Appendix for "Evolution Meets Diffusion: Efficient Neural Architecture Generation"

A Details of EDNAG

A.1 Generation Process

A.1.1 Deoising Process of DDIM

Traditional diffusion models [9] initiate with noise sampled from the known prior distribution (e.g., Gaussian distribution) and then iteratively remove noise predicted by a trained network. Denoising Diffusion Implicit Model (DDIM) [9] is an efficient diffusion model that significantly reduces the number of steps in denoising process while achieving comparable generation performance. DDIM performs a denoising iteration by the following formula:

$$x_{t-1} = \sqrt{\overline{\alpha_{t-1}}}\hat{x}_0 + \sqrt{1 - \overline{\alpha_{t-1}} - \sigma_t^2} \cdot \hat{\epsilon}_\theta\left(x_t, t\right) + \sigma_t \epsilon_t, \tag{1}$$

where $\hat{x}_0$ is the predicted optimal samples at time step 0, which can be calculated by the noise $\hat{\epsilon}_\theta\left(x_t, t\right)$ estimated by a trained network, detailed as follows:

$$\hat{x}_0 = \frac{x_t - \sqrt{1 - \overline{\alpha_t}} \cdot \hat{\epsilon}_\theta\left(x_t, t\right)}{\sqrt{\overline{\alpha_t}}}. \tag{2}$$

Combining Equations (1) and (2), the formula for the denoising process in DDIM is as follows:

$$x_{t-1} = \sqrt{\overline{\alpha_{t-1}}}\frac{x_t - \sqrt{1 - \overline{\alpha_t}} \cdot \hat{\epsilon}_\theta\left(x_t, t\right)}{\sqrt{\overline{\alpha_t}}} + \sqrt{1 - \overline{\alpha_{t-1}} - \sigma_t^2} \cdot \hat{\epsilon}_\theta\left(x_t, t\right) + \sigma_t \epsilon_t. \tag{3}$$

After obtaining $\hat{\epsilon}_\theta\left(x_t, t\right)$ predicted by a trained network, DDIM utilize Equation (3) to denoise the samples from x_t to x_{t-1}. Through multiple iterations, DDIM is capable of generating the optimal sample x_0.

A.1.2 Deoising Process of EDNAG

We propose a fitness-guided denoising (FD) strategy to simulate the denoising process in diffusion models. Unlike DDIM, which performs denoising process through $\hat{\epsilon}_\theta\left(x_t, t\right)$ predicted by a network, FD strategy directly obtains $\hat{x}_0$ by the fitness of x_t. Subsequently, based on the estimated $\hat{x}_0$, we can inversely derive $\hat{\epsilon}_\theta\left(x_t, t\right)$ as follows:

$$\hat{\epsilon}_\theta\left(x_t, t\right) = \frac{x_t - \sqrt{\overline{\alpha_t}}\hat{x}_0}{\sqrt{1 - \overline{\alpha_t}}}. \tag{4}$$

With $\hat{x}_0$ and $\hat{\epsilon}_\theta\left(x_t, t\right)$, EDNAG can generate optimal samples by iteratively denoising. Therefore, combining Equations (1) and (4), the formula for the denoising process of FD strategy is as follows:

$$x_{t-1} = \sqrt{\overline{\alpha_{t-1}}}\hat{x}_0 + \sqrt{1 - \overline{\alpha_{t-1}} - \sigma_t^2} \cdot \frac{x_t - \sqrt{\overline{\alpha_t}}\hat{x}_0}{\sqrt{1 - \overline{\alpha_t}}} + \sigma_t \epsilon_t. \tag{5}$$

In this way, we only needs to derive $\hat{x}_0$ to perform the denoising process, avoiding the usage of trained networks to estimate $\hat{\epsilon}_\theta\left(x_t, t\right)$, thereby enabling generation without network involvement and the training process.

Therefore, the core challenge of FD strategy lies in determining how to estimate the final samples $\hat{x}_0$ at time step 0. In a nutshell, we achieve this by mapping the previous architecture samples x_t to the distribution of the optimal architecture samples $\hat{x}_0$ based on fitness.

First, we consider that the denoising process of diffusion models is essentially equivalent to the iterative process of evolutionary algorithms [27]. Both processes transform an initial distribution of random samples into an ultimate distribution of optimal samples progressively. Therefore, the denoising of samples is equivalent to the evolution of the population. Specifically, high-fitness architectures are more likely to steer the evolutionary direction during the multiple denoising processes. Consequently, high-fitness architectures are more likely to be preserved throughout the denoising processes, increasing their probability of being retained in the final generated samples, as shown in Figure 1.

We model this relationship between fitness and optimal architectures as a mapping function $g(x)$, which maps the fitness values $f(x_t)$ of architecture samples x_t to the distribution of optimal architecture samples $\hat{x}_0$, expressed as:

$$p(x_0 = x) = g[f(x)]. \tag{6}$$

Through our mapping function, higher-fitness architectures in the population are each assigned a higher probability density of $\hat{x}_0$. According to Bayesian formula, we can estimate the optimal architecture samples $\hat{x}_0$, as shown below:

$$p(x_0 = x|x_t) = \frac{p(x_0 = x) \cdot p(x_t|x_0 = x)}{p(x_t)}. \tag{7}$$

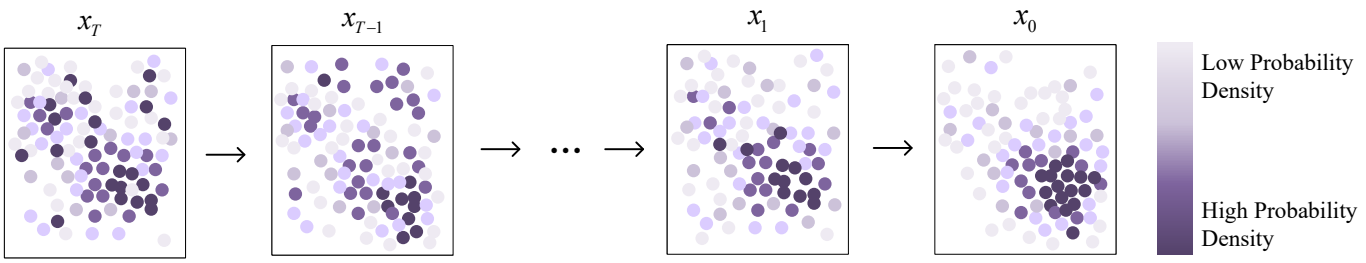

Figure 1. High fitness correlates with a high final probability density of x_0. During iterations, the initial architectures are progressively refined towards high-probability regions. Consequently, high-fitness individuals are more likely to be retained in x_0, resulting in a higher probability density at x_0.

