# Multi-conditioned Graph Diffusion for Neural Architecture Search

**Anonymous authors**
**Paper under double-blind review**

## Abstract

Neural architecture search automates the design of neural network architectures usually by exploring a large and thus complex architecture search space. To advance the architecture search, we present a graph diffusion-based NAS approach that uses discrete conditional graph diffusion processes to generate high-performing neural network architectures. We then propose a multi-conditioned classifier-free guidance approach applied to graph diffusion networks to jointly impose constraints such as high accuracy and low hardware latency. Unlike the related work, our method is completely differentiable and requires only a single model training. In our evaluations, we show promising results on six standard benchmarks, yielding novel and unique architectures at a fast speed, i.e. less than 0.2 seconds per architecture. Furthermore, we demonstrate the generalisability and efficiency of our method through experiments on ImageNet dataset.

## 1 Introduction

The design of neural network architectures has been normally a manual and time-consuming task, requiring domain expertise and trial-and-error experimentation (Elsken et al., 2019). Neural Architecture Search (NAS) addresses this limitation by leveraging data-driven methods to automatically search for well-performing neural network architectures (Liu et al., 2019; Howard et al., 2019; Pham et al., 2018). Existing works in NAS mostly represent the architectures as graphs and include search based methods (Li & Talwalkar, 2020; White et al., 2021b), reinforcement learning (Zoph & Le, 2017; Tian et al., 2020), and evolution-based approaches (Real et al., 2019; Chu et al., 2020). However, the large size of the architecture search space makes it challenging for these methods to search for high-performing topologies.

To accelerate the architecture search, generative methods reduce the search queries by learning the architecture search space and optimising the latent space from which a generator network draws architectures (Rezaei et al., 2021; Huang & Chu, 2021). These methods not only enhance the efficiency but also capture intricate architecture distributions, generating novel architectures. However, the choice of graph generative model significantly impacts the NAS search time. The existing methods employ complex GAN-based generators (Rezaei et al., 2021) or use computationally intensive supernets (Huang & Chu, 2021). More recently, An et al. (2023) employs conditional diffusion processes guided by a classifier while Lukasik et al. (2022) uses a simple generator paired with a surrogate predictor. However, these methods require separate predictor networks for the generated architecture performance. Hence, we present a diffusion-based generative approach that is completely differentiable and thus training involves only a single model. As a result, we reach promising performance with much smaller search time.

Denoising diffusion probabilistic models (DDPMs) (Ho et al., 2020) have recently gained attention because of their ability to effectively model complex data distributions through an iterative denoising process. DDPMs offer precise generative control, improving distribution coverage compared to other generative models (Dhariwal & Nichol, 2021). This characteristic makes diffusion models particularly appealing for NAS, as they fulfil the requirement to generate neural network architectures and eventually facilitate the exploration of the search space. In addition to their superior performance, diffusion models excel in conditional generation through the classifier-free guidance technique (Ho & Salimans, 2021). This technique enables the condition-

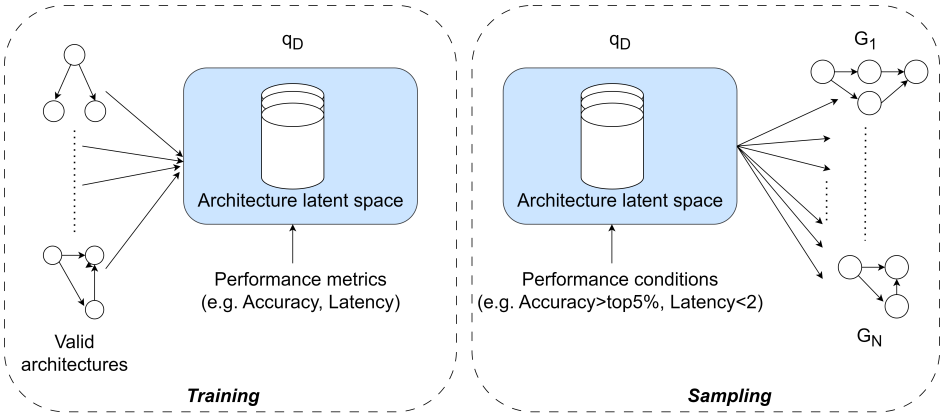


Figure 1: Overview of our approach. We train a discrete graph diffusion model (denoted as $q_D$) on valid architectures from the architecture search space along with their performance metrics (eg. accuracy, latency). After training, we sample architectures given the required conditions (eg. accuracy>top5%, latency<2).

ing of diffusion models on a specific target class, allowing the model to generate samples belonging to that class without utilizing the gradients of an external classifier. Previous studies show that along with image synthesis, classifier-free guidance works well in molecule (graph) synthesis using graph diffusion networks (Hoogeboom et al., 2022). Moreover, Giambi & Lisanti (2023) demonstrated the capability of classifier-free guidance approach using multiple conditions. Motivated by this idea, we present a multi-conditioned graph diffusion model in which constraints such as high model accuracy and low latency jointly contribute to architecture sampling.

We introduce a graph diffusion-based NAS approach (DiNAS), depicted in Figure 1, that utilises discrete conditional graph diffusion processes to generate high-performing neural network architectures [1]. We leverage classifier-free (CF) guidance, initially developed for image tasks, and extend its application to graph models. Additionally, to impose multiple constraints, we utilize a multi-conditioned CF guidance technique, and apply it within our graph diffusion framework. To demonstrate the effectiveness of our proposed method, we perform extensive evaluations on six standard benchmarks, including experiments on ImageNet (Deng et al., 2009), and ablation studies to demonstrate state-of-the-art performance and faster generation rate (less than 0.2 seconds per architecture on a single GPU) compared to the prior work. To the best of our knowledge, this is the first formulation of NAS using multi-conditioned graph-based diffusion models. In summary, our contributions are as follows:

- We introduce a differentiable generative NAS method, which employs discrete conditional diffusion processes to learn the architecture latent space by training a single model.
- We propose a multi-conditioned diffusion guidance technique for graph diffusion networks, effectively applied within our NAS approach.
- We demonstrate promising results in six standard benchmarks while using less or same number of queries with rapid generation of novel and unique high-performing architectures.

## 2 Related Work

**Neural Architecture Search (NAS)** Automating the neural network architectural design has gained substantial interest in the past few years (Liu et al., 2019; Jin et al., 2019; Zoph et al., 2018; Bender et al., 2018; Shala et al., 2023). A straightforward approach is to randomly select and evaluate architectures from the search space (Li & Talwalkar, 2020). However, the lack of optimisation in the search space makes this approach inefficient. To address this limitation, earlier works rely on reinforcement learning (Zoph & Le,

[1]The code will be made available upon paper acceptance.

