Progressive Neural Architecture Generation

Caiyang Yu[1] Chen Huang[2] Yun Liu[3] Chenwei Tang[3] Wei Ju[3,†] Jiancheng Lv[3,†]
[1]Wenzhou University [2]National University of Singapore [3]Sichuan University
yucy324@gmail.com huang_chen@nus.edu.sg yliu@stu.scu.edu.cn
{tangchenwei, juwei, lvjiancheng}@scu.edu.cn

Abstract

*As a representative technique in neural architecture search, neural architecture generation aims to construct high-performance architectures for a given task directly. It is poised to replace the inefficient random exploration components of some search strategies, such as the acquisition strategies in Bayesian optimization. Despite significant research, current architecture generation techniques face problems such as low generation efficiency and insufficient constraints, leading to invalidly generated architectures. To this end, we propose Progressive Neural Architecture Generation (**PNAG**), which constructs architectures incrementally through coarse-to-fine evolution, enhancing generation efficiency, and incorporates step-wise refinements to ensure the validity of the generated architecture. To achieve this, **PNAG** involves two modules, multi-scale sub-architecture quantization (MSQ) and step-wise consistency constraint (SCC). Specifically, MSQ constructs sub-architectures using quantization decoding and progressively expands them, transitioning from simple to complex forms. This operation bypasses network inference to enhance efficiency. Complementing MSQ, SCC, implemented through a tailored regularization mechanism, introduces penalties for deviations during sub-architecture generation, guiding the process towards valid target architectures. As such, **PNAG** establishes a clear generation path, laying the groundwork for generating suitable architectures in downstream tasks. Extensive experiments demonstrate that **PNAG** not only generates superior architectures for various downstream tasks (+8.43%/+5.07%, on average) but also significantly improves generation efficiency, reducing the architecture generation time by 1300×. Furthermore, **PNAG** demonstrates strong extensibility by successfully generating Transformer-based architectures.*

1. Introduction

While search strategies endow Neural Architecture Search (NAS) with automated exploration capabilities [39, 47, 49], they still harbor subtle and often overlooked pitfalls. For instance, random initialization in evolutionary algorithms or Bayesian optimization may inadvertently introduce uncertainty and bias into the search process. To overcome these challenges, Neural Architecture Generation (NAG) [1] emerges as a novel paradigm, capable of directly constructing high-quality candidate architectures for a task-specific search process, thereby obviating the need for random or uncertain exploration steps (*c.f.*, Fig. 1). Recent studies have demonstrated this potential through graph-based generative approaches. Hemmi [23] proposes a graph Variational AutoEncoder (VAE)-based NAG method, where the final architecture is generated by sampling in the graph latent space through a complex decoder, while An [1] proposes a graph diffusion model and generates the final architecture by multiple iterative denoising in the latent space.

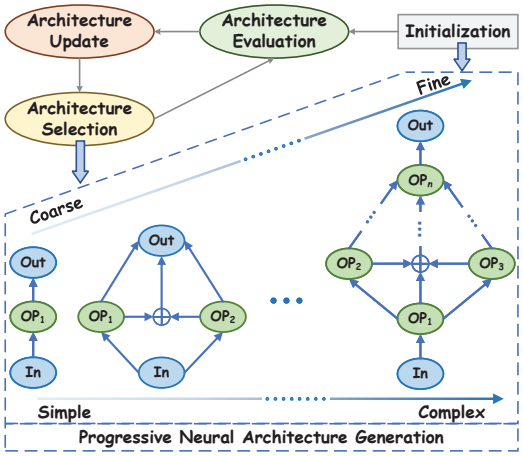

Figure 1. **PNAG** can replace any stochastic step in the NAS pipeline, such as random initialization and sampling.

Despite considerable progress, current generation methods still face limitations in two key aspects. The first issue is ***Generation Efficiency***, which reflects the computational cost required to produce valid architectures. Existing generation techniques often involve complex and computation-intensive procedures, such as reconstructing architectures through de-

[†]Corresponding authors.

coders [23] or iteratively refining outputs via neural layers [1, 2]. These approaches rely on repeated transformations in high-dimensional latent spaces through network inference, which significantly increases computational overhead and limits the efficiency of the generation process. The second issue concerns ***Architectural Validity***, referring to whether the generated architectures can be properly trained and executed. Current generation methods typically impose validity constraints only on the final output architecture while neglecting intermediate-stage supervision, which is essential for maintaining structural correctness throughout the generation process. Although recent approaches, such as diffusion-based generation [1], attempt to guide the process incrementally, they still lack architectural-level supervision and thus fail to guarantee the validity of the resulting architectures.

To this end, we propose Progressive Neural Architecture Generation (**PNAG**), which constructs architectures incrementally through coarse-to-fine linear evolution, enhancing generation efficiency, and incorporates step-wise refinements to ensure the validity of the generated architecture (*c.f.*, Fig. 1). **PNAG** reformulates the generation process as an autoregressive (AR) learning process on architectures, where each AR unit is a fully functional sub-architecture, gradually evolving from a simple (coarse) to a complex (fine) form. This process is driven by two key modules: multi-scale sub-architecture quantization (MSQ) and step-wise consistency constraint (SCC). In particular, MSQ employs quantization decoding [44] to obtain structural elements and progressively enriches the sub-architecture by increasing the number of elements, thereby transforming the sub-architecture from a simple to a complex form until the final architecture is generated. Since the generation process of the sub-architecture relies solely on linear generation, the computation time can be significantly reduced. Complementing MSQ, SCC employs a tailored regularization mechanism that penalizes deviations during sub-architecture generation, ensuring the structural rationality of each intermediate sub-architecture and guiding the process towards a valid target architecture. Unlike existing generation methods, **PNAG** is a tailored architecture generation approach with a discrete generation process that aligns better with architectural properties.

We conduct extensive experiments on several benchmark datasets, *i.e.*, NAS-Bench-201 [15], MobileNetV3 [5], and DARTS [29], to evaluate the generation efficiency and architectural validity of **PNAG**. The result shows that compared to baselines, **PNAG** not only generates superior architectures for various downstream tasks (+8.43%/+5.07%, on average) but also significantly improves generation efficiency, reducing the time to a single generation of the architecture by 1300×. Furthermore, we extend **PNAG** to generate Transformer-based architectures and achieve similarly strong performance. In summary, our contributions are as follows. ❶ ***Framework Design.*** Calling attention to the generation efficiency and architectural validity, we propose a pioneering coarse-to-fine approach, named **PNAG**, which reformulates the generation process as an autoregressive process and gradually evolves from simple (coarse) to complex (fine). ❷ ***Core Modules.*** We propose a multi-scale sub-architecture quantization to progressively generate sub-architectures via vector quantization decoding, significantly improving efficiency. Additionally, a step-wise consistency constraint ensures structural rationality, guiding the process towards a valid target architecture. ❸ ***Empirical Results.*** We experimentally show the superiority of **PNAG** in terms of both generation efficiency and architectural validity compared to state-of-the-art methods. Moreover, **PNAG** also demonstrates strong performance when applied to Transformer-based architecture generation, highlighting its extensibility.