With Equation (6), we further detail Equation (7) as follows:

$$p(x_0 = x|x_t) = \frac{g[f(x)] \cdot p(x_t|x_0 = x)}{p(x_t)}. \tag{8}$$

Consequently, we only need to determine $p(x_t|x_0 = x)$. Considering the forward diffusion process of diffusion models, as shown below, we can derive x_t by x_0.

$$x_t = \sqrt{\overline{\alpha_t}}x_0 + \sqrt{1 - \overline{\alpha_t}}\epsilon_t, \;\; \epsilon_t \sim \mathcal{N}(0, I). \tag{9}$$

We convert Equation (9) into the form of a conditional probability, as shown below:

$$p(x_t|x_0 = x) \sim \mathcal{N}(x_t; \sqrt{\overline{\alpha_t}}x, 1 - \overline{\alpha}_t). \tag{10}$$

Substituting Equation (10) into Equation (8), and denoting the samples x_t as $x_t = [\boldsymbol{x}_t^1, \boldsymbol{x}_t^2, \ldots, \boldsymbol{x}_t^N]$, where N represents the number of neural architectures or the size of architecture populations, the estimation formula for $\hat{x}_0$ is derived as follows:

$$\begin{aligned} \hat{x}_0 &= \frac{1}{p(x_t)} \sum_{\boldsymbol{x}_t^i \in x_t} g[f(\boldsymbol{x}_t^i)]\mathcal{N}(x_t; \sqrt{\overline{\alpha_t}}\boldsymbol{x}_t^i, 1 - \overline{\alpha_t})\boldsymbol{x}_t^i, \\ p(x_t) &= \sum_{\boldsymbol{x}_t^i \in x_t} g[f(\boldsymbol{x}_t^i)]\mathcal{N}(x_t; \sqrt{\overline{\alpha_t}}\boldsymbol{x}_t^i, 1 - \overline{\alpha_t}), \end{aligned} \tag{11}$$

where $p(x_t)$ serves as a regularization term. In Equation (11), $g[f(\boldsymbol{x}_t^i)]$ assigns greater weights to architectures with higher fitness, which can be viewed as a weighted summation of the architectures in the sample. Additionally, $\mathcal{N}(x_t; \sqrt{\overline{\alpha_t}}\boldsymbol{x}_t^i, 1 - \overline{\alpha_t})$ enhances local sensitivity by increasing the influence of neighboring architectures on $\boldsymbol{x}_t^i$ while reducing the impact of distant architectures.

Finally, combining the denoising formula (Equation (5)) with the prediction formula for $\hat{x}_0$ (Equation (11)), we derive the final denoising iteration formula of FD strategy as follows:

$$\begin{cases} x_{t-1} = \sqrt{\overline{\alpha_{t-1}}}\hat{x}_0 + \sqrt{1 - \overline{\alpha_{t-1}} - \sigma_t^2} \cdot \dfrac{x_t - \sqrt{\overline{\alpha_t}}\hat{x}_0}{\sqrt{1 - \overline{\alpha_t}}} + \sigma_t\epsilon_t, \\ \hat{x}_0 = \dfrac{1}{p(x_t)} \displaystyle\sum_{\boldsymbol{x}_t^i \in x_t} g[f(\boldsymbol{x}_t^i)]\mathcal{N}(x_t; \sqrt{\overline{\alpha_t}}\boldsymbol{x}_t^i, 1 - \overline{\alpha_t})\boldsymbol{x}_t^i, \\ p(x_t) = \displaystyle\sum_{\boldsymbol{x}_t^i \in x_t} g[f(\boldsymbol{x}_t^i)]\mathcal{N}(x_t; \sqrt{\overline{\alpha_t}}\boldsymbol{x}_t^i, 1 - \overline{\alpha_t}). \end{cases} \tag{12}$$

The FD strategy employs the fitness-guided denoising process in Equation (12) to denoise from x_t to x_{t-1}. By iteratively applying denoising process of FD, EDNAG can progressively denoise x_T to x_0, ultimately achieving the generation of optimal neural architectures.

A.1.3 Optimization Strategies for Selection

Following Equation (12), the previous sample x_t is denoised to produce x_{t-1}. Subsequently, optimization strategies (denoted as S) are employed to select the final sample x_{t-1} at time step t from x_t and denoised x_{t-1}, expressed as follows:

$$x_{t-1} = S(x_t, x_{t-1}). \tag{13}$$

Specifically, we initially employ the elitism strategy, selecting a few high-fitness architectures $\boldsymbol{x}_{t-1}^i$ from x_t and x_{t-1} to ensure the retention of optimal architectures during the generation process, thus enhancing stability and preventing degradation through iterations, detailed as follows:

$$\boldsymbol{x}_{t-1}^i = \arg \max_{\boldsymbol{x} \in \{x_t, x_{t-1}\}} f(\boldsymbol{x}). \tag{14}$$

Algorithm 1 Detailed Neural Architecture Generation Process

Input: Population size N, number of nodes D_1, types of operations D_2, denoising steps T, fitness evaluator f, mapping function g.
Output: Generated optimal neural architectures.

$x_T \sim \mathcal{N}(0, I^{N\times D_1\times D_2})$. ▷ Initialize
for t in $[T, T-1, \ldots, 1]$ **do**
  $p(x_t) \leftarrow \sum_{\boldsymbol{x}_t^i \in x_t} g[f(\boldsymbol{x}_t^i)]\mathcal{N}(x_t; \sqrt{\overline{\alpha_t}}\boldsymbol{x}_t^i, 1-\overline{\alpha_t})$
  $\hat{x}_0 \leftarrow \frac{1}{p(x_t)} \sum_{\boldsymbol{x}_t^i \in x_t} g[f(\boldsymbol{x}_t^i)]\mathcal{N}(x_t; \sqrt{\overline{\alpha_t}}\boldsymbol{x}_t^i, 1-\overline{\alpha_t})\boldsymbol{x}_t^i$ ▷ Estimate $\hat{x}_0$
  $x_{t-1} \leftarrow \sqrt{\overline{\alpha_{t-1}}}\hat{x}_0 + \sqrt{1-\overline{\alpha_{t-1}}-\sigma_t^2}\cdot\frac{x_t-\sqrt{\overline{\alpha_t}}\hat{x}_0}{\sqrt{1-\overline{\alpha_t}}} + \sigma_t\epsilon_t$ ▷ Denoise
  $x_{t-1} \leftarrow S(x_t, x_{t-1})$ ▷ Optimization Strategies for Selection
end for
return x_0

Furthermore, to ensure diversity of generated architecture samples, we compute the Euclidean distance between architectures, and then we select a few the most diverse architectures $\boldsymbol{x}_{t-1}^i$ from x_t and x_{t-1}, as shown below:

$$\boldsymbol{x}_{t-1}^i = \arg\max_{\boldsymbol{x}\in\{x_t, x_{t-1}\}} \sum_{j=1}^{2N} \|\boldsymbol{x} - \boldsymbol{x}^j\|. \tag{15}$$

Finally, we employ a roulette wheel selection strategy to select the remaining architectures. We map the fitness of architectures to their probability of being selected, which is calculated as follows:

$$P(\boldsymbol{x}^i) = \frac{f(\boldsymbol{x}^i)}{\sum_{\boldsymbol{x}^j\in\{x_t, x_{t-1}\}} f(\boldsymbol{x}^j)}. \tag{16}$$

Through Equation (16), architectures with higher fitness have a higher probability of being selected, therefore balancing randomness and convergence in the generation process.