2017; Baker et al., 2017; Franke et al., 2021) to discover well-performing architectures. Gradient-based approaches (Brock et al., 2018; Chen et al., 2021b; Yang et al., 2020) employ gradient-based optimisation, while evolutionary methods (Real et al., 2019; 2017) deploy evolutionary algorithms to perform the search. Although these approaches exhibit faster search pace than random search due to the optimisation, they are still regarded slow in searching high-performing architectures (Liu et al., 2019). Another major challenge with search-based methods is the requirement to train networks at each iteration (Luo et al., 2022). This becomes particularly problematic when NAS approaches require a substantial number of iterations to generate well-performing architectures, which is often the case with reinforcement learning-based methods. This issue is solved by the recently developed generative methods (Lukasik et al., 2021; Rezaei et al., 2021; Lukasik et al., 2022), which reduce the search time by learning the architecture search space. Following the same direction, we present a generative model that remarkably reduces the search time compared to the prior work, while minimising the performance loss.

**Diffusion Models** Although the original idea of data generation through diffusion goes back several years (Sohl-Dickstein et al., 2015), diffusion models later gained popularity for image (Ho et al., 2020; Rombach et al., 2022; Saharia et al., 2022)), text (Austin et al., 2021) and more recently graph generation (Wang et al., 2022). The ability of diffusion models to effectively synthesise graphs motivates our work to formulate NAS as a graph generation problem. Nevertheless, the generation of well-performing architectures requires conditional generation through guidance, e.g. by specifying the minimum architecture accuracy. Hence we utilize a formulation of classifier-free guidance for graph-diffusion networks. Then, we introduce a multi-conditioned graph-diffusion approach that accounts for several constraints in the architecture generation.

## 3 Background

### 3.1 Denoising Diffusion Probabilistic Models

Denoising Diffusion Probabilistic Models (DDPMs) (Ho et al., 2020) comprise two fundamental processes, namely, forward and reverse processes. The forward process sequentially corrupts the data sample $\mathbf{x}$ using a noise model $q$ that follows a Gaussian distribution until $\mathbf{x}$ reaches a state of pure noise. The noisy variants of $\mathbf{x}$ are denoted as $(\mathbf{x^1}, \mathbf{x^2}, \ldots, \mathbf{x^T})$, where $T$ represents the total number of corruption steps. Subsequently, the reverse process involves learning a denoising model represented as a deep neural network $\phi_\theta$ with parameters $\theta$ to estimate the noise state of sample $\mathbf{x}$ at time step $t-1$, i.e. $\mathbf{x}^{t-1}$ given the current state $\mathbf{x}^t$. This is achieved using a scoring function that maximises the likelihood of $\mathbf{x}^{t-1}$. Formally, the scoring function is defined as $S_F = \nabla_{\mathbf{x}^{t-1}} \log p_\theta(\mathbf{x}^{t-1}|\mathbf{x}^t)$ which corresponds to the gradient of the log-likelihood with respect to state $\mathbf{x}^{t-1}$. Following the network training, another data point can be sampled from a noisy prior (denoted as $\mathbf{z}^T$), and by iterative denoising the data point (i.e. predicting $\mathbf{z}^{t-1}$ from $\mathbf{z}^t$), a sample $\mathbf{z}^0$ is obtained, which corresponds to the original data distribution. This process is referred to as the sampling process.

Diffusion models can generate high-quality samples from complex data distributions. However, in our task, we do not intend to sample from the entire distribution but a subset of it containing high-performing and/or low latency architectures. Hence, diffusion models need to be modified to incorporate conditioning. This can be achieved using conditional diffusion models.

### 3.2 Conditional diffusion models with guidance

The conditional diffusion model estimates the distribution $p_\theta(\mathbf{x}^{t-1}|\mathbf{x}^t, y)$. From Bayes rule, we have:

$$p_\theta(\mathbf{x}^{t-1}|\mathbf{x}^t, y) \propto p_\theta(\mathbf{x}^t, y|\mathbf{x}^{t-1})p(\mathbf{x^{t-1}}). \tag{1}$$

To ensure the balance between sampling diversity and quality, generative models can incorporate guidance scale $\gamma$, modifying the Eq. 1 to:

$$p_\theta(\mathbf{x}^{t-1}|\mathbf{x}^t, y) \propto p_\theta(\mathbf{x}^t, y|\mathbf{x}^{t-1})^\gamma p(\mathbf{x^{t-1}}). \tag{2}$$

Specifically, increasing $\gamma$ sharpens the distribution which favors enhanced sample quality at the expense of sample diversity during the sampling process, referred to as guidance in diffusion models. To guide a diffusion model for labelled data, the model is conditioned on the classification target $y$ and the score function $\nabla_{\mathbf{x}^{t-1}} \log p_\theta(\mathbf{x}^{t-1}|\mathbf{x}^t, y)$ is computed. Dhariwal & Nichol (2021) approach this problem using an external classifier (parameterised by $\psi$) where the score function $S_F$ is modified to include the gradients of the classifier. The reformulated score function is then the weighted sum of the unconditional score function and the conditioning term obtained by the classifier, defined as:

$$\nabla_{\mathbf{x}^{t-1}} \log p_\theta(\mathbf{x}^{t-1}|\mathbf{x}^t, y) = \nabla_{\mathbf{x}^{t-1}} \log p_\theta(\mathbf{x}^{t-1}|\mathbf{x}^t) + \gamma \nabla_{\mathbf{x}^{t-1}} \log p_\psi(y|\mathbf{x}^{t-1}). \tag{3}$$

While we have successfully expressed the reverse denoising process as a weighted sum of two score functions, estimation of the conditional score function requires training a separate classifier. Moreover, calculating $\log p_\psi(y|\mathbf{x}^{t-1})$ requires inferring $y$ from noisy data $\mathbf{x}^t$. Although feeding noisy data to the classifier yields decent performance, it disrupts the robustness of the model since it ignores most of the original input signal. To address this issue, Ho & Salimans (2021) came up with the classifier-free guidance, which develops the classifier using the generative model itself. In this case, the score function is defined as:

$$\nabla_{\mathbf{x}^{t-1}} \log p_{\theta_\gamma}(\mathbf{x}^{t-1}|\mathbf{x}^t, y) = (1-\gamma) \nabla_{\mathbf{x}^{t-1}} \log p_\theta(\mathbf{x}^{t-1}|\mathbf{x}^t) + \gamma \nabla_{\mathbf{x}^{t-1}} \log p_\theta(\mathbf{x}^{t-1}|\mathbf{x}^t, y). \tag{4}$$

Eq. 4 demonstrates that it is possible to achieve the same behaviour as the classifier-based guidance without explicitly using a classifier. This is achieved through a weighted sum, specifically a barycentric combination, of the conditional and unconditional score functions.