2. Related Work

Neural Architecture Search. Neural Architecture Search (NAS) [35, 36, 53] aims to automatically design neural network architectures for given tasks, thereby reducing the labor-intensive trial-and-error process and reliance on expert knowledge inherent in manual design. NAS methods have achieved network architectures that match or even surpass manually designed counterparts in performance for certain tasks. From the perspective of search strategies, existing methods can be categorized into reinforcement learning-based approaches [6, 54], evolutionary algorithm-based approaches [28, 34], and Bayesian optimization-based approaches [45]. However, most existing studies focus on improving the overall performance of search algorithms while overlooking fine-grained operational details. In particular, elements such as random initialization or other stochastic components are often treated as minor implementation choices, yet they can introduce randomness and bias that undermine the stability and reliability of the obtained architectures. To address these issues, this paper adopts Neural Architecture Generation (NAG) to replace such components, aiming to more effectively mitigate these problems.

Neural Architecture Generation. Neural Architecture Generation (NAG) [1, 20] aims to leverage diverse generative techniques to construct high-performance architectures tailored to specific tasks. For example, the variational autoencoder (VAE)-based generation method maps architectures into a continuous latent space, utilizing a complex decoder to transform high-dimensional data within the latent space to reconstruct the architecture [23, 32, 51]. Additionally, the diffusion model-based generation introduces noise to perturb the architecture and gradually refines it through an iterative denoising process [1, 2, 18]. However, despite considerable progress, these methods still face limitations in terms of generation efficiency and architectural validity. To this end, we introduce linear generation and step-wise constraints into the generation process to address the aforementioned issues.

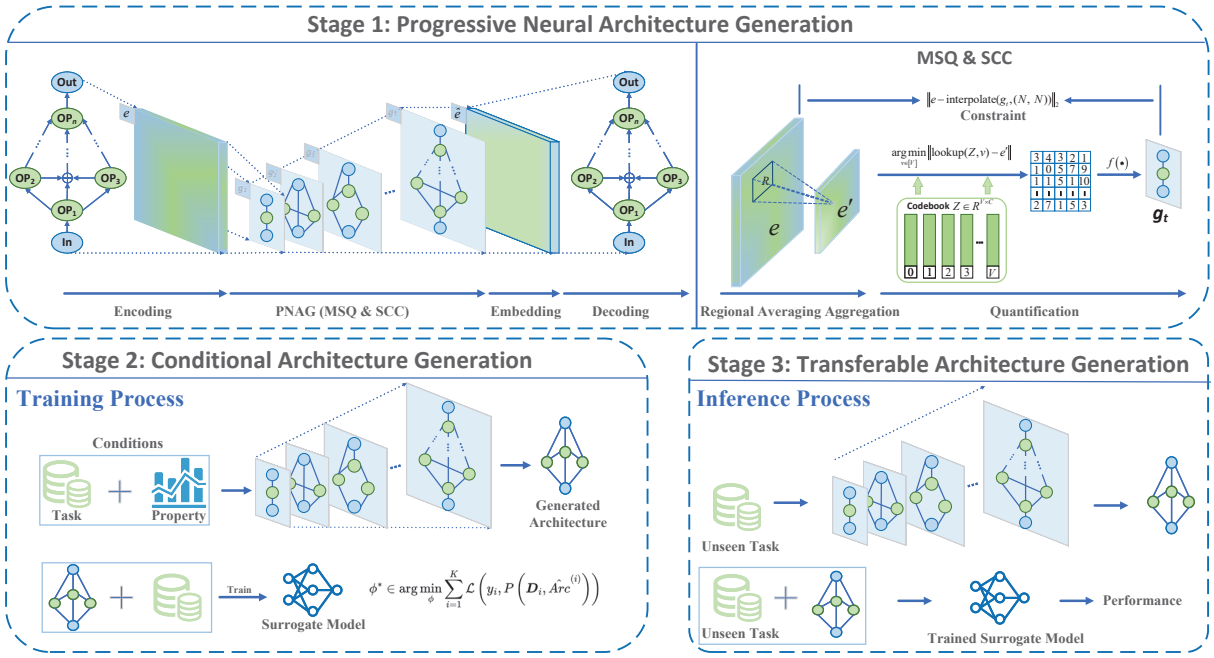

Figure 2. The overall framework. **Stage 1:** The **PNAG** quantifies the encoded architecture into a sub-architecture sequence $G = (g_1, g_2, \cdots, g_T)$ and trained with the reconstruction loss Eq. 7. "Embedding" in the figure means converting discrete operations into continuous embedding vectors. The process of generating and constraining each sub-architecture is described in the MSQ & SCC module. **Stage 2:** Generate the architecture based on the conditions and train the surrogate model. **Stage 3:** Extending **PNAG** to transferable NAG for architecture generation on unseen tasks.

3. Method

To address both generation efficiency and architectural validity, we introduce **PNAG**, a novel approach that leverages linear generation to reduce computational overhead while incorporating constraints during the generation process. This section begins with preliminary information (Sec. 3.1), followed by a framework overview (Sec. 3.2). We then detail **PNAG**'s technical components and applications (Sec. 3.3 to Sec. 3.5). Since **PNAG** mirrors the autoregressive (AR) model, we first provide a brief overview.

3.1. Preliminary on Autoregressive Model

The AR model is a statistical model used to predict the next token based on the previous tokens in the sequence [4, 19]. Formally, given a discrete token sequence $X = \{x_1, x_2, \cdots, x_T\}$ of length T, the goal of the AR model is to learn a conditional probability distribution $p(x_t \mid x_1, x_2, \ldots, x_{t-1})$ at each time step t. By this means, we can sample from an AR generative model over an indexed sequence X where the probability distribution is defined as:

$$p(x_1, x_2, \ldots, x_T) = \prod_{t=1}^{T} p(x_t \mid x_1, x_2, \ldots, x_{t-1}), \quad (1)$$

where the token x_t may represent a word in natural language processing [10], or pixel value [43] in the visual domain. In our approach, each AR unit represents a complete, functional sub-architecture, which is inspired by [43]. This establishes a coarse-to-fine architecture generation process.

3.2. Overview

PNAG generates neural architectures using a coarse-to-fine approach, as illustrated in Fig. 2. This is accomplished through multi-scale sub-architecture quantization (MSQ) and step-wise consistency constraint (SCC). In particular, MSQ employs efficient vector quantization decoding, tailored for architecture generation, to autoregressively predict and generate subsequent sub-architectures (Sec. 3.3). Meanwhile, SCC ensures the validity of each generated sub-architecture, and consequently the target architecture, by applying a specialized step-wise regularization mechanism (Sec. 3.4). Notably, **PNAG** is designed for extensibility and can be integrated with other optimization strategies to generate meaningful architectures (*c.f.*, Fig. 1), as in Sec. 3.5.