Therefore, the process of generating architecture samples in EDNAG can be represented in Algorithm 1.

A.2 Neural Predictor

EDNAG introduces a transferable neural predictor, which first encodes neural architectures and task datasets separately, then predicts the accuracy of architectures on the given task dataset, achieving cross-dataset performance evaluation of neural architectures. Therefore, our neural predictor consists of three parts: a dataset encoder, an architecture encoder, and a linear predictor.

A.2.1 Dataset Encoder

The generalization ability of neural predictors across various datasets is closely related to dataset encoders. The same with [13], the dataset encoder comprises two stacked modules that are permutation-invariant, where the encoder's output is independent of the order of input images. The intra-class encoder, captures common features among images of the same class, while the inter-class encoder distinguishes between different classes and maps images to a latent vector.

We employ Set Attention Blocks (SAB) as intra-class encoders and Pooling by Multi-head Attention (PMA) as inter-class encoders [14]. Let LN denotes layer normalization, MLP represents multi-layer perceptrons, MH denotes multi-head attention, S represents learnable seed vectors, and $\mathcal{D}$ represents the input dataset. SAB can be represented as follows:

$$\begin{aligned} &\text{SAB}(\mathcal{D}) = \text{LN}(H + \text{MLP}(H)),\\ &H = \text{LN}(\mathcal{D} + \text{MH}(\mathcal{D}, \mathcal{D}, \mathcal{D})). \end{aligned} \tag{17}$$

And PMA can be represented as follows:

$$\begin{aligned} &\text{PMA}(\mathcal{D}) = \text{LN}(H + \text{MLP}(H)),\\ &H = \text{LN}(\mathcal{D} + \text{MH}(S, \text{MLP}(\mathcal{D}), \text{MLP}(\mathcal{D}))). \end{aligned} \tag{18}$$

Through SAB and PMA, our dataset encoder $p(z_{\mathcal{D}}|\mathcal{D})$ encodes the target dataset into a latent vector $z_{\mathcal{D}}$.

It is particularly noteworthy that in DiffusionNAG, the neural predictor randomly samples n images from each class in the task dataset and then encodes them with the dataset encoder. We have demonstrated in our experiments that this approach overlooks the uniformity of the target dataset distribution, leading to biased sampling and consequently inaccurate predictions. Additionally, it introduces instability in predictions, resulting in significant variations in predictions for the same architecture on the same task dataset, further causing instability in the architecture generation process.

To address this issue in DiffusionNAG, EDNAG employs a coreset selection methodology for sampling from the task dataset. A widely adopted method for coreset selection is K-Means clustering, which is also utilized by EDNAG. We cluster the images in each class of the task dataset into n clusters, where n denotes the number of samples per class. Subsequently, we select the images corresponding to the centroids of each cluster as the coreset for dataset encoding.

Table 1. Comparison of search spaces. For each search space, we report the total number of architectures, architecture configurations, number of vision tasks, number of datasets per task domain in our experiements.

Metric	NAS-Bench-101	NAS-BENCH-201	TransNASBench-101-Micro	TransNASBench-101-Micro	MobieNetV3
Size	4.23×10^5	15625	4096	3256	2.17×10^{19}
Number of Nodes	7	8	6	6	22
Number of Operations	3	7	4	5	12
Task Domains	1	1	7	7	1
Datasets (for each domain)	1	4	1	1	4

Let $\mathcal{T}$ represent the dataset and $\mathcal{D}$ denote the coreset. The objective of K-Means clustering is to identify the centroid images $x_c \in \mathcal{D}$, minimizing the distance from each data point x to its respective cluster center x_c, as illustrated below:

$$\min_{x_c \in \mathcal{D}} \sum_{x \in \mathcal{T}} \|x - x_c\|^2. \tag{19}$$

A.2.2 Architecture Encoder

We employ a graph convolution network specifically designed for directed graphs, DiGCN [23], as the neural architecture encoder. A neural architecture x is represented as $x = (\mathcal{V}, \mathcal{E})$, where $\mathcal{E}$ is the adjacency matrix and $\mathcal{V}$ is the node operation matrix. We averaged the outputs of two GCN layers in a module of architecture encoder. One layer uses the adjacency matrix $\hat{\mathcal{E}}$ to capture forward information, while the other layer uses the transpose of the adjacency matrix $\hat{\mathcal{E}}^\top$ to capture backward information. In the l-th modules of architecture encoder, the bi-directional encoding strategy can be described as follows:

$$H^l = \frac{1}{2}\text{ReLU}\left(\hat{\mathcal{E}} H^{l-1} W_1^{l-1}\right) + \frac{1}{2}\text{ReLU}\left(\hat{\mathcal{E}}^\top H^{l-1} W_2^{l-1}\right), \tag{20}$$

where W_1^{i-1} and W_2^{i-1} denotes different weights in two GCN layers, and $H^0 = \mathcal{V}$. By stacking 4 DiGCN modules in Equation (20), our architecture encoder effectively integrates bidirectional information flow, efficiently capturing the structure information of the neural architectures. Subsequently, we apply an average pooling layer and several fully connected layers to H^l, aggregating the information into the final latent vector $z_{\mathcal{G}}$.

A.2.3 Linear Predictor

After concatenating the outputs of both the graph encoder and the set encoder, we used them as the input to a linear predictor, which consists of only two linear layers with ReLU activations. We trained the entire neural predictor f_ω by minimizing the MSE loss between the predicted accuracy and the true accuracy a, as shown below:

$$\min_{\omega} \mathcal{L}(f_\omega(\mathcal{D}, \mathcal{G}) - a)^2. \tag{21}$$

Notably, EDNAG achieves architecture generation without training, referring to the generation method that does not any require training networks. However, similar to all other methods, our neural predictor still requires training. After training on the ImageNet-1K [20] dataset, our neural predictor can be applied to various unseen task datasets, exhibiting its outstanding efficacy and adaptability.

B Experimental Setup

Neural architectures $x = (\mathcal{V}, \mathcal{E}) \in \mathbb{R}^{N \times O} \times \mathbb{R}^{N \times N}$ are represented as a directed acyclic graph (DAG), including the node operation matrix $\mathcal{V} \in \mathbb{R}^{N \times O}$ and the adjacency matrix $\mathcal{E} \in \mathbb{R}^{N \times N}$, where N denotes the number of nodes and O denotes the number of operations. We compared the settings within search spaces as described in Table 1.

B.1 Search Space

B.1.1 NAS-BENCH-101

NAS-Bench-101 [26] is the first large-scale benchmark dataset for NAS, which trains and evaluates a large number of different convolutional neural network (CNN) architectures on CIFAR-10 [11]. In NAS-Bench-101, each neural architecture consists of three repeated stages, with each stage containing three identical cells, followed by a down-sampling layer. Each cell contains a maximum of 9 edges and 7 nodes, with three possible operations: **3×3 convolution**, **1×1 convolution**, and **3×3 max-pooling**. Therefore, NAS-Bench-101 encompasses 5.10×10^8 architectures. However, many of these architectures are either invalid or isomorphic graphs, resulting in only 4.23×10^5 unique architectures. In the implementation of EDNAG, nodes represent layer operations, while edges signify layer connections. Consequently, each neural architecture is characterized by an adjacency matrix and a node operation matrix.