### 3.3 Discrete Graph Diffusion

Diffusion models typically work in a continuous space and apply Gaussian noise to the data (Ho et al., 2020; Saharia et al., 2022). Training a diffusion model to generate graphs in the same manner, however, leads to the loss of graph sparsity and structural information. DiGress, a discrete diffusion approach proposed by Vignac et al. (2023), addresses this problem with a Markov processes as discrete noise model. In this case, the graph comprises of nodes and edges, both being categorical variables, and the goal is to progressively add or remove edges as well as change graph node categories. Hence, the diffusion process is applied on the node categories $\mathbf{X}$ and edges $\mathbf{E}$. Eventually, this model solves a simple classification task for nodes and edges instead of a complex distribution learning task, normally performed in generative models like VAEs (Kingma & Welling, 2013) or DDPMs (Ho et al., 2020). Our approach, in principle, follows DiGress to generate graphs which correspond to neural network architectures.

At each forward step, discrete marginal noise is added to both $\mathbf{X}$ and $\mathbf{E}$ using the transition probability matrices $Q_X$ and $Q_E$ respectively, which incorporate the marginal distributions $m'_X$ and $m'_E$. We select the noisy prior distribution such that it is close to the original data distribution. Then, the transition matrices are defined as follows:

$$Q_X^t = \bar{a}^t I + (1-\bar{a}^t) 1_i m'_X; \quad Q_E^t = \bar{a}^t I + (1-\bar{a}^t) 1_j m'_E, \tag{5}$$

where $I$ is the identity matrix, $1_i$ and $1_j$ are the indicator functions, $t$ is the time-step, and $\bar{a}^t$ is the cosine schedule defined as $\bar{a}^t = \cos(0.5\pi(t/T+s)/(1+s))^2$ with $s$ close to 0.

**Training** For the reverse (denoising) step, a Graph Transformer network $\phi_\theta$, parameterised by $\theta$, is employed. This network learns the mapping between the noisy graphs $\mathbf{G^t}$ and the corresponding clean graphs $\mathbf{G}$. During training, $\phi_\theta$ can take noisy graphs at any time step $t \in (1, .., T)$ to predict the clean graph. The loss functions for $\mathbf{X}$ and $\mathbf{E}$ are based on the cross-entropy between their respective predicted probabilities $\hat{p}^G = (\hat{p}^X, \hat{p}^E)$ and the ground-truth graph $\mathbf{G} = (\mathbf{X}, \mathbf{E})$. The total loss is then, a weighted sum of node-level and edge-level losses, which is given by:

$$L_G(\hat{p}^X, \mathbf{X}, \hat{p}^E, \mathbf{E}) = \sum_{1 \le i \le n} CE(x_i, \hat{p}_i^X) + \lambda \sum_{1 \le i,j \le n} CE(e_{ij}, \hat{p}_{ij}^E), \tag{6}$$

where $CE$ is the cross-entropy loss function, $\lambda$ is a parameter to weight the importance of nodes and edges and, n is the number of nodes.

**Sampling** Let the posterior distribution be $p_\theta$. We start from a noisy prior distribution $\mathbf{G}^T \sim (q_X(n_T) \times q_E(n_T))$, where $n_T$ is sampled from the node distribution in the training data. Then, we estimate the node and edge distributions $p_\theta(x_i^{t-1}|\mathbf{G}^t)$ and $p_\theta(e_{ij}^{t-1}|\mathbf{G}^t)$ using the predicted probabilities $\hat{p}_i^X$ and $\hat{p}_{ij}^E$. This can be written as:

$$p_\theta(x_i^{t-1}|\mathbf{G}^t) = \sum_{x \in \mathbf{X}} p_\theta(x_i^{t-1}|x_i = x, \mathbf{G}^t)\hat{p}_i^X(x); \quad p_\theta(e_{ij}^{t-1}|\mathbf{G}^t) = \sum_{e \in \mathbf{E}} p_\theta(e_{ij}^{t-1}|eij = e, \mathbf{G}^t)\hat{p}_{ij}^E(e). \tag{7}$$

Finally, sampling new graphs can be seen as iteratively estimating the distribution $p_\theta(\mathbf{G}^{t-1}|\mathbf{G}^t)$ until a clean graph $\mathbf{G}^0$ is obtained. $p_\theta(\mathbf{G}^{t-1}|\mathbf{G}^t)$ can be seen as the product of the node and edge distributions marginalised over predictions from the network $\phi_\theta$:

$$p_\theta(\mathbf{G}^{t-1}|\mathbf{G}^t) = \prod_{1 \leq i \leq n} p_\theta(x_i^{t-1}|\mathbf{G}^t) \prod_{1 \leq i,j \leq n} p_\theta(e_{ij}^{t-1}|\mathbf{G}^t). \tag{8}$$

The above model successfully handles sparse categorical graph data in a discrete manner, synthesising graphs from complex data distributions. Therefore, it is suitable for our problem. Nevertheless, in our task, we seek to introduce conditioning in discrete graph diffusion models through classifier-free (CF) guidance. To that end, we propose next a multi-conditioned graph diffusion formulation for NAS.

## 4 Method

Consider the diffusion model $q_D$ comprising of a neural network $\phi_\theta$ paramterised by $\theta$. During training, the model $q_D$ takes the directed acyclic graph $\mathbf{G}$ as input and learns to reconstruct $\mathbf{G}$ from the noisy version $\mathbf{G^t}$, where $t \in (1, \ldots, T)$ is the number of diffusion time steps. This reconstruction is essentially performed by learning to estimate the actual data distribution $\mathcal{G}$ from the noisy version of $\mathcal{G}$, which we denote as $P_N$, through iterative denoising. Following the training of $\phi_\theta$, we aim to generate DAGs representing high-performing neural network architectures using samples from $P_N$, where we denote a sample as $\mathbf{z}$.