3.3. Multi-scale Sub-architecture Quantization

Rather than directly generating the target architecture, **PNAG** reframes autoregressive modeling by predicting the next

scale of the sub-architecture, instead of the next token. MSQ achieves this by iteratively increasing the number of elements (*i.e.,* layers or blocks) in the sub-architecture selected through vector quantization decoding [44], progressively enhancing the complexity of the sub-architectures while maintaining efficient generation. Given an architecture Arc, MSQ adopts the VAE training paradigm for encoding but employs an autoregressive decoding process.

Sub-architecture Feature Map. We encode the original architecture Arc into a feature map e through a graph encoder. This feature map is then scaled and decoded into intermediate sub-architectures (*i.e.*, trainable neural architectures containing n operations) by quantization operations. As illustrated in Fig. 2 (stage 1), we employ regional averaging to aggregate features and proportionally scale each point. Formally, given a feature map $e \in \mathbb{R}^{C\times N\times N}$ to be scaled to $e' \in \mathbb{R}^{C\times n\times n}$, where $n < N$ and N denotes the maximum number of layers or blocks in the predefined set. Notably, the minimum value of n is 3, representing the simplest sub-architecture composed of three operations: input, output, and one functional operation. Each feature point in $e'_{c,i,j}$ is computed as the average of its corresponding region in e:

$$e'_{c,i,j} = \frac{1}{|R_{i,j}|} \sum_{(h,w)\in R_{i,j}} e_{c,h,w}, \quad e' = \sum_{c=1}^{C}\sum_{i=1}^{n}\sum_{j=1}^{n} e'_{c,i,j}, \tag{2}$$

where $R_{i,j}$ denotes a region in the original feature map e that is compressed through a linear mapping into the position (i, j) in the new feature map e', and we define (h, w) as the position of the feature in $R_{i,j}$. Note that n increases with each generation step, ultimately reaching N.

Sub-architecture Quantization. We utilize vector quantization for efficient decoding, employing a shared codebook across all sub-architecture decoding steps. Specifically, the scaled feature map e' is quantified and mapped to its corresponding sub-architecture. This quantization process employs a learnable and shared codebook $Z \in \mathrm{R}^{V\times C}$ of size V (equal to the number of available structural elements in the predefined set, *i.e.*, N) [25]. Here, C denotes the vector dimension. In this case, when generating the t-th sub-architecture g_t, quantization is performed by identifying the nearest distance arrival to Z for each vector in e' based on minimum Euclidean distance, yielding a code index $o = \{o_1, o_2, \cdots, o_n\}$. This code index is then decoded into corresponding structural elements and assembled to form the sub-architecture g_t:

$$o = \left(\arg\min_{v\in[V]} \|\mathrm{lookup}(Z, v) - e'\|_2 \right), \quad g_t = \bigcup_{i=1}^{n} f\left(o_i\right), \tag{3}$$

where $\mathrm{lookup}(Z, v)$ denotes taking the v-th vector in the codebook Z, f is a linear decoding function that maps the code index o_i to the corresponding structure. Thus, we reformulate Eq. 1 as:

$$p\left(e'_1, e'_2, \ldots, e'_T\right) = \prod_{t=1}^{T} p\left(e'_t \mid e'_1, e'_2, \ldots, e'_{t-1}\right), \tag{4}$$

where T is the maximum generation step.

To sum up, we illustrate the training process in Algorithm 1, where $\mathcal{E}$ is the graph encoder, $\mathcal{Q}$ denotes the quantization decoding, $\mathrm{RAA}(\cdot)$ denotes region averaging aggregation, and $\mathrm{interpolate}(\cdot)$ denotes linearly interpolating the feature map of the sub-architecture to expand it into an $N \times N$ feature map. The accumulated feature map $\hat{e}$ is finally obtained and decoded by $\mathcal{D}$ to reconstruct the final architecture. Notably, in the MSQ generation process, both codebook lookup and mapping functions operate independently of network inference, employing simple linear operations that significantly reduce generation time.

Algorithm 1 Multi-scale Sub-architecture Quantization

Inputs: original architecture Arc
Hyperparameters: steps T, scale n
$e = \mathcal{E}(Arc), G = \{\}, \hat{e} = 0$
for $t = 1, \ldots, T$ **do**
  $e'_t = \mathrm{RAA}(e, n)$
  $g_t = \mathcal{Q}(e'_t)$
  $G = \mathrm{queue\_push}(G, g_t)$
  $\hat{e}_t = \mathrm{interpolate}(e'_t, N)$
  $\hat{e} = \hat{e} + \hat{e}_t$
  $e = e - \hat{e}_t$
  $n = n + 1$
end for
$\hat{Arc} = \mathcal{D}(\hat{e})$
Return: sub-architecture sequence G, reconstructed feature map $\hat{e}$, reconstructed architecture $\hat{Arc}$

3.4. Step-wise Consistency Constraint

Unlike existing generation methods, architecture validity constraints are typically only applied at the final stage of generation [1]. To address this limitation, we propose a step-wise consistency constraint strategy, which is designed to impose constraints on every step of the generation process. By applying a regularization term to penalize inconsistencies from the original input architecture, **PNAG** ensures the validity of each generated sub-architecture, leading to a structurally sound target architecture.

In details, existing approaches typically apply a loss function at the final step [43, 44], which measures the discrepancy between the generated architecture $\hat{Arc}$ and the input architecture Arc:

$$\mathcal{L} = \|Arc - \hat{Arc}\|_2 + \|e - \hat{e}\|_2. \tag{5}$$

However, such constraints overlook the intermediate generation steps, potentially allowing error accumulation and

degraded structural fidelity.To address this limitation, we introduce a step-wise consistency constraint (SCC), which enhances architectural validity by enforcing consistency throughout the entire generation process. SCC adds a regularization term at each intermediate step to penalize deviations between the generated sub-architecture and the encoded representation of the input architecture. Concretely, for each generated feature map e'_t of sub-architecture at step t, we first resize it to match the dimension of e via bilinear interpolation and then apply a consistency penalty:

$$\mathcal{R}_{SCC} = \lambda \sum_{t=1}^{T} \|e - \text{interpolate}(e'_t, (N, N))\|_2 , \quad (6)$$

where λ is a regularization coefficient and T is the total number of generation steps. Finally, the overall training objective combines both the final-step and step-wise constraints:

$$\mathcal{L} = \|Arc - \hat{Arc}\|_2 + \|e - \hat{e}\|_2 + \mathcal{R}_{SCC}. \quad (7)$$

This unified objective enables **PNAG** to maintain structural consistency across the entire generation trajectory, leading to more valid and coherent architectures.

To establish the efficacy of SCC, we utilize the Lyapunov Stability theorem [38] to demonstrate that the intermediate sub-architecture generated at each step gradually approaches the input, ensuring a stable generation path and reducing accumulated deviation. More details are in the Appendix.

Proposition 3.1 *Asymptotic stability of the sub-architecture generation trajectory is guaranteed if the learning rate satisfies the condition* $\alpha < -\frac{2\left(e-e'_t\right)^T \nabla_{e'_t} \mathcal{L}\left(e'_t\right)}{\left\|\nabla_{e'_t} \mathcal{L}(e'_t)\right\|_2^2}$.