B.1.2 NAS-BENCH-201

As a cell-based search space, the NAS-BENCH-201 search space comprises 15,625 neural architectures, which is the most commonly used benchmark datasets in NAS. Each neural architecture comprises a stem cell, a stage, residual blocks, a stage, residual blocks, a stage, and a final classification layer. The stem cell, serving as the initial building block, consists of a 3-by-3 convolutional layer and a batch normalization layer. The final classification layer, serving as the concluding building block, includes an average pooling layer and a softmax-activated fully connected layer. Each stage encompasses five cells, and the total 15 cells across three stages share identical layer configurations, although the output channels differ between stages, being 16, 32, and 64, respectively. For each cell described in DAG, nodes represent operations (layers), and edges denote connections between layers. Each cell comprises eight nodes and each node consists seven distinct operations: **none**, **skip connection**, **1-by-1 convolution**, **3-by-3 convolution**, **3-by-3 average pooling**, **input**, and **output**.

Actually, the none operation governs layer connections. When a node type is none, it signifies that the preceding and subsequent layers are not connected. Consequently, the adjacency matrix is unnecessary due to the fact that the node operation matrix alone suffices to control the connections between layers. In EDNAG, each node is connected to all subsequent nodes, thereby fixing the adjacency matrix $\mathcal{E}$ as that of a fully connected DAG, as illustrated in Figure 2. Thus, EDNAG only needs to generate the operation matrix $\mathcal{V}$, shape of 8×7, for one cell in NAS-BENCH-201.

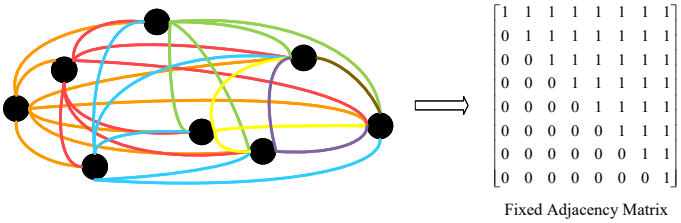

Figure 2. Illustrations of NAS-BENCH-201 cell. We connect each node to all subsequent nodes, thereby fixing the adjacency matrix and using only the node operation matrix to control the structure of the cell.

B.1.3 TransNASBench-101

TransNAS-Bench-101 includes two distinct search spaces. The first, known as the macro skeleton search space, leverages residual blocks and introduces variations in network depth and block operations, including channel adjustments and resolution reduction. The second is the widely explored cell-based search space, where each cell is represented as a directed acyclic graph. The macro-level search space comprises networks constructed with 4 to 6 modules, where each module performs 1 to 4 downsampling operations. Each operation reduces the spatial dimensions of the feature map by a factor of 2. In contrast, the micro-level search space consists of architectures with 6 network layers, each applying one of the predefined operations: zeroize, skip connection, 1×1 convolution, or 3×3 convolution. Consequently, the macro-level search space contains 3,256 architectures, while the cell-level search space encompasses 4,096 architectures.

Networks in TransNAS-Bench-101 are trained and evaluated across seven distinct vision tasks: **object classification**, **scene classification**, **semantic segmentation**, **autoencoding**, **room layout estimation**, **surface normal prediction**, and **jigsaw puzzle solving**. Scene classification predicts room types through a 47-way classification task, using labels from a ResNet-152 model pre-trained on the MIT Places dataset. Similarly, object classification recognizes objects in a 75-way classification problem, with labels derived from a ResNet-152 model pre-trained on ImageNet. Semantic segmentation performs pixel-level predictions to classify image components into 17 semantic classes, leveraging labels from a model trained on MSCOCO. Autoencoding encodes images into low-dimensional representations and reconstructs them, following the Pix2Pix framework. Room layout estimation identifies and aligns a 3D bounding box to define the room structure, incorporating semantic and geometric understanding. Surface normal prediction estimates pixel-wise surface statistics, employing a similar structure and training process as autoencoding. The self-supervised task jigsaw involves rearranging shuffled image patches into their original sequence based on one of 1,000 permutations.

B.1.4 MobileNetV3

As a layer-wise search space, MobileNetV3 search space comprises more than 2.17×10^{19} neural architectures. In EDNAG, the neural architecture of MobileNetV3 is represented as a directed acyclic graph with 22 layers and 12 operations. Each neural architecture comprises five stages and each stage consists of 2 to 4 layers. Considering the input and output layers, the maximum number of layers in MobileNetV3 is $4 \times 5 + 2 = 22$ layers. For each layer, the kernel size of the convolution can be selected from {**3**, **5**, **7**}, and the expansion ratio of channels can be chosen from {**3**, **4**, **6**}, resulting in a total of 9 operations. Considering **input**, **output**, and **none** operations, each node has $3 \times 3 + 3 = 12$ operation types. Distinct 12 operations can be represented as follows: **input**, **output**, **none**, **3-3**, **3-4**, **3-6**, **5-3**, **5-4**, **5-6**, **7-3**, **7-4**, **7-6**.

Similar to NAS-BENCH-201, the neural architectures in the MobileNetV3 search space fix the adjacency matrix $\mathcal{E}$ as that of a fully connected DAG. Consequently, EDNAG only needs to generate the operation matrices $\mathcal{V}$ of architectures, which have the shape of 22×12.

Notably, during the generation process, EDNAG generates neural architectures with 12 distinct operations, while in the graph encoder of the neural predictor, the number of node operation types is set to 27. Following MetaD2A, we combine the number of layers in each stage with the layer operations, resulting in a total of 27 operations. Distinct 27 operations are described in the format of "number of layers - kernel size

Table 2. Hyperparameter settings of EDNAG. We present detailed information of the experimental hyperparameter settings for EDNAG within the NAS-BENCH-101 (NB101), NAS-BENCH-201 (NB201), TransNASBench-101 (Trans101) and MobileNetV3 (MBV3) search spaces.

