research Award, a J. P. Morgan Faculty Research Award, an Amazon Research Award, an Adobe Data Science Research Award, a Meta Reality Labs Research Award, and two Google TensorFlow Model Garden Awards.