[Proof Sketch] 3.1 *We define the following Lyapunov function $V(x)$, measured by the L2 norm. This ensures stable convergence of the generated architecture towards the input by progressively minimizing the deviation.*

$$V\left(e'_t\right) = \|e - \textit{interpolate}(e'_t, (N, N))\|_2 . \quad (8)$$

For simplicity, we represent this equation as:

$$V\left(e'_t\right) = \|e - e'_t\|_2^2 . \quad (9)$$

This Lyapunov function describes the "deviation energy" between the intermediate sub-architecture and e. In this case, a smaller $V\left(e'_t\right)$ indicates greater stability in the generation process. In a discrete generation process, we approximate the derivative of the Lyapunov function with the difference form:

$$\Delta V\left(e'_t\right) = V\left(e'_{t+1}\right) - V\left(e'_t\right). \quad (10)$$

A stable generation path is guaranteed if $\Delta V\left(e'_t\right) < 0$, indicating a decrease in deviation energy at each step. Substituting the update equation $e'_{t+1} = e'_t - \alpha \nabla_{e'_t} \mathcal{L}\left(e'_t\right)$ into Eq. 6 yields:

$$\Delta V\left(e'_t\right) = \left\|e - \left(e'_t - \alpha \nabla_{e'_t} \mathcal{L}\left(e'_t\right)\right)\right\|_2^2 - \|e - e'_t\|_2^2 . \quad (11)$$

In this scenario, $\Delta V\left(e'_t\right) < 0$ holds true if α meets the following conditions:

$$\alpha < -\frac{2\left(e - e'_t\right)^T \nabla_{e'_t} \mathcal{L}\left(e'_t\right)}{\left\|\nabla_{e'_t} \mathcal{L}\left(e'_t\right)\right\|_2^2}. \quad (12)$$

This implies that with a suitably chosen learning rate α, the Lyapunov function monotonically decreases, guaranteeing the asymptotic stability of the generation trajectory.

3.5. Application of Method

PNAG is designed for extensibility and can be integrated with other optimization strategies to generate meaningful architectures. This section demonstrates the applications of **PNAG** in the contexts of conditional and transferable generation.

In basic NAG, the generation process is unconditional, aiming to explore diverse architectural possibilities. However, in practical scenarios, it is often desirable to conditionally generate architectures that satisfy specific performance attributes (*e.g.*, high accuracy, low latency) according to the needs of a given task. To achieve this, we introduce a surrogate model and formulate its training as a meta-learning task. Specifically, instead of learning a fixed mapping $P(y|\boldsymbol{D}, \hat{Arc})$ on a single task $\boldsymbol{D}$, the model is trained on a series of diverse tasks to acquire a more generalizable meta-knowledge of "how to evaluate an architecture's quality". This not only guides the model in generating architectures with the desired performance for known tasks, but more critically, it can also directly make reliable predictions on the performance of architectures on new, unseen tasks $\tilde{\boldsymbol{D}}$, thereby efficiently achieving transferability (detailed in Fig. 2(stage2 and stage 3)). Overall, the objective for our transferable **PNAG** is formally defined below:

$$p\left(g_1, g_2, \ldots, g_T \mid P(y \mid \tilde{\boldsymbol{D}}, \hat{Arc})\right) = \prod_{t=1}^{T} p\left(g_t \mid g_1, g_2, \ldots, g_{t-1}, P(y \mid \tilde{\boldsymbol{D}}, \hat{Arc})\right). \quad (13)$$

4. Experiments

4.1. Experimental Setup

Search Space and Datasets. To facilitate a comprehensive evaluation of our proposed method, we explicitly separate the discussion of search space and datasets, as they play distinct roles in the architecture generation pipeline: the search space defines the design scope of architectures, while the datasets determine the tasks on which the generated architectures are evaluated. We evaluate **PNAG** on three widely adopted search spaces: NAS-Bench-201 (NB201) [15], MobileNetV3 (MBV3) [5], and DARTS [29]. Additionally, we extend **PNAG** to perform Transformer-based architecture generation on the AutoFormer search space [7], which is

Table 1. **Main Results on NB201 Search Space.** Reported accuracies represent the mean value (%) with a 95% confidence interval over three independent runs. We also provide the number of neural architectures trained (Trained Archs) to achieve the reported accuracy. The bold is the optimal result.

Method	CIFAR-10		CIFAR-100		Aircraft		Oxford-IIIT Pets		Optimization
	Accuracy	Trained Archs	Accuracy	Trained Archs	Accuracy	Trained Archs	Accuracy	Trained Archs	
RSPS	84.07±3.61	N/A	52.31±5.77	N/A	42.19±3.88	N/A	22.91±1.65	N/A	Random
RFGIAug	94.25±0.00	5	-	-	-	-	-	-	Evolution
HAAP	94.20±0.25	10	71.58±1.56	10	55.32±0.12*	10	38.61±0.20*	10	Evolution
OStr-DARTS	94.36±0.00	N/A	73.51±0.00	N/A	-	-	-	-	Gradient
SETN	87.64±0.00	N/A	59.09±0.24	N/A	44.84±3.96	N/A	25.17±1.68	N/A	Gradient
GDAS	93.61±0.09	N/A	70.70±0.30	N/A	53.52±0.48	N/A	24.02±2.75	N/A	Gradient
PC-DARTS	93.66±0.17	N/A	66.64±2.34	N/A	26.33±3.40	N/A	25.31±1.38	N/A	Gradient
DrNAS	94.36±0.00	N/A	73.51±0.00	N/A	46.08±7.00	N/A	26.73±2.61	N/A	Gradient+BO
BOHB	93.61±0.52	> 500	70.85±1.28	> 500	-	-	-	-	BO
GP-UCB	**94.37±0.00**	58	73.14±0.00	100	41.72±0.00	40	40.60±1.10	11	BO
BANANAS	**94.37±0.00**	46	**73.51±0.00**	88	41.72±0.00	40	40.15±1.59	17	BO
NASBOWL	94.34±0.00	100	**73.51±0.00**	87	53.73±0.83	40	41.29±1.10	17	BO
HEBO	94.34±0.00	100	72.62±0.20	100	49.32±6.10	40	40.55±1.15	18	BO
TNAS	**94.37±0.00**	29	**73.51±0.00**	59	59.15±0.58	26	40.00±0.00	6	Transferable
MetaD2A	**94.37±0.00**	100	73.34±0.04	100	57.71±0.20	40	39.04±0.20	40	Transferable
DiffusionNAG	**94.37±0.00**	**1**	**73.51±0.00**	2	58.83±3.75	3	41.80±3.82	2	Transferable
PNAG (Ours)	**94.37±0.00**	**1**	**73.51±0.00**	**1**	**66.99±1.25**	**1**	**45.35±0.78**	**2**	Transferable

† denotes that the results are taken from [1], * denotes that the results are implemented by ourselves.

Table 2. **Main Results on MBV3.** Reported accuracies represent the mean value with a 95% confidence interval over three independent runs. Following [1], each method generates 30 architectures on the dataset per run.