	Hyperparameter	NB101	NB201	Trans101-Micro	Trans101-Macro	MBV3
Denoising Process	Max Noise Scale of Diffusion α_0	$1-10^{-4}$	$1-10^{-4}$	$1-10^{-4}$	$1-10^{-4}$	$1-10^{-4}$
	Min Noise Scale of Diffusion α_T	10^{-4}	10^{-4}	10^{-4}	10^{-4}	10^{-4}
	Noise Scale of Denoising σ	0.8	0.8	0.8	0.8	0.8
	Number of Steps	100	100	100	100	100
	Population Size (Batch Size)	30	30	30	30	100
	Shape of Gene	(7, 7)	(8, 7)	(6, 4)	(6, 5)	(22, 12)
	Mapping Function	Energy	Energy	Energy	Energy	Energy
	Temperature of Mapping Function	1.0	1.0	1.0	1.0	1.0
Selection Strategies	Ratio of Elitism Strategy	0.1	0.1	0.1	0.1	0.1
	Ratio of Diversity Strategy	0.2	0.2	0.2	0.2	0.2
Neural Predictor	Distance in K-Means Clustering	-	Euclidean	-	-	Euclidean
	Hidden Dimension of Graph Encoder	-	144	-	-	144
	Hidden Dimension of Dataset Encoder	-	56	-	-	56
	Number of Image Samples in each class	-	20	-	-	20
	Number of Different Node Types	-	7	-	-	27

- expansion rate", represented as follows: **2-3-3**, **2-3-4**, **2-3-6**, **2-5-3**, **2-5-4**, **2-5-6**, **2-7-3**, **2-7-4**, **2-7-6**, **3-3-3**, **3-3-4**, **3-3-6**, **3-5-3**, **3-5-4**, **3-5-6**, **3-7-3**, **3-7-4**, **3-7-6**, **4-3-3**, **4-3-4**, **4-3-6**, **4-5-3**, **4-5-4**, **4-5-6**, **4-7-3**, **4-7-4**, **4-7-6**.

B.2 Datasets

In most search spaces, our experiments are conducted on the following task datasets: CIFAR10 [12], CIFAR100 [11], Aircraft [16], and Oxford-IIIT Pets [17]. Within the NAS-BENCH-201 search space, all images are resized to 32×32 pixels, while in the MobileNetV3 search space, images are resized to 224×224 pixels.

(1) CIFAR-10 is a classic benchmark dataset in NAS, consisting of 32×32 color images across ten distinct classes: airplane, automobile, bird, cat, deer, dog, frog, horse, ship, and truck. It includes 50,000 images for training and 10,000 images for testing.

(2) CIFAR-100 consists of 32×32 color images, but includes 100 distinct classes. It comprises 50,000 training images and 10,000 testing images.

(3) FGVC-Aircraft comprises 10,200 images of aircrafts, with 100 images for each of 102 different aircraft model variants. The dataset is organized into a three-tiered hierarchical structure, containing 102, 70, and 41 categories respectively. In the NAS-BENCH-201 search space, we employed the 41-category Aircraft dataset, classified by manufacturer. And in the MobileNetV3 search space, we used the 102-category Aircraft dataset, classified by variant.

(4) Oxford-IIIT Pets includes 37 distinct pet categories, with roughly 200 images per class. This dataset consists of 7,349 images in total, with 85% allocated for training and 15% for testing.

B.3 Hyperparameters of NAG Process

For the neural architecture generation process, the hyperparameter settings are detailed in Table 2.

B.3.1 Hyperparameters of Denoising Process

During the diffusion process, the scale of added noise at each step gradually decreases from α_0 to α_T, while the noise scale in the denoising process remains fixed at σ. Additionally, we provide details on the number of iteration steps for the denoising process, the types of mapping functions utilized, and the temperature, which governs the numerical sensitivity of these functions. Furthermore, the shape of the sample genotype is defined as (number of nodes, number of operations), and the operation matrix $\mathcal{V}$ for the generated samples is a three-dimensional matrix with the shape (batch size, number of nodes $\times$ number of operations). These details are all provided in the table.

B.3.2 Hyperparameters of Optimization Strategies for Selection

EDNAG employs acquisition optimization strategies to select the final samples x'_{t-1} from x_t and x_{t-1}. This process is divided into three stages: the elitism strategy, the diversity strategy, and the roulette wheel selection strategy. These strategies respectively select 10%, 20%, and 70% of the architectures in the samples x'_{t-1}.

B.3.3 Hyperparameters of Neural Predictor

In evaluating the fitness of neural architectures, we first employs K-Means clustering to select a coreset $\mathcal{D}$ from the task dataset $\mathcal{T}$. This selection process actually samples a small subset from the task dataset, with K-Means clustering performed on each class of images respectively. The number of clusters is determined by the number of samples per class. The dataset encoder then encodes the coreset $\mathcal{D}$ into a latent vector $z_{\mathcal{D}}$, with the hidden dimension of $z_{\mathcal{D}}$ provided in our table. Simultaneously, the neural architecture is transformed into a graph, with the number

of node operation types specified in the table. The graph encoder subsequently encodes the neural architecture $\mathcal{G}$, and the Hidden Dimension of the encoded $z_{\mathcal{G}}$ is also detailed in the table.

B.4 Training Pipeline for Neural Architectures

For NAS-Bench-101, NAS-Bench-201 on CIFAR-10 and CIFAR-100, as well as TransNASBench-101-Micro and TransNASBench-101-Macro, we obtain the metrics of architectures on specified task datasets directly through lookup tables. This eliminates the need for independent training or testing.

B.4.1 NAS-BENCH-201

In the NAS-BENCH-201 search space, we select the top-1 architecture from the generated samples for training and evaluation. Our training pipeline is identical to that of DiffusionNAG and MetaD2A, ensuring a fair comparison of results. For image pre-processing, we perform a random horizontal flip with a 50% probability, crop a 32×32 patch with 4 pixels of padding on each edge, and normalize across the RGB channels. For each architecture, we use the SGD optimizer with Nesterov momentum and apply the cross-entropy loss over 200 training epochs. Additionally, we implement a weight decay of 0.0005 and a cosine annealing scheduler for the learning rate, which decays from 0.1 to 0.

B.4.2 MobileNetV3

In the MobileNetV3 search space, we select the top-30 architectures from the generated samples for training and evaluation. Image pre-processing includes resizing the images to 224×224 pixels using bicubic interpolation with antialiasing, followed by a random horizontal flip, a cutout with a length of 16 and normalization. During the training process, we employ auto-augmentation, a drop-path rate of 0.2, a weight decay of 0.0005, and a cosine annealing scheduler for the learning rate, which starts at 0.01 and decays to 0. Considering that the neural architecture in MobileNetV3 is more complex than in NAS-BENCH-201, we first pre-train a supernet on ImageNet-1K [22]. Subsequently, we fine-tune the supernet weights on CIFAR-10, CIFAR-100, Aircraft, and Oxford-IIIT Pets datasets respectively to adapt to image classification tasks specific to each task dataset. Then we inherit the weights from the supernet to the subnet, reinitialize the classifier randomly, and set its output size to match the number of classes in the task dataset. Finally, we train the final classification layer for 20 epochs and the total weights of network for 120 epochs using the SGD optimizer with cross-entropy loss.

B.5 Baselines

B.5.1 Classic NAS

Random Search [3] optimizes neural architectures by randomly selecting hyper-parameters, allowing exploration of the search space and potentially finding superior models. **NASNet-A** [28] is a reinforcement learning based NAS approach. It automates architecture design via RL controller that optimizes cell structures using validation accuracy as reward, enabling transferable convolutional cells from CIFAR-10 to ImageNet. **AmoebaNet-A** [18] employs regularized evolution with age-aware tournament selection, where older architectures are systematically removed from the population to prioritize newer candidates, balancing exploration and exploitation while evolving high-performance transferable cells. However, early classic NAS methods typically required thousands of GPU hours, thereby consuming substantial computational resources.