Our directed acyclic graph (DAG) representation of architectures follows the standard cell-based NAS search spaces (Liu et al., 2019; Klyuchnikov et al., 2020), where each cell is a DAG. $\mathbf{G}$ consists of a set of nodes and edges. The sequence of nodes in $\mathbf{G}$ is represented by $\mathbf{X} = [v_1, v_2 \ldots v_n]$, where the number of nodes is $n$, and the edges as the adjacency matrix $\mathbf{E}$ of shape $(n, n)$. Hence, each DAG is represented by $\mathbf{G} = (\mathbf{X}, \mathbf{E})$. Each node is a categorical variable, describing operations, e.g., 1x1 convolution, while each edge is a binary variable, specifying the presence or absence of the connection between nodes. In addition, $\mathbf{G}$ maps to the ground-truth performance metrics $P$ e.g., the accuracy and latency of each DAG.

Our objective is twofold, namely to generate valid cells $C_v = (\mathbf{X_v}, \mathbf{E_v})$ from the latent variable $\mathbf{z}$, sampled from the noise distribution $P_N$ and, second, to learn the mapping between the valid cell $C_v$ and its corresponding performance metrics $P$. The learned mapping is then used to generate high-performing cells with accuracy close to the maximum achievable accuracy or cells with latency below a certain latency constraint. Note that a cell is valid when the corresponding DAG is connected and includes a realistic sequence of nodes.

### 4.1 Diffusion based NAS

We consider the unconditional and conditional graph generation. First, we present the unconditional model that learns to generate valid cells. Since some of the generated cells might have poor performance, we propose the single conditioned and multi-conditioned graph diffusion models to generate just the high-performing cells based on metrics like the model accuracy and latency.

#### 4.1.1 Unconditional model

Our unconditional model is based on the discrete denoising graph diffusion model (Vignac et al., 2023), outlined in Section 3.3. The forward process involves adding discrete marginal noise $Q_X^t$ and $Q_E^t$ (Eq. 5) to both nodes $\mathbf{X}$ and edges $\mathbf{E}$ respectively. To perform denoising, we employ the Graph Transformer network $\phi_\theta$, which is trained to predict clean graphs $\mathbf{G}$ from noisy graphs $\mathbf{G^t}$. While this model effectively captures the data distribution for undirected graphs, it lacks the ability to incorporate directional information of DAGs. This directional information depicts the flow of data from input to output in the cells and hence is crucial for generating valid cells. To address this limitation, we integrate into our model the positional encoding technique by Vaswani et al. (2017b). In detail, we add sinusoidal signals of different frequencies to the node features $\mathbf{X}$ before passing them through the Graph Transformer $\phi_\theta$, thereby enhancing the network's capability to consider sequential information.

Despite the ability of our unconditional model in forming valid cells necessary for complete network architectures, our goal lies in generating a particular subset of the learned architecture distribution comprising high-performing cells. To that end, we first condition our model on the accuracy metric.

#### 4.1.2 Conditional model

To achieve the generation of high-performing architectures, we propose a guidance approach, inspired by the classifier-free guidance (Ho & Salimans, 2021), and integrate it to our unconditional graph diffusion model. Unlike the unconditional model, our conditional model estimates the distribution $p_\theta(\mathbf{G}^{t-1}|\mathbf{G}^t, a)$, essentially by computing the score function $\nabla_{\mathbf{G}^{t-1}} \log p_{\theta_\gamma}(\mathbf{G}^{t-1}|\mathbf{G}^t, a)$ as:

$$\nabla_{\mathbf{G}^{t-1}} \log p_{\theta_\gamma}(\mathbf{G}^{t-1}|\mathbf{G}^t, a) = (1-\gamma)\nabla_{\mathbf{G}^{t-1}} \log p_\theta(\mathbf{G}^{t-1}|\mathbf{G}^t) + \gamma \nabla_{\mathbf{G}^{t-1}} \log p_\theta(\mathbf{G}^{t-1}|\mathbf{G}^t, a), \tag{9}$$

where $\gamma$ is the guidance scale. The first term of Eq. 9 corresponds to the unconditional distribution $p_\theta(\mathbf{G}^{t-1}|\mathbf{G}^t)$ learning, while the second one corresponds to the conditional distribution $p_\theta(\mathbf{G}^{t-1}|\mathbf{G}^t, a)$ learning. Following Ho & Salimans (2021), we remove the conditioning information for some forward passes determined by the control parameter $\epsilon$. This leads to the unconditional training of the network. For the rest forward passes, we keep this information to enable conditional training.

#### 4.1.3 Discretisation of the target variable

Our guidance approach assumes that the target variable $y$, e.g. accuracy or latency, belongs to a finite set such that $y \in \{y_1, y_2, \ldots, y_w\} \subseteq \mathbb{R}$, where $w$ is the number of possible values of $y$. However, in our case, $y$ takes continuous values from the real number domain, $\mathbb{R}^+$. To address this issue, we split $y$ into $d$ discrete classes based on their value. The choice of the split affects the balance of the class data distribution. In our implementation (Sec. 5.1), we provide information on how we select the number of classes and splits according to the problem.

### 4.2 Incorporating Multiple conditions

Next, we introduce multiple conditions to the diffusion guidance to impose several constraints. Consider the unconditional noise model $q$ that corrupts the data progressively for $t$ time steps. Our objective is to estimate the reverse conditional diffusion process $\hat{q}(\mathbf{G}^{t-1}|\mathbf{G}^t, y_1, y_2, ..., y_k)$, given the $k$ independent conditions $y_1, y_2, ..., y_k$. Assuming $p_\theta(\mathbf{G}^{t-1}|\mathbf{G}^t, y_1, y_2, ..., y_k)$ approximates $\hat{q}(\mathbf{G}^{t-1}|\mathbf{G}^t, y_1, y_2, ..., y_k)$, we perform the estimation of reverse conditional diffusion process by computing the score function $\nabla_{\mathbf{G}^{t-1}} \log p_{\theta_\gamma}(\mathbf{G}^{t-1}|\mathbf{G}^t, y_1, ..., y_k)$ as:

$$\begin{aligned}\nabla_{\mathbf{G}^{t-1}} \log p_{\theta_\gamma}(\mathbf{G}^{t-1}|\mathbf{G}^t, y_1, ..., y_k) = (1-\gamma)\nabla_{\mathbf{G}^{t-1}} \log p_\theta(\mathbf{G}^{t-1}|\mathbf{G}^t) \\ + \gamma \nabla_{\mathbf{G}^{t-1}} \log p_\theta(\mathbf{G}^{t-1}|\mathbf{G}^t, y_1, ..., y_k),\end{aligned} \tag{10}$$

where $\gamma$ is the guidance scale. The derivation is provided in Appendix A.1. Similar to the standard single-conditioned guidance (Eq. 9), the conditional score function for multi-conditioned guidance can be expressed

as a weighted sum of conditional and unconditional score function. These score functions can be computed using two forward passes of our network, the unconditional and conditional forward pass.