Method	CIFAR10	CIFAR100	Aircraft	Oxford IIIT Pets
MetaD2A	97.45±0.07	86.00±0.19	82.18±0.70	95.28±0.50
TNAS	97.48±0.14	85.95±0.29	82.31±0.31	95.04±0.44
DiffusionNAG	97.52±0.07	86.07±0.16	82.28±0.29	95.34±0.29
PNAG (Ours)	**97.67±0.05**	**87.20±0.20**	**84.55±0.17**	**95.77±0.11**

designed to evaluate the NAS methods specialized to Vision Transformers. Following the protocol in [1], we evaluate **PNAG** on four representative image classification datasets that span different levels of difficulty and domain characteristics: CIFAR-10, CIFAR-100, Aircraft [33], and Oxford-IIIT Pet [37]. These datasets serve as downstream tasks to validate the generalizability of the generated architectures.

Baselines. We compare **PNAG** against a variety of NAS methods, encompassing: RSPS [27], RFGIAug [48], HAAP [30], ModuleNet [9], GPT-NAS [53], DARTS [29], OStr-DARTS [52], SETN [13], GDAS [14], PC-DARTS [50], IS-DARTS [21] and DrNAS [8], BOHB [17], GP-UCB, BANANAS [46], NASBOWL [40], HEBO [11], NASNet-A [55], MetaQNN [3], NASI [42], RoBoT [22], MetaD2A [26], TNAS [41], and DiffusionNAG [1].

Implementation Details. We divide the **PNAG** framework into two stages. The first involves training the VAE model, where we use a simple graph encoder to encode the architecture and two linear layers to decode it. No additional model is used during the architecture AR generation process. Second, in the conditional generation model, we follow [43] and adopt a standard decoder-only transformer similar to GPT-2 and VQGAN [16] to process the conditional input, replacing traditional layer normalization with adaptive normalization (AdaLN) [24]. During training, we use the AdamW optimizer [31], with learning rates of 0.0001 and 0.001 for the two stages, weight decay of 0.0001, batch size of 256, and train for 300 epochs. More details are provided in the Appendix.

Evaluation Metrics. Following [1], we assess generated architectures via: 1) Overall performance is evaluated using task accuracy (Accuracy) and the number of architectures trained to achieve that accuracy (Train Arcs). 2) Generation efficiency is measured by the single time to generate architectures (Once Time) and the total time to find the task-specified optimal architecture (Total time). 3) Validity and quality are assessed via the proportion of valid architectures (Validity), the percentage of unique valid architectures (Uniqueness), and the proportion of valid architectures not present in the training set (Novelty).

4.2. Empirical Results

Main Results. We conduct evaluations across three distinct search spaces, **PNAG** consistently achieves superior generation performance and architectural validity across all three settings. ❶ Given the established optimal accuracy results for CIFAR-10 and CIFAR-100 within the NB201 search space (Table 1), we find that **PNAG** successfully generates optimal architectures. Furthermore, **PNAG** demonstrates significant performance gains on the Aircraft and Oxford-IIIT Pets datasets, achieving average accuracy im-

Table 3. **Main results on DARTS.** Test accuracy on image datasets.

Methods	CIFAR-10 / CIFAR-100		ImageNet	Search Method
	Test Acc (%)	Search Cost (GPU Day)	Top-1/Top-5 Acc (%)	
NASNet-A	97.35 / 82.19	1800	74.0 / 91.6	RL
MetaQNN	93.08 / 72.86	80	- / -	RL
ModuleNet	97.23 / 82.01	2.00	- / -	Evolution
GPT-NAS	97.35 / 83.51	1.20	75.1 / 92.3	Evolution
DARTS	97.35 / 82.24	1.00	73.3 / 91.3	Gradient
OStr-DARTS	97.58 / 84.22	0.40	**76.2 / 93.0**	Gradient
IS-DARTS	97.44 / -	0.42	75.9 / 92.9	Gradient
NASI	97.10 / 83.16	0.24	75.0 / 92.5	Training-free
RoBoT	97.40 / 83.48	3.50	75.9 / 92.7	Training-free
DiffusionNAS	97.35 / 82.91	0.04	75.0 / 92.6	Generation
PNAG	**97.93 / 83.54**	**0.03**	75.9 / **93.0**	Generation

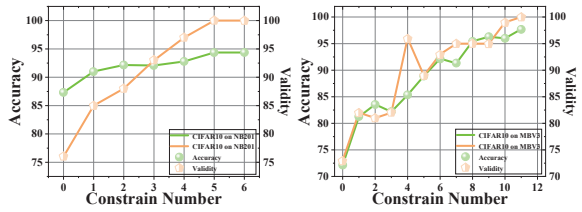

Figure 3. **Analysis on the influence of SCC**. Results of the different number of constraints on architectural performance (left axis) and architectural validity (right axis).

provements of 8.43% and 5.07%, respectively, compared to other transferable-based baselines. More importantly, **PNAG** exhibits remarkable efficiency, requiring the training of at most two architectures across all four datasets, significantly fewer than others. ❷ On MBV3 (Table 2), compared to the suboptimal method, **PNAG** achieves an average improvement of 0.99%. Especially on the Aircraft dataset, **PNAG** achieves the greatest improvement, obtaining an accuracy improvement of 2.24%. Furthermore, except for CIFAR-100, the performance of architectures generated by **PNAG** across multiple runs demonstrates superior stability. This highlights the effectiveness and robustness of our proposed method. ❸ In the DARTS search space (Table 3), **PNAG** achieves the best overall performance, with the highest test accuracy on CIFAR-10 (97.93%) and CIFAR-100 (83.54%), and the highest Top-5 accuracy on ImageNet (93.0%). Moreover, it attains these results with the lowest search cost, demonstrating strong effectiveness and efficiency. These results validate the general applicability of **PNAG** across search spaces of varying complexity and design principles.

Transformer-Based Generation Results. We evaluate **PNAG** in the AutoFormer search space to generate Transformer-based (ViT) models of different sizes. As shown in Table 4, **PNAG** achieves the best accuracy (76.6%) under the **Tiny** setting, outperforming all baselines while maintaining a comparable parameter count (6.03M). Under the **Base** setting, **PNAG** matches the accuracy of AutoFormer

Table 4. **Results on AutoFormer.** We report the Top-1 accuracy of generated models with different sizes on ImageNet.

Methods	**Tiny**		**Base**	
	Acc (%)	#Params	Acc (%)	#Params
AutoFormer	74.7	**5.70M**	**82.4**	54.0M
TF-NAS	75.3	6.20M	82.2	56.5M
AZ-NAS	76.4	6.16M	82.3	54.1M
PNAG	**76.6**	6.03M	**82.4**	**53.8M**

Table 5. **Adaptation ability across different objectives.**

Type	State	f_{Clean}	f_{APGD}	f_{Blur}
Clean	Max	**92.17**	82.27	87.19
	Mean	**91.35**	75.21	79.39
APGD	Max	7.43	**34.52**	29.18
	Mean	3.95	**32.61**	26.43
Blur	Max	35.40	40.06	**48.37**
	Mean	31.77	35.31	**45.11**

and TF-NAS (82.4%) with the smallest model size (53.8M parameters), demonstrating a strong performance-efficiency trade-off. These results demonstrate that **PNAG** is not only effective in convolutional architecture generation, but also excels in generating high-performing Transformer-based (ViT) models across different model scales.