B.5.2 One-shot NAS

RSPS [15] enhances NAS by utilizing a one-shot strategy, where different architectures share weights during training, leading to efficient and scalable neural architecture optimization. **SETN** [6] introduces a selective candidate sampling strategy that leverages an evaluative model to predict validation losses of child architectures, thereby prioritizing competitive candidates during search iterations. **GDAS** [7] employs a differentiable architecture sampler via Gumbel-Softmax relaxation, facilitating direct gradient-based optimization of architectural parameters by linking sampler decisions to post-training validation performance. **PC-DARTS** [25] improves search efficiency through partial channel sampling and redundancy reduction. It implements edge normalization to stabilize architecture parameter learning while operating on subsets of channels, significantly reducing computational overhead. **DrNAS** [4] redefines architecture search as differentiable distribution learning, modeling architectural weights with Dirichlet distributions and optimizing their concentration parameters via gradient descent. Although it reduces the computational cost of frequently training neural architectures, one-shot NAS methods often suffer from unreliable architecture evaluation due to weight-sharing interference and supernet optimization bias.

B.5.3 Bayesian Optimization based NAS

BOHB [8] merges the strengths of Bayesian optimization with bandit-based configuration evaluation to achieve robust and efficient hyper-parameter optimization at scale in NAS. **HEBO** [5] employs non-linear input-output transformations and robust acquisition optimization to address heteroscedasticity and non-stationarity in NAS-oriented black-box hyperparameter tuning. **BANANAS** [24] utilizes Bayesian optimization for NAS through path-encoded architectural representations and aggregated neural network predictors, serving as surrogates. **NAS-BOWL** [19] combines Gaussian processes with Weisfeiler-Lehman topological kernels to guide architecture exploration in high-dimensional

Type	Method	CIFAR-10 Acc (%)	CIFAR-10 GPU Time (h)	CIFAR-100 Acc (%)	CIFAR-100 GPU Time (h)	Aircraft Acc (%)	Aircraft GPU Time (h)	Oxford-IIIT Pets Acc (%)	Oxford-IIIT Pets GPU Time (h)
Classic NAS	NASNet-A [28]	96.59	2000	80.03	-	-	-	-	-
	AmoebaNet-A [18]	96.66	3150	81.07	-	-	-	-	-
One-shot NAS	RSPS [15]	84.07	170	52.31	314	42.19	311.6	22.91	56
	SETN [6]	87.64	503.3	59.09	980	44.84	309.4	25.17	143.8
	GDAS [7]	93.61	418	70.70	859.7	53.52	308.4	24.02	116.1
	PC-DARTS [25]	93.66	173.3	66.64	332.5	26.33	58.7	25.31	47.4
	DrNAS [4]	94.36	362.7	**73.51**	575.5	46.08	575.5	26.73	100.3
BO-based NAS	BOHB [8]	93.61	792	70.85	-	-	-	-	-
	BANANAS [24]	**94.37**	11.8	**73.51**	45	-	-	-	-
	NASBOWL [19]	94.34	72	**73.51**	-	-	-	-	-
	TNAS [21]	**94.37**	52.3	**73.51**	52.3	59.15	21.4	40.00	21.4
NAG	MetaD2A [13]	**94.37**	50	73.34	50	57.71	20	39.04	20
	DiffusionNAG [1]	**94.37**	3.43	**73.51**	3.43	58.83	3.43	41.80	3.43
	DiNAS [2]	**94.37**	0.25	**73.51**	0.25	-	-	-	-
	EDNAG (Ours)	**94.37**	**<0.001**	**73.51**	**<0.001**	**60.14**	**0.011**	**46.17**	**0.016**

Table 3. Comprehensive Comparison of GPU Hours

NAS design spaces via Bayesian optimization. **TNAS** [21] utilizing Bayesian optimization with a deep-kernel Gaussian Process (GP) strategy to enhance the generalization of meta-learned dataset-aware predictor, thus improving the performance of architecture search. However, BO-based NAS still suffers from higher computational costs than gradient-based approaches and faces scalability limitations in high-dimensional search spaces.

B.5.4 Neural Architecture Generation

MetaD2A employs a straightforward unconditional architecture generative model to simultaneously generate neural architectures and subsequently evaluates the top-performing architectures by its neural predictor. **DiffusionNAG** [1] represents the groundbreaking application of diffusion models in neural architecture generation, which utilizes a diffusion process to perturb the distribution of architectures to a known prior distribution and then generate optimal architectures through a reverse diffusion process. During the denoising process, DiffusionNAG initiates with noise sampled from the known prior distribution and then iteratively removes noise predicted by a sophisticated pre-trained network, which is specifically trained to approximate the gradients of the architecture distribution. **DiNAS** [2] improves DiffusionNAG by employing discrete conditional graph diffusion processes with multi-conditioned guidance to generate Pareto-optimal neural architectures.

However, previous methods for neural architecture generation require training a generative model, leading to prohibitively high computational costs that made them impractical for consumer-grade GPUs. While our method, EDNAG, eliminates the need for training a generative model, enabling rapid architecture generation on consumer-grade GPUs. Specifically, it completes generation tasks in less than 70 seconds on an NVIDIA RTX 3060 GPU, marking a historic advancement in NAG methodologies.

C Supplementary Experiments

C.1 Analysis of Time Cost

We assess the GPU time for various neural architecture search baselines, as detailed in Table 3. For search-based methods, we measure search times, while for generation-based methods, we record both the training and inference times. One-shot NAS methods substantially reduce the time costs of traditional NAS methods, and BO-based NAS methods further lower GPU time to under 100 hours. Recent neural architecture generation methods, which directly generate architectures by training a generative model, require less than 10 hours of training with negligible inference time. Nevertheless, as a training-free NAG method, EDNAG not only eliminates generative model training time but also demonstrates remarkable advantages in inference time.

C.1.1 Analysis of Neural Predictor

We conduct experiments within the NAS-BENCH-201 search space and report Pearson correlation coefficients between predicted and actual accuracies, as illustrated in Figure 3. On CIFAR-10, the predictor of DiffusionNAG suffers from biased predictions due to random sampling, resulting in a lower correlation coefficient, while EDNAG achieves a 12.09% improvement. On CIFAR-100, the predictor of EDNAG achieves a 7.29% enhancement compared to DiffusionNAG. This experiment reveals that the neural predictor of EDNAG, which utilizes K-Means

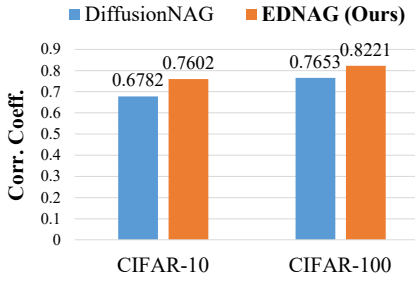

Figure 3. Performance analysis of neural predictor. We predict the accuracy of architectures for CIFAR-10 and CIFAR-100, and display the Pearson correlation coefficient between predicted and actual accuracies.