### 4.3 Training and Sampling

**Training procedure** Let $c_1, \ldots, c_k$ denote the metrics, which we want to constrain e.g. accuracy or hardware latency. The training procedure starts by randomly selecting the time-step $t$ from the range $(1, .., T)$. Subsequently, the performance metrics $P = (c_1, ..., c_k)$ undergo a substitution with a null token $\emptyset$ for a probability of $\epsilon$ instances. Then, marginal noise is introduced to both $\mathbf{X}$ and $\mathbf{E}$ for a duration of $t$ time-steps. Next, each of $c_1, .., c_k$ is individually processed through distinct embeddings, with the resultant embeddings being included to both $\mathbf{X}$ and $\mathbf{E}$. We then apply positional encoding to $\mathbf{X}$. Finally, the resultant graph is provided as input to our Graph Transformer network $\phi_\theta$. This network then generates the denoised graph $(\mathbf{X}, \mathbf{E})$ which is used to calculate the loss (Eq. 6). We present the training algorithm in Alg. 1.

**Sampling procedure** Let $(\hat{c}_1, .., \hat{c}_k)$ be the constraints desired to be imposed (e.g. $\hat{c}_1$=top 5%). The sampling procedure is initiated with sampling a random noisy graph $\mathbf{G^t}$ from the prior distribution $(q_X(n_T) \times q_E(n_T))$. Next, we apply the positional encoding to X and perform two forward passes of our trained network $\phi_\theta$, namely the unconditional and conditional pass. In the unconditional pass, $(c_1 = \emptyset, c_2 = \emptyset, .., c_k = \emptyset)$, where $\emptyset$ is a null token, whereas for the conditional pass, $(c_1 = \hat{c}_1, ..., c_k = \hat{c}_k)$. Then, the score estimates are computed for both functions ($\hat{p}_c$ for conditional and $\hat{p}_u$ for unconditional). Lastly, we calculate the resulting score by a linear combination of the score estimates and sample a less noisy graph $\mathbf{G}^{t-1}$ with Eq. 7 and 8. This is iteratively performed to produce the clean graph $\mathbf{G}^0$. The sampling algorithm is presented in Alg. 2 and implementation details in Appendix Sec. A.2.

**Algorithm 1** Training DiNAS

**Input**: $\mathbf{G}^0 = (\mathbf{X}, \mathbf{E}, c_1, ..., c_k)$ and $\epsilon$
$t \sim \nu(1, .., T)$ ▷ Sample t randomly from $(1, ...T)$
$c_1, ..., c_k \leftarrow \emptyset$ with probability $\epsilon$ ▷ Conditional dropout to train unconditionally
$\mathbf{G}^t \leftarrow (\mathbf{X}Q_X^t, \mathbf{E}Q_E^t, c_1, ..., c_k)$ ▷ Apply marginal noise for $t$ time steps
$\mathbf{G}^t \leftarrow (\mathbf{G}^t, Emb(c_1), ..., Emb(c_k))$ ▷ Append embeddings to nodes and edges
$\mathbf{X}^t \leftarrow \mathbf{X}^t + PosEnc(\mathbf{X}^t)$ ▷ Add sinusoids to X for positional encoding
$\hat{p}^X, \hat{p}^E \leftarrow p_\theta(\mathbf{G}^t | c_1 ..., c_k)$ ▷ Forward pass
$optimiser.step(L_G(\hat{p}^X, \mathbf{X}, \hat{p}^E, \mathbf{E}))$ ▷ Calculate loss and optimise (Eq. 6)

**Algorithm 2** Sampling from DiNAS

**Input**: guidance scale $\gamma$, and conditions $\hat{c}_1, .., \hat{c}_k$
Sample $n_T$ number of nodes from training data distribution
Sample random graph $\mathbf{G}^t \sim (q_X(n_T) \times q_E(n_T))$ ▷ Sample from prior distribution
**for** $t = T$ to 1 **do**
  $\mathbf{X}^t \leftarrow \mathbf{X}^t + PosEnc(\mathbf{X}^t)$ ▷ Add sinusoids to X for positional encoding
  $\hat{p}_u^X, \hat{p}_u^E = p_\theta(\mathbf{G}^t | c_1 = \emptyset, ..., c_k = \emptyset)$ ▷ Unconditional forward pass
  $\hat{p}_c^X, \hat{p}_c^E = p_\theta(\mathbf{G}^t | c_1 = \hat{c}_1, .., c_k = \hat{c}_k)$ ▷ Conditional forward pass
  $\hat{p}^X = (1 - \gamma)\hat{p}_u^X + \gamma \hat{p}_c^X$ ▷ Linear combination of score estimates
  $\hat{p}^E = (1 - \gamma)\hat{p}_u^E + \gamma \hat{p}_c^E$ ▷ Linear combination of score estimates
  Calculate $p_\theta(x_i^{t-1} | \mathbf{G}^t)$ and $p_\theta(e_{ij}^{t-1} | \mathbf{G}^t)$ ▷ Eq. (7)
  $\mathbf{G}^{t-1} \sim \prod_i p_\theta(x_i^{t-1} | \mathbf{G}^t) \prod_{ij} p_\theta(e_{ij}^{t-1} | \mathbf{G}^t)$ ▷ Sample $\mathbf{G}^{t-1}$ (Eq. 8)
**end for**
**return** $\mathbf{G}^0$

# 5 Experiments

We evaluate our approach on six standard benchmarks- encompassing tabular, surrogate, hardware aware benchmarks, and the challenging ImageNet image classification task (Deng et al., 2009).

## 5.1 Experimental Setup

**Tabular Benchmarks** We first consider the tabular benchmarks- NAS-Bench-101 (Ying et al., 2019) and NAS-Bench-201 (Dong & Yang, 2020) for our experiments. Tabular benchmarks list unique architectures with their corresponding accuracy. We utilise the validation accuracy as performance metrics $P$. The evaluation protocol [2] follows the established standard (Yan et al., 2020; Wu et al., 2021) of conducting a search for the maximum validation accuracy within a fixed number of queries and reporting the corresponding test accuracy, both as a mean over 10 runs.