Ablation Study. We evaluated the impact of SCC's stepwise regularization constraints on the autoregressive generation process in the NB201 and MBV3 search spaces. Figure 3 shows the results for varying step numbers, where 0 indicates no regularization, and a step number of t means constraints are applied during the first t steps. The lowest architectural validity occurs with no constraints (step 0), with both validity and performance improving as more steps are constrained. This demonstrates that SCC effectively ensures structural validity by maintaining consistency throughout the generation process, validating the efficacy of SCC.

Adaptation Across Different Objectives. **PNAG** can adaptively generate architectures for any task dataset, requiring only the replacement of the surrogate model rather than retraining the generation model. To verify this adaptive capability of **PNAG**, we follow the experimental design of DiffusionNAG [1]. We use CIFAR10 as the task dataset and establish three scenarios: 1) Clean (no processing); 2) APGD (applying APGD perturbation [12]); and 3) Blur (applying Blur corruption perturbation). Subsequently, we collected 50 architectures from NB201, obtained their performance on these three datasets, and used this data to train three corresponding surrogate models (f_{Clean}, f_{APGD}, f_{Blur}). Finally, we used these surrogate models to guide **PNAG**'s architecture generation within the NB201 search space. With the guidance of these predictors, we generate a pool of 20 architectures for each and statistically analyzed the maximum

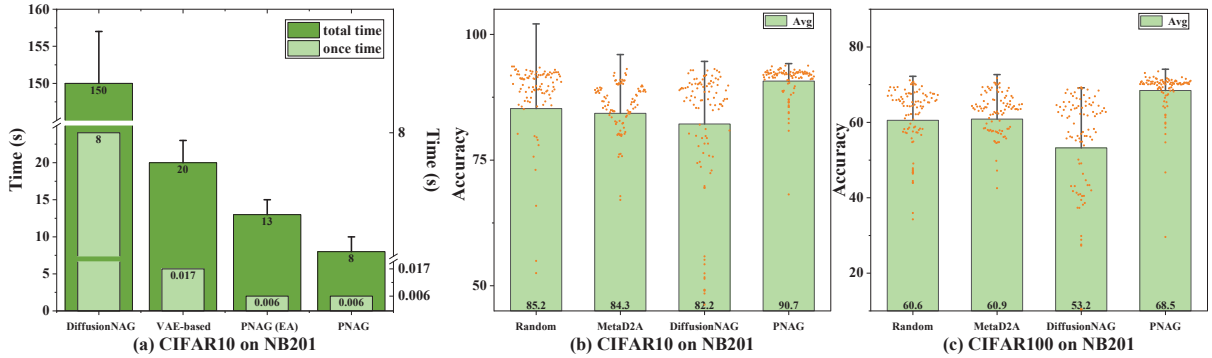

Figure 4. **Analysis of Generation Efficiency**. *Left:* Generation efficiency of different methods. The VAE-Based method is implemented by ourselves. *Middle and Right:* Average accuracy of 100 generated samples generated by different methods on cifar10 and cifar100. Dots indicate the accuracy value for each sample.

Table 6. **Analysis of Generation Validity and Quality.** We generate 1,000 architectures across three independent runs.

Method	NAS-Bench-201			MobileNetV3		
	Validity (%) ↑	Uniq. (%) ↑	Novelty (%) ↑	Validity (%) ↑	Uniq. (%) ↑	Novelty (%) ↑
GDSS	4.56 ± 1.44	-	-	0.00 ± 0.00	42.17 ± 1.80	-
POMONAG	99.97 ± 0.10	34.14 ± 17.33	37.41 ± 7.73	72.58 ± 5.84	91.79 ± 7.32	100.00 ± 0.00
DiffusionNAG	98.97 ± 0.29	98.70 ± 0.66	49.20 ± 1.96	99.09 ± 0.28	100.00 ± 0.00	100.00 ± 0.00
PNAG	$\mathbf{100.00 \pm 0.00}$	$\mathbf{99.70 \pm 0.30}$	$\mathbf{59.34 \pm 0.53}$	$\mathbf{100.00 \pm 0.00}$	$\mathbf{100.00 \pm 0.00}$	$\mathbf{100.00 \pm 0.00}$

accuracy (Max) and average accuracy (Mean) across 20 architectures. As shown in Tab. 5, architectures guided by a specific surrogate (e.g., f_{APGD}) performed best on the corresponding objective (APGD: 34.52%), significantly outperforming those guided by other models. This held true for the f_{Blur} (48.37%) and f_{Clean} (92.17%) guides. This demonstrates PNAG's adaptability in generating architectures for diverse performance requirements.

Analysis of Generation Efficiency. This section evaluates generation efficiency, focusing on the time required to complete a single-generation process and obtain an optimal architecture through various optimization strategies. We compare two **PNAG** variants (guided by Random and EA strategies) with two baseline methods. As shown in Figure 4(a), **PNAG** achieves remarkable efficiency, generating an architecture in just 0.006 seconds, representing a 1300× speedup over DiffusionNAG (8 seconds). On CIFAR-10, **PNAG** completes the full generation in 8 seconds, compared to 150s for DiffusionNAG and 20s for VAE, yielding 17× and 2× improvements, respectively. This gain primarily stems from the network-free design of MSQ, which eliminates costly inference steps.

Analysis of Generation Validity and Quality. This part evaluates the validity and quality of 1000 generated architectures across two search spaces. For NB201, the training set consists of 50% randomly selected architectures, while for MBv3, we sample 500,000 architectures. Table 6 summarizes our results, and observations are outlined below. According to Table 6, **PNAG** achieves top performance across both search spaces, maintaining 100% validity. Remarkably, **PNAG** attains 100% across all three metrics on MBV3. While its novelty on NB201 is slightly lower due to the smaller search space, it still substantially outperforms other methods. Furthermore, as illustrated in Figure 4(b) and (c), **PNAG** generates architectures with the highest average accuracy across 100 samples in both tasks, exhibiting a notably tighter distribution in the high-accuracy region compared to other methods. To sum up, **PNAG** has a significant advantage in generating high-quality architectures.

5. Conclusion

We present **PNAG**, a novel method for progressive neural architecture generation that incrementally constructs architectures using a coarse-to-fine approach, enhancing generation efficiency while ensuring architectural validity through step-wise refinements. **PNAG** utilizes multi-scale sub-architecture quantification (MSQ) to bypass costly network evaluations, accelerating the process, and incorporates step-wise consistency constraints (SCC) for generating high-quality architectures. It also adapts to transferable neural architecture generation with dataset-aware predictors. Experimental results show **PNAG**'s superior efficiency and architectural quality, offering new opportunities for efficient exploration of architectural space.