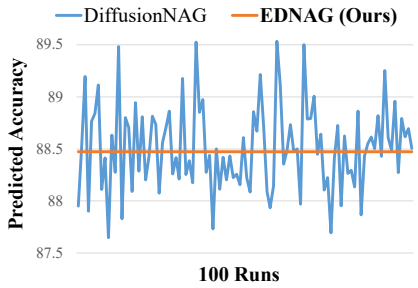

Figure 4. Stability analysis of neural predictor. With neural predictors in DiffusionNAG and EDNAG, we assessed the accuracy of identical neural architectures on consistent task datasets across 100 repeated runs.

clustering to select coreset images for dataset encoding, aligns more closely with real architectural accuracy. This approach effectively circumvents the limitations of random sampling in DiffusionNAG, which fails to preserve the uniformity of the target dataset distribution, resulting in biased predictions.

Moreover, during the denoising iterations, identical architectures in successive generations (x_t and x_{t-1}) necessitate multiple evaluations of the same architecture. Our experiments, shown in Figure 4, evaluate the same neural architecture over 100 repeated runs. The results demonstrate significant fluctuations in the predicted accuracy of DiffusionNAG due to random image sampling. In contrast, EDNAG demonstrates consistent predicted accuracy, effectively avoiding random perturbations in fitness values and further enhancing the stability of the whole generation process.

C.2 Quality Analysis of Generated Architectures

We futher analyze the statistical information of the generated neural architectures, as presented in Table 4. On the CIFAR-10 and CIFAR-100 datasets, the architectures generated by EDNAG successfully avoid lower-performing models. The minimum accuracies achieved are 88.61% and 67.14%, respectively, surpassing those of DiffusionNAG by 2.51% and 15.58%. Furthermore, the average accuracies of EDNAG reach 93.52% and 71.43%, exceeding DiffusionNAG by 0.42% and 1.55%, respectively. The highest accuracies attained are 94.37% and 73.51%, demonstrating results comparable to other baselines.

In addition, we evaluated the validity of the generated neural architectures, as shown in Table 5. In the NAS-BENCH-201 search space, a valid architecture is defined by three criteria: **(1)** the initial node is the input layer; **(2)** the terminal node is the output layer; and **(3)** none of the intermediate nodes are designated as input or output layers. An architecture that simultaneously satisfies all three conditions is considered valid. In the MobileNetV3 search space, validity is defined by four criteria: **(1)** the first node is the input layer; **(2)** the last node is the output layer; **(3)** no intermediate node is an input or output layer; and **(4)** within each stage, the number of layers that are not none layers ranges between 2 and 4. An architecture fulfilling all four conditions qualifies as valid.

GDSS [10], a state-of-the-art graph diffusion model for undirected graphs, performs poorly in both the NAS-BENCH-201 and MobileNetV3

Datasets	Metric	MetaD2A	DiffusionNAG	**EDNAG** (Ours)
CIFAR-10	Max	**94.37**	**94.37**	**94.37**
	Mean	91.52	93.13	**93.52**
	Min	10.00	86.44	**88.61**
CIFAR-100	Max	**73.51**	**73.51**	**73.51**
	Mean	67.14	70.34	**71.43**
	Min	1.00	58.09	**67.14**

Table 4. Statistics of generated architectures. Within the NAS-BENCH-201 search space and CIFAR-10 dataset, we record the maximum, minimum, and average accuracy of architectures produced in a single generation process.

Method	Valid Rate (%)	
	NAS-BENCH-201	MobileNetV3
GDSS	4.56	0.00
DiffusionNAG	**100.00**	42.17
EDNAG (Ous)	**100.00**	**98.70**

Table 5. Validity of generated architectures. We conduct neural architecture generation within two search spaces and subsequently report the valid rate of the generated architectures.

search spaces, scarcely generating any valid neural architectures. Although DiffusionNAG without positional embedding achieves a 100.00% valid rate in NAS-BENCH-201, it exhibites poor performance in the MobileNetV3 search space. In contrast, EDNAG not only attained a 100.00% valid rate in NAS-BENCH-201 but also demonstrated a high valid rate of 98.70% in the MobileNetV3 search space.

C.3 Ablation Study on Fitness-to-Probability Mapping Functions

The key of EDNAG lies in mapping fitness $f(x_t)$ of x_t to probability density of $\hat{x}_0$, enabling conditional architecture generation without training. To investigate the impact of different mapping functions on architecture generation, we conduct experiments in the NAS-BENCH-201 search space, evaluating three candidate functions: Identity, Power, and Energy, as shown in Figure 5. Specifically, the Identity function is the simplest form of mapping:

$$f(x) = x \cdot \exp\left(-\text{l2\_factor} \cdot \|x\|^2\right). \tag{22}$$

The Power function is given by:

$$f(x) = \left(\frac{x}{\text{temperature}}\right)^{\text{power}}. \tag{23}$$

And the Energy function is formulated as:

$$f(x) = \exp\left(\frac{x - x_{\max} + \epsilon}{\text{temperature}}\right). \tag{24}$$

On CIFAR-10, the energy mapping function achieves improvements in architecture accuracy of 0.43% and 0.25% compared to the identity and power mapping functions, respectively. While on CIFAR-100, the energy mapping function achieves improvements of 2.21% and 1.39% respectively. Consequently, the energy mapping function achieves SOTA performance on both CIFAR-10 and CIFAR-100. Thus, EDNAG employs the energy function to map fitness to the probabilistic space.

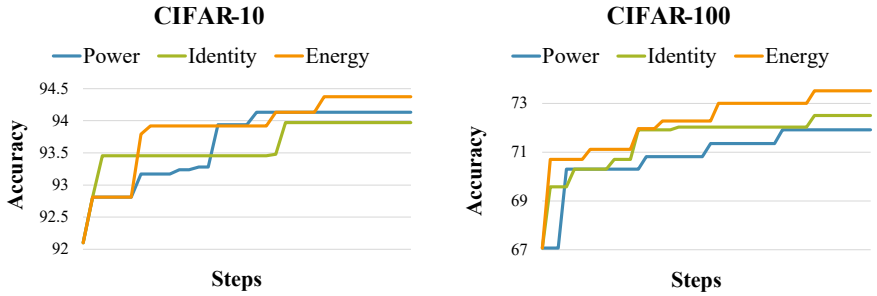

Figure 5. Ablation experiments on mapping functions. In the NAS-BENCH-201 search space, we employ Power, Identity, and Energy functions to map the fitness $f(x_t)$ of x_t to the probability density of $\hat{x}_0$. We display the highest accuracy of architecture samples during the iterative generation process on CIFAR-10 and CIFAR-100.

C.3.1 Analysis of Step Sensitivity in the Iterative Generation Process

The generation process of EDNAG corresponds to the reverse denoising iterative process in diffusion models, where the number of iterative steps plays a crucial role. To investigate the sensitivity of steps, we analyze architectural accuracy changes under different steps, as shown in Figure 6. With fewer steps, the average and minimum accuracies are relatively low, accompanied by significant fluctuations in the accuracy curves. Conversely, increasing the number of steps results in smoother and more stable accuracy changes, alongside significant improvements in the average, minimum, and maximum accuracies. Striking a balance between computational cost and generation performance, we adopt 100 steps as the optimal configuration.