For NAS-Bench-101, we compare our approach with Arch2Vec (Yan et al., 2020), NAO (Luo et al., 2018), BANANAS (White et al., 2021a), Bayesian Optimisation (Snoek et al., 2015), Local Search (White et al., 2021b), Random Search (Li & Talwalkar, 2020), Regularised Evolution (Real et al., 2019), WeakNAS (Wu et al., 2021) and AG-Net (Lukasik et al., 2022). For NAS-Bench-201, our approach is evaluated against SGNAS (Huang & Chu, 2021), GANAS (Rezaei et al., 2021), BANANAS, Bayesian Optimisation, Random Search, AG-Net, TNAS (Shala et al., 2023) and MetaD2A (Lee et al., 2021). The corresponding results are reported in Tables 1 and 2.

**Surrogate Benchmarks** Next, we evaluate our method on surrogate benchmarks. Surrogate benchmarks operate on significantly larger search spaces like DARTS (Liu et al., 2019) or NAS-Bench-NLP (Klyuchnikov et al., 2020) and therefore use a simple surrogate predictor to estimate the ground truth accuracy. We perform our experiments on two surrogate benchmarks, the NAS-Bench-301 (Siems et al., 2021) (trained on CIFAR-10 (Krizhevsky et al., 2009)) on DARTS search space and NAS-Bench-NLP. We report the maximum validation accuracy as a mean over 10 runs, along with the number of queries and compare our method to the prior work as with NAS-Bench-101. The results are presented in Table 3.

Table 1: Comparison of results on NAS-Bench-101. 'Val' represents the maximum validation accuracy and 'Test' represents the corresponding test accuracy, both as a mean over 10 runs. Queries are the number of retrieval attempts for accuracy from the benchmark.

| Methods | Val(%) | Test(%) | Queries ↓ |
|---|---|---|---|
| Optimum | 95.06 | 94.32 | |
| Arch2vec + RL | - | 94.10 | 400 |
| Arch2vec + BO | - | 94.05 | 400 |
| NAO † | 94.66 | 93.49 | 192 |
| BANANAS † | 94.73 | 94.09 | 192 |
| Local Search † | 94.57 | 93.97 | 192 |
| Random Search † | 94.31 | 93.61 | 192 |
| Bayesian Optimisation † | 94.57 | 93.96 | 192 |
| WeakNAS | - | 94.18 | 200 |
| Regularised Evolution † | 94.47 | 93.89 | 192 |
| AG-Net | 94.90 | 94.18 | 192 |
| **DiNAS (ours)** | **94.98** | **94.27** | **150** |

**Hardware Aware Benchmark** Our next evaluation is on the Hardware Aware Benchmark (HW-NAS-Bench) (Li et al., 2021). HW-NAS-Bench provides hardware information (e.g. latency) along with the accuracy for multiple edge devices. We follow the standard protocol (Lukasik et al., 2022) and report the accuracy of best found architectures for ImageNet classification task given the latency constraint (in milliseconds) as a mean over 10 runs along with the number of queries. We also report the feasibility, which indicates the percentage of generated architectures following the given latency constraint. We compare our approach to Random Search and AG-Net as strong baselines for multiple devices each in multiple latency constraints. The results are available in Table 4.

**Experiments on ImageNet** Lastly, we conduct experiments on the large-scale image classification task ImageNet (Deng et al., 2009), following the protocol from Liu et al. (2019); Chen et al. (2021a). This

[2]The detailed evaluation protocol for each benchmark can be found in Appendix A.3.

involves training and evaluating the best generated architecture from NASBench301 (trained on CIFAR10 image classification task) on the ImageNet dataset. We report the top-1 and top-5 errors along with the number of queries and search time (in GPU hours), comparing our method to several robust baselines (e.g. DARTS, TENAS (Chen et al., 2021a), NASNET-A (Zoph et al., 2018) and AG-Net. To ensure a fair comparison, we report the results of methods with search on CIFAR-10 and evaluation on ImageNet. We summarise the results in Table 5.

**Implementation** We empirically found that in our task, $d = 2$ for the accuracy metric has a slightly superior performance over other values of $d$ (see ablation study in Appendix Sec. 5.3.3) and thus, we discretise the accuracy into two classes. One class includes $> f_{th}$ percentile of accuracy values, while the remaining values belong to the other class. Using higher values of $f$ for accuracy generates better-performing architectures, but they also lead to class imbalance, thereby reducing the model performance. We address this issue by modifying $f$ depending on the data availability of the specific benchmark. For generating high performing samples, the model is conditioned to generate the samples belonging to $> f_{th}$ percentile class for accuracy during the sampling process. Specific values for each benchmark can be found in the Appendix Sec. A.3. For the metric of latency in the hardware-aware benchmark (Li et al., 2021), we wish to generate high-performing architectures lower than the given latency constraint. To achieve this, we discretize latency into two discrete classes- one below the constraint value and one above the constraint.

Table 2: Comparison of results on NAS-Bench-201 for different datasets. 'Val' represents the maximum validation accuracy and 'Test' represents the corresponding test accuracy, both as a mean over 10 runs. Queries/Gen. are the number of retrieval attempts for accuracy from the benchmark or number of architecture generations.

| Methods | CIFAR-10 | | CIFAR-100 | | ImageNet16-120 | | Queries/Gen. ↓ |
|---|---|---|---|---|---|---|---|
| | Val(%) | Test (%) | Val(%) | Test(%) | Val(%) | Test(%) | |
| **Optimum*** | 91.61 | 94.37 | 73.49 | 73.51 | 46.77 | 47.31 | - |
| SGNAS | 90.18 | 93.53 | 70.28 | 70.31 | 44.65 | 44.98 | - |
| BANANAS † | 91.56 | 94.30 | 73.49* | 73.50 | 46.65 | 46.51 | 192 |
| Bayesian Opt.† | 91.54 | 94.22 | 73.26 | 73.22 | 46.43 | 46.40 | 192 |
| Random Search† | 91.12 | 93.89 | 72.08 | 72.07 | 45.97 | 45.98 | 192 |
| GANAS | - | 94.34 | - | 73.28 | - | **46.80** | 444 |
| AG-Net | 91.60 | 94.37* | 73.49* | 73.51* | 46.64 | 46.43 | 192 |
| TNAS | - | 94.37* | - | 73.51* | - | - | - |
| MetaD2A | - | 94.37* | - | 73.34 | - | - | 500 |
| **DiNAS (ours)** | **91.61*** | **94.37*** | **73.49*** | **73.51*** | **46.66** | 45.41 | 192 |

### 5.2 Discussion of Results

**Tabular Benchmarks** Tables 1 and 2 present empirical evidence of the superior performance of our proposed method in the context of tabular benchmarks. Across both tabular benchmarks, our approach con-