Acknowledgments

This work is supported in part by the National Major Scientific Instruments and Equipments Development Project of National Natural Science Foundation of China under Grant 62427820, the National Natural Science Foundation of China under Grants 62306014, the Postdoctoral Fellowship Program (Grade A) of CPSF under Grant BX20250376, the Sichuan Science and Technology Program under Grant 2025ZNSFSC1506, the Fundamental Research Funds for the Central Universities under Grant 1082204112K97, and the Sichuan University Interdisciplinary Innovation Fund.

References

[1] Sohyun An, Hayeon Lee, Jaehyeong Jo, Seanie Lee, and Sung Ju Hwang. Diffusionnag: Predictor-guided neural architecture generation with diffusion models. *arXiv preprint arXiv:2305.16943*, 2023. 1, 2, 4, 6, 7

[2] Rohan Asthana, Joschua Conrad, Youssef Dawoud, Maurits Ortmanns, and Vasileios Belagiannis. Multi-conditioned graph diffusion for neural architecture search. *arXiv preprint arXiv:2403.06020*, 2024. 2

[3] Bowen Baker, Otkrist Gupta, Nikhil Naik, and Ramesh Raskar. Designing neural network architectures using reinforcement learning. *arXiv preprint arXiv:1611.02167*, 2016. 6

[4] George EP Box, Gwilym M Jenkins, Gregory C Reinsel, and Greta M Ljung. *Time series analysis: forecasting and control*. John Wiley & Sons, 2015. 3

[5] Han Cai, Chuang Gan, Tianzhe Wang, Zhekai Zhang, and Song Han. Once-for-all: Train one network and specialize it for efficient deployment. In *International Conference on Learning Representations*, 2020. 2, 5

[6] Amber Cassimon, Siegfried Mercelis, and Kevin Mets. Scalable reinforcement learning-based neural architecture search. *Neural Computing and Applications*, 37(1):231–261, 2025. 2

[7] Minghao Chen, Houwen Peng, Jianlong Fu, and Haibin Ling. Autoformer: Searching transformers for visual recognition. In *Proceedings of the IEEE/CVF international conference on computer vision*, pages 12270–12280, 2021. 5

[8] Xiangning Chen, Ruochen Wang, Minhao Cheng, Xiaocheng Tang, and Cho-Jui Hsieh. Drnas: Dirichlet neural architecture search. *arXiv preprint arXiv:2006.10355*, 2020. 6

[9] Yaran Chen, Ruiyuan Gao, Fenggang Liu, and Dongbin Zhao. Modulenet: Knowledge-inherited neural architecture search. *IEEE Transactions on Cybernetics*, 52(11):11661–11671, 2021. 6

[10] Aakanksha Chowdhery, Sharan Narang, Jacob Devlin, Maarten Bosma, Gaurav Mishra, Adam Roberts, Paul Barham, Hyung Won Chung, Charles Sutton, Sebastian Gehrmann, et al. Palm: Scaling language modeling with pathways. *Journal of Machine Learning Research*, 24(240):1–113, 2023. 3

[11] Alexander I Cowen-Rivers, Wenlong Lyu, Rasul Tutunov, Zhi Wang, Antoine Grosnit, Ryan Rhys Griffiths, Alexandre Max Maraval, Hao Jianye, Jun Wang, Jan Peters, et al. Hebo: Pushing the limits of sample-efficient hyper-parameter optimisation. *Journal of Artificial Intelligence Research*, 74: 1269–1349, 2022. 6

[12] Francesco Croce and Matthias Hein. Reliable evaluation of adversarial robustness with an ensemble of diverse parameter-free attacks. In *International conference on machine learning*, pages 2206–2216. PMLR, 2020. 7

[13] Xuanyi Dong and Yi Yang. One-shot neural architecture search via self-evaluated template network. In *Proceedings of the IEEE/CVF International Conference on Computer Vision*, pages 3681–3690, 2019. 6

[14] Xuanyi Dong and Yi Yang. Searching for a robust neural architecture in four gpu hours. In *Proceedings of the IEEE/CVF conference on computer vision and pattern recognition*, pages 1761–1770, 2019. 6

[15] Xuanyi Dong and Yi Yang. Nas-bench-201: Extending the scope of reproducible neural architecture search. *arXiv preprint arXiv:2001.00326*, 2020. 2, 5

[16] Patrick Esser, Robin Rombach, and Bjorn Ommer. Taming transformers for high-resolution image synthesis. In *Proceedings of the IEEE/CVF conference on computer vision and pattern recognition*, pages 12873–12883, 2021. 6

[17] Stefan Falkner, Aaron Klein, and Frank Hutter. Bohb: Robust and efficient hyperparameter optimization at scale. In *International conference on machine learning*, pages 1437–1446. PMLR, 2018. 6

[18] Alexandros Graikos, Srikar Yellapragada, and Dimitris Samaras. Conditional generation from unconditional diffusion models using denoiser representations. *arXiv preprint arXiv:2306.01900*, 2023. 2

[19] Clive WJ Granger and Paul Newbold. Spurious regressions in econometrics. *Journal of econometrics*, 2(2):111–120, 1974. 3

[20] Yong Guo, Yaofo Chen, Yin Zheng, Qi Chen, Peilin Zhao, Junzhou Huang, Jian Chen, and Mingkui Tan. Pareto-aware neural architecture generation for diverse computational budgets. In *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition*, pages 2248–2258, 2023. 2

[21] Hongyi He, Longjun Liu, Haonan Zhang, and Nanning Zheng. Is-darts: stabilizing darts through precise measurement on candidate importance. In *Proceedings of the AAAI Conference on Artificial Intelligence*, pages 12367–12375, 2024. 6

[22] Zhenfeng He, Yao Shu, Zhongxiang Dai, and Bryan Kian Hsiang Low. Robustifying and boosting training-free neural architecture search. *arXiv preprint arXiv:2403.07591*, 2024. 6

[23] Kazuki Hemmi, Yuki Tanigaki, and Masaki Onishi. Navigator-d3: Neural architecture search using variational graph auto-encoder toward optimal architecture design for diverse datasets. In *International Conference on Artificial Neural Networks*, pages 292–307. Springer, 2024. 1, 2

[24] Tero Karras, Samuli Laine, and Timo Aila. A style-based generator architecture for generative adversarial networks. In *Proceedings of the IEEE/CVF conference on computer vision and pattern recognition*, pages 4401–4410, 2019. 6

[25] Doyup Lee, Chiheon Kim, Saehoon Kim, Minsu Cho, and Wook-Shin Han. Autoregressive image generation using residual quantization. In *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition*, pages 11523–11532, 2022. 4

[26] Hayeon Lee, Eunyoung Hyung, and Sung Ju Hwang. Rapid neural architecture search by learning to generate graphs from datasets. *arXiv preprint arXiv:2107.00860*, 2021. 6

[27] Liam Li and Ameet Talwalkar. Random search and reproducibility for neural architecture search. In *Uncertainty in artificial intelligence*, pages 367–377. PMLR, 2020. 6