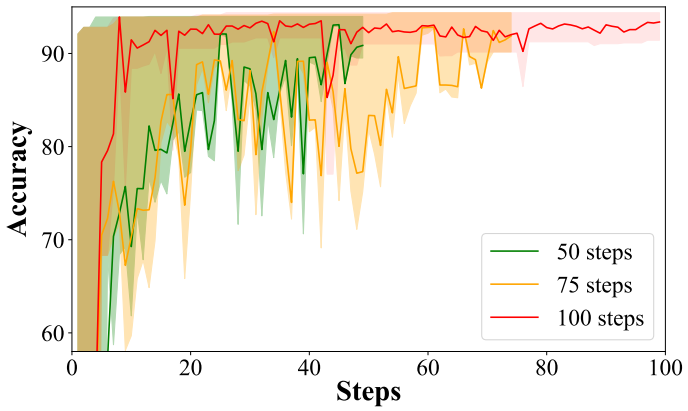

Figure 6. Ablation experiments on the number of iterative steps. We display the variation in accuracy of architecture samples over iterative steps in the CIFAR-10 dataset and NAS-BENCH-201 search space, using a total of 50, 75, and 100 steps.

References

[1] S. An, H. Lee, J. Jo, S. Lee, and S. J. Hwang. Diffusionnag: Predictor-guided neural architecture generation with diffusion models. *arXiv preprint arXiv:2305.16943*, 2023.

[2] R. Asthana, J. Conrad, Y. Dawoud, M. Ortmanns, and V. Belagiannis. Multi-conditioned graph diffusion for neural architecture search. *arXiv preprint arXiv:2403.06020*, 2024.

[3] J. Bergstra and Y. Bengio. Random search for hyper-parameter optimization. *The Journal of Machine Learning Research*, 13(1):281–305, 2012.

[4] X. Chen, R. Wang, M. Cheng, X. Tang, and C.-J. Hsieh. DrNAS: Dirichlet neural architecture search. In *International Conference on Learning Representations*, 2021.

[5] A. Cowen-Rivers, W. Lyu, R. Tutunov, Z. Wang, A. Grosnit, R.-R. Griffiths, A. Maravel, J. Hao, J. Wang, J. Peters, and H. Bou Ammar. HEBO: Pushing the limits of sample-efficient hyperparameter optimisation. *Journal of Artificial Intelligence Research*, 74, 07 2022.

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

[7] X. Dong and Y. 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.

[8] S. Falkner, A. Klein, and F. Hutter. BOHB: Robust and efficient hyperparameter optimization at scale. In *International Conference on Machine Learning*, pages 1437–1446. PMLR, 2018.

[9] J. Ho, A. Jain, and P. Abbeel. Denoising diffusion probabilistic models. *Advances in neural information processing systems*, 33:6840–6851, 2020.

[10] J. Jo, S. Lee, and S. J. Hwang. Score-based generative modeling of graphs via the system of stochastic differential equations. In *International conference on machine learning*, pages 10362–10383. PMLR, 2022.

[11] A. Krizhevsky, G. Hinton, et al. Learning multiple layers of features from tiny images. *Technical report*, 2009.

[12] A. Krizhevsky, I. Sutskever, and G. E. Hinton. Imagenet classification with deep convolutional neural networks. *Advances in neural information processing systems*, 25, 2012.

[13] H. Lee, E. Hyung, and S. J. Hwang. Rapid neural architecture search by learning to generate graphs from datasets. *arXiv preprint arXiv:2107.00860*, 2021.

[14] J. Lee, Y. Lee, J. Kim, A. Kosiorek, S. Choi, and Y. W. Teh. Set transformer: A framework for attention-based permutation-invariant neural networks. In *International conference on machine learning*, pages 3744–3753. PMLR, 2019.

[15] L. Li and A. Talwalkar. Random search and reproducibility for neural architecture search. In *Uncertainty in Artificial Intelligence*, pages 367–377. PMLR, 2019.

[16] S. Maji, E. Rahtu, J. Kannala, M. Blaschko, and A. Vedaldi. Fine-grained visual classification of aircraft. *arXiv preprint arXiv:1306.5151*, 2013.

[17] O. M. Parkhi, A. Vedaldi, A. Zisserman, and C. Jawahar. Cats and dogs. In *2012 IEEE conference on computer vision and pattern recognition*, pages 3498–3505. IEEE, 2012.

[18] E. Real, A. Aggarwal, Y. Huang, and Q. V. Le. Regularized evolution for image classifier architecture search. In *Proceedings of the aaai conference on artificial intelligence (AAAI)*, 2019.

[19] B. Ru, X. Wan, X. Dong, and M. Osborne. Interpretable neural architecture search via bayesian optimisation with weisfeiler-lehman kernels. In *International Conference on Learning Representations*, 2021. URL https://openreview.net/forum?id=j9Rv7qdXjd.

[20] O. Russakovsky, J. Deng, H. Su, J. Krause, S. Satheesh, S. Ma, Z. Huang, A. Karpathy, A. Khosla, M. Bernstein, et al. Imagenet large scale visual recognition challenge. *International journal of computer vision*, 115:211–252, 2015.

[21] G. Shala, T. Elsken, F. Hutter, and J. Grabocka. Transfer NAS with meta-learned bayesian surrogates. In *The Eleventh International Conference on Learning Representations*, 2023.

[22] P. Wang, A. Yang, R. Men, J. Lin, S. Bai, Z. Li, J. Ma, C. Zhou, J. Zhou, and H. Yang. Ofa: Unifying architectures, tasks, and modalities through a simple sequence-to-sequence learning framework. In *International conference on machine learning*, pages 23318–23340. PMLR, 2022.

[23] W. Wen, H. Liu, Y. Chen, H. Li, G. Bender, and P.-J. Kindermans. Neural predictor for neural architecture search. In *European Conference on Computer Vision*, pages 660–676. Springer, 2020.

[24] C. White, W. Neiswanger, and Y. Savani. Bananas: Bayesian optimization with neural architectures for neural architecture search. In *Proceedings of the AAAI Conference on Artificial Intelligence*, 2021.

[25] Y. Xu, L. Xie, X. Zhang, X. Chen, G.-J. Qi, Q. Tian, and H. Xiong. Pc-darts: Partial channel connections for memory-efficient architecture search. In *International Conference on Learning Representations (ICLR)*, 2020.

[26] C. Ying, A. Klein, E. Christiansen, E. Real, K. Murphy, and F. Hutter. Nas-bench-101: Towards reproducible neural architecture search. In *International conference on machine learning*, pages 7105–7114. PMLR, 2019.

[27] Y. Zhang, B. Hartl, H. Hazan, and M. Levin. Diffusion models are evolutionary algorithms. *arXiv preprint arXiv:2410.02543*, 2024.

[28] B. Zoph, V. Vasudevan, J. Shlens, and Q. 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.