Table 3: Comparison of results on NAS-Bench-301 (left) and NAS-Bench-NLP (right). 'Val' represents the maximum validation accuracy as a mean over 10 runs. Queries are the number of retrieval attempts for accuracy from the benchmark.

| Methods | NAS-Bench-301 | | NAS-Bench-NLP | |
|---|---|---|---|---|
| | Val(%) | Queries↓ | Val(%) | Queries↓ |
| BANANAS† | 94.47 | 192 | 95.68 | 304 |
| Bayesian Opt.† | 94.71 | 192 | - | - |
| Random Search† | 94.31 | 192 | 95.64 | 304 |
| Regularised Evolution† | 94.75 | 192 | 95.66 | 304 |
| AG-Net ‡ | 94.79 | 192 | 95.95 | 304 |
| **DiNAS (ours)** | **94.92** | **100** | **96.06** | 304 |

Table 4: Comparison of results on HW-NAS-Bench. 'Val' represents the maximum validation accuracy as a mean over 10 runs and 'Feas' represents the feasibility considering generations of all the runs. Queries are the number of retrieval attempts for accuracy and latency from the benchmark.

| Device | Lat. (ms) | DiNAS | | AG-Net | | Random | Queries ↓ |
|---|---|---|---|---|---|---|---|
| | | Val(%) | Feas. (%)↑ | Val(%) | Feas. (%)↑ | Val(%) | |
| EdgeGPU | 2 | 39.44 | **92.60** | 39.70 | 29.00 | 37.20 | 200 |
| | 4 | **43.91** | **93.20** | 42.80 | 29.00 | 41.70 | 200 |
| | 6 | 45.03 | **66.35** | 45.30 | 64.00 | 44.90 | 200 |
| Raspi4 | 2 | **34.67** | **92.80** | 34.60 | 28.00 | 33.90 | 200 |
| | 4 | **43.25** | **77.80** | 42.00 | 47.00 | 41.90 | 200 |
| | 6 | **44.72** | **57.70** | 44.00 | 56.00 | 43.20 | 200 |
| EdgeTPU | 1 | 45.31 | 48.37 | 46.40 | 74.00 | 45.40 | 200 |
| Pixel3 | 2 | 40.01 | **97.30** | 40.90 | 48.00 | 38.80 | 200 |
| | 4 | 44.74 | **82.50** | 45.30 | 69.00 | 43.8 | 200 |
| | 6 | **45.95** | **78.50** | 45.70 | 77.00 | 45.1 | 200 |
| Eyeris | 1 | **44.67** | **78.12** | 44.50 | 49.00 | 43.30 | 200 |
| FPGA | 1 | **44.53** | **91.65** | 43.30 | 65.00 | 42.90 | 200 |

sistently outperforms the SOTA or converges to optimal validation accuracy. Notably, for NAS-Bench-101, our method concurrently reduces query count by 25%, demonstrating its effectiveness. GANAS exhibits a slightly better test accuracy on ImageNet in NAS-Bench-201 experiments, which can be explained by the fact that our method searches for the best architecture in terms of validation accuracy and the best validation accuracy does not necessarily imply best test accuracy.

**Surrogate Benchmarks** Furthermore, the results in Table 3 demonstrate that DiNAS excels in surrogate benchmarks as well. Our method achieves the SOTA in nearly 50% reduction in queries for NAS-Bench-301 and the same query count in NAS-Bench-NLP, surpassing the performance of previous methods such as Random Search, Bayesian Optimisation, and AG-Net. The results from NAS-Bench-NLP experiment also prove that our approach is not only effective in image classification tasks but also in NLP tasks, proving our approach to be task-independent.

**Hardware-Aware benchmark** From Table 4, we can observe that our approach outperforms Random Search in most cases and surpassing AG-Net in over half of them. Additionally, our method excels in feasibility across diverse devices and latency constraints while using the same number of queries compared to AG-Net, proving that our multi-conditioned guidance was indeed able to replicate the behaviour of multiple independent predictors (for accuracy and latency) using a single set of hyperparameters.

**ImageNet** Lastly, we can observe from Table 5 that our approach demonstrates competitive performance with low top-1 and top-5 error rates on ImageNet, outperforming robust baselines such as DARTS, NASNET-A and TENAS while requiring almost a third of the queries and significantly less search time (400x less compared to AG-Net, and three orders of magnitude less compared to DARTS). However, the CIFAR-10 generations from AG-Net are slightly better performing on ImageNet than our method, with 0.7% difference in top-1 error rate and 0.2% difference in top-5 error rate. This experiment also highlights that the generated architectures from our method possess generalisation capabilities across different datasets.

### 5.3 Ablation studies

#### 5.3.1 Novelty and Uniqueness Analysis

We conduct an analysis of novelty and uniqueness for the generated architectures. We start by generating 2000 architectures based on our proposed method. To assess novelty, we calculate the percentage of generated samples absent in the training data whereas to assess uniqueness, we calculate the ratio of architectures

‡Note that Lukasik et al. (2022) refers to validation accuracy of NAS-Bench-NLP as validation perplexity

†Results taken from Lukasik et al. (2022)

Table 5: Comparison of results for top-1, top-5 errors and search time in GPU hours on ImageNet. $T_1$ represents the one-time cost incurred in training DiNAS and $T_2$ refers to the training cost of AG-Net.

| Methods | Top-1 ↓ | Top-5 ↓ | Queries↓ | Search Time (GPU hrs)↓ |
|---|---|---|---|---|
| NASNET-A | 26.0 | 8.4 | 20000 | 48000 |
| DARTS | 26.7 | 8.7 | - | 96 |
| TENAS | 26.2 | 8.3 | - | 1.2 |
| AG-Net | **24.1** | **7.2** | 304 | 0.48+($T_2 = 10$) |
| **DiNAS (ours)** | 24.8 | 7.4 | **100** | **0.001+**$T_1$ |

present just once in the generations to the total number of generations. Given the enormous size of DARTS and NAS-Bench-NLP search spaces, we consider NAS-Bench-301 and NAS-Bench-NLP for our analysis. Furthermore, to examine the efficiency of our method, we record and report the training times for our method (for 100 epochs), along with the sampling times per architecture using a single NVIDIA A6000 GPU on five different benchmarks. We can observe the results of our ablation studies in Table 6. Note that for benchmarks involving multiple cases (e.g. HWNAS and NAS-Bench-201), we take the mean of the training times for all cases.

We observe from the results of the novelty analysis that in both the cases all the generated samples are novel and most of them are unique, proving that our method is not just selecting the best-performing architectures from the training set. Moreover, we can observe that the sampling rates of each benchmark are in milliseconds, proving the rapid generation capabilities of our method.

Table 6: Ablation study on novelty analysis (left) and efficiency analysis (right). Note that 'Nov' represents the novelty ratio and 'Uni' represents the uniqueness ratio. The training time is reported in hours and sampling time in seconds.

| Benchmark | Nov.(%) ↑ | Uni.(%) ↑ |
|---|---|---|
| NAS-Bench-301 | 100 | 97.37 |
| NAS-Bench-NLP | 100 | 97.57 |

| Benchmark | Train (hrs)↓ | Sample (sec)↓ |
|---|---|---|
| NB101 | 0.96 | 0.09 |
| NB201 | 0.25 | 0.08 |
| NB301 (Normal) | 8.3 | 0.14 |
| NB301 (Reduced) | 8.3 | 0.14 |
| NBNLP | 0.95 | 0.15 |
| HWNAS | 1.5 | 0.08 |

#### 5.3.2 Required number of training samples

This section analyses the performance of our method on DARTS search space (Liu et al., 2019) when trained on different number of samples. In particular, we consider training in three scenarios- on 100,000 samples, 10,000 samples and 1,000 samples, for which, we randomly sample the given number of training samples from DARTS search space and follow the same training protocol as our experiments on NAS-Bench-301. Upon training in each case, 192 architectures are generated for a total of 10 runs and the maximum validation accuracy is reported as a mean over these runs. In addition, we calculate and report the novelty and uniqueness of the generated architectures, calculated using the same methodology as Section 5.3.1 considering generations from all the 10 runs. Finally, we report the number of queries. The results for this ablation study are reported in Table 7.

We observe from Table 7 that our method learns the architectural representation and finds the high-performing architectures with one tenth of the number of training samples as well. However, the mean maximum validation accuracy drops when we further reduce the training samples to 1000. We found out that the reason for this decline was the unavailability of any valid generations in one of the runs. This resulted in the maximum validation accuracy for that run to be 0, which influenced the mean. From this, we can conclude that the data capture capabilities of our model are compromised when training on a very small number of samples. Moreover, we observe that the novelty and uniqueness ratios do not suffer at all when reducing the training data availability.

Table 7: Peformance on NAS-Bench-301 (Siems et al., 2021) with different number of samples. The Val. acc represents the maximum validation accuracy, reported as a mean over 10 runs, whereas novelty and uniqueness are calculated considering the generations from all the runs. Queries are the number of retrieval attempts for accuracy from the benchmark.

| Training Samples | Val acc.(%) ↑ | Novelty(%) ↑ | Uniqueness(%) ↑ | Queries |
|---|---|---|---|---|
| 100,000 | 94.92 | 100 | 97.37 | 192 |
| 10,000 | 94.89 | 100 | 100 | 192 |
| 1,000 | 85.29 | 100 | 100 | 192 |

### 5.3.3 Number of classes for guidance

The goal of this ablation study is to analyse the effect of the number of classes $d$ for accuracy present in the training data on our approach. We train and evaluate our method on NAS-Bench-201 (Dong & Yang, 2020) for the task of CIFAR-10 image classification involving four different cases: $d = \{2, 3, 4, 5\}$. We use the same training and evaluation protocol as experiments in Table 2. The split of the data into classes, denoted as $s_T$, is performed depending on specific percentiles $f = (f_1, f_2, \ldots, f_{d-1})$ of the accuracy. For instance, for two classes, $f = 95$ and the split $s_T = [95_{th} - 100_{th}, 0_{th} - 95_{th}]$, while for three classes, $f = [80, 95]$ and $s_T = [95_{th} - 100_{th}, 80_{th} - 95_{th}, 0_{th} - 80_{th}]$. In all the cases, we generate architectures belonging to the class of >95th percentile. The choice of $f$ is empirical as it does not affect the samples in the class we want to condition our model on (i.e. $> 95_{th}$ percentile). We report the maximum validation accuracy and the corresponding test accuracy, both as a mean over 10 runs, along with the number of queries. Moreover, we report the percentiles $f$ used for splitting. The results for this study are reported in Table 8.

Table 8: Comparison of results on NAS-Bench-201 for CIFAR-10 when using different number of classes. Here, $f$ represents the percentiles used for splitting the data into classes, 'Val' and 'Test' represent the maximum validation accuracy and the corresponding test accuracy, both represented as a mean over 10 runs. Queries are the number of retrieval attempts for accuracy from the benchmark.

| Number of classes ($d$) | $f$ | Val(%)↑ | Test(%)↑ | Queries |
|---|---|---|---|---|
| 2 | [95] | **91.61** | **94.37** | 192 |
| 3 | [80, 95] | 91.60 | 94.00 | 192 |
| 4 | [50, 80, 95] | 91.52 | 93.79 | 192 |
| 5 | [30, 50, 80, 95] | 91.57 | 93.89 | 192 |

We observe from Table 8 that discretising the accuracy into two classes results in a slightly superior performance over other values of $d$. However, the differences are marginal.

## 6 Conclusion

We presented a generative method to facilitate the search process for neural architectures. Our approach uses a conditional graph diffusion model to rapidly generate novel, unique and high-performing neural network architectures. In this context, we first formulated the classifier-free guidance for graph diffusion models and then proposed a multi-conditioned classifier-free guidance for diffusion models. Unlike the related work, our method does not require an external surrogate predictor and is thus differentiable. In the experiments, we demonstrated state-of-the-art performance in tabular, surrogate and hardware-aware evaluations by considering six standard benchmarks. Furthermore, we have shown the search efficiency of our approach compared to the previous work using experiments on ImageNet. We observed that our method samples architectures two orders of magnitude faster than other generative NAS approaches and at least three orders of magnitude faster than classic approaches.

## References


Sohyun An, Hayeon Lee, Jaehyeong Jo, Seanie Lee, and Sung Ju Hwang. Diffusionnag: Task-guided neural architecture generation with diffusion models. *arXiv preprint arXiv:2305.16943*, 2023.

Jacob Austin, Daniel D. Johnson, Jonathan Ho, Daniel Tarlow, and Rianne van den Berg. Structured denoising diffusion models in discrete state-spaces. In M. Ranzato, A. Beygelzimer, Y. Dauphin, P.S. Liang, and J. Wortman Vaughan (eds.), *Advances in Neural Information Processing Systems*, volume 34, pp. 17981–17993. Curran Associates, Inc