[28] Yangyang Li, Guanlong Liu, Ronghua Shang, and Licheng Jiao. Meta knowledge assisted evolutionary neural architecture search. *IEEE Transactions on Circuits and Systems for Video Technology*, 2025. 2

[29] Hanxiao Liu, Karen Simonyan, and Yiming Yang. Darts: Differentiable architecture search. *arXiv preprint arXiv:1806.09055*, 2018. 2, 5, 6

[30] Yuqiao Liu, Yehui Tang, and Yanan Sun. Homogeneous architecture augmentation for neural predictor. In *Proceedings of the IEEE/CVF International Conference on Computer Vision*, pages 12249–12258, 2021. 6

[31] I Loshchilov. Decoupled weight decay regularization. *arXiv preprint arXiv:1711.05101*, 2017. 6

[32] Jovita Lukasik, David Friede, Arber Zela, Frank Hutter, and Margret Keuper. Smooth variational graph embeddings for efficient neural architecture search. In *2021 International Joint Conference on Neural Networks (IJCNN)*, pages 1–8. IEEE, 2021. 2

[33] Subhransu Maji, Esa Rahtu, Juho Kannala, Matthew Blaschko, and Andrea Vedaldi. Fine-grained visual classification of aircraft. *arXiv preprint arXiv:1306.5151*, 2013. 6

[34] Krzysztof Maziarz, Mingxing Tan, Andrey Khorlin, Marin Georgiev, and Andrea Gesmundo. Evolutionary-neural hybrid agents for architecture search. *arXiv preprint arXiv:1811.09828*, 2018. 2

[35] Keith G Mills, Fred X Han, Mohammad Salameh, Shengyao Lu, Chunhua Zhou, Jiao He, Fengyu Sun, and Di Niu. Building optimal neural architectures using interpretable knowledge. In *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition*, pages 5726–5735, 2024. 2

[36] Rohit Mohan, Thomas Elsken, Arber Zela, Jan Hendrik Metzen, Benedikt Staffler, Thomas Brox, Abhinav Valada, and Frank Hutter. Neural architecture search for dense prediction tasks in computer vision. *International Journal of Computer Vision*, 131(7):1784–1807, 2023. 2

[37] Omkar M Parkhi, Andrea Vedaldi, Andrew Zisserman, and CV Jawahar. Cats and dogs. In *2012 IEEE conference on computer vision and pattern recognition*, pages 3498–3505. IEEE, 2012. 6

[38] Patrick C Parks. Am lyapunov's stability theory—100 years on. *IMA journal of Mathematical Control and Information*, 9 (4):275–303, 1992. 5

[39] Yameng Peng, Andy Song, Vic Ciesielski, Haytham M Fayek, and Xiaojun Chang. Pre-nas: Evolutionary neural architecture search with predictor. *IEEE Transactions on Evolutionary Computation*, 27(1):26–36, 2022. 1

[40] Binxin Ru, Xingchen Wan, Xiaowen Dong, and Michael Osborne. Interpretable neural architecture search via bayesian optimisation with weisfeiler-lehman kernels. In *International Conference on Learning Representations*, 2021. 6

[41] Gresa Shala, Thomas Elsken, Frank Hutter, and Josif Grabocka. Transfer nas with meta-learned bayesian surrogates. In *The Eleventh International Conference on Learning Representations*, 2023. 6

[42] Yao Shu, Shaofeng Cai, Zhongxiang Dai, Beng Chin Ooi, and Bryan Kian Hsiang Low. Nasi: Label-and data-agnostic neural architecture search at initialization. *arXiv preprint arXiv:2109.00817*, 2021. 6

[43] Keyu Tian, Yi Jiang, Zehuan Yuan, Bingyue Peng, and Liwei Wang. Visual autoregressive modeling: Scalable image generation via next-scale prediction. *arXiv preprint arXiv:2404.02905*, 2024. 3, 4, 6

[44] Aaron Van Den Oord, Oriol Vinyals, et al. Neural discrete representation learning. *Advances in neural information processing systems*, 30, 2017. 2, 4

[45] Wei Wen, Kuang-Hung Liu, Igor Fedorov, Xin Zhang, Hang Yin, Weiwei Chu, Kaveh Hassani, Mengying Sun, Jiang Liu, Xu Wang, et al. Rankitect: Ranking architecture search battling world-class engineers at meta scale. In *Companion Proceedings of the ACM Web Conference 2024*, pages 73–82, 2024. 2

[46] Colin White, Willie Neiswanger, and Yash Savani. Bananas: Bayesian optimization with neural architectures for neural architecture search. In *Proceedings of the AAAI conference on artificial intelligence*, pages 10293–10301, 2021. 6

[47] Colin White, Mahmoud Safari, Rhea Sukthanker, Binxin Ru, Thomas Elsken, Arber Zela, Debadeepta Dey, and Frank Hutter. Neural architecture search: Insights from 1000 papers. *arXiv preprint arXiv:2301.08727*, 2023. 1

[48] Xiangning Xie, Yanan Sun, Yuqiao Liu, Mengjie Zhang, and Kay Chen Tan. Architecture augmentation for performance predictor via graph isomorphism. *IEEE Transactions on Cybernetics*, 54(3):1828–1840, 2023. 6

[49] Peng Xu, Lin Zhang, Xuanzhou Liu, Jiaqi Sun, Yue Zhao, Haiqin Yang, and Bei Yu. Do not train it: A linear neural architecture search of graph neural networks. In *International Conference on Machine Learning*, pages 38826–38847. PMLR, 2023. 1

[50] Yuhui Xu, Lingxi Xie, Xiaopeng Zhang, Xin Chen, Guo-Jun Qi, Qi Tian, and Hongkai Xiong. Pc-darts: Partial channel connections for memory-efficient architecture search. *arXiv preprint arXiv:1907.05737*, 2019. 6

[51] Shen Yan, Yu Zheng, Wei Ao, Xiao Zeng, and Mi Zhang. Does unsupervised architecture representation learning help neural architecture search? *Advances in neural information processing systems*, 33:12486–12498, 2020. 2

[52] Le Yang, Ziwei Zheng, Yizeng Han, Shiji Song, Gao Huang, and Fan Li. Ostr-darts: Differentiable neural architecture search based on operation strength. *IEEE Transactions on Cybernetics*, 2024. 6

[53] Caiyang Yu, Xianggen Liu, Yifan Wang, Yun Liu, Wentao Feng, Xiong Deng, Chenwei Tang, and Jiancheng Lv. Gpt-nas: Neural architecture search meets generative pre-trained transformer model. *Big Data Mining and Analytics*, 2024. 2, 6

[54] B Zoph. Neural architecture search with reinforcement learning. *arXiv preprint arXiv:1611.01578*, 2016. 2

[55] Barret Zoph, Vijay Vasudevan, Jonathon Shlens, and Quoc V Le. Learning transferable architectures for scalable image recognition. In *Proceedings of the IEEE conference on computer vision and pattern recognition*, pages 8697–8710, 2018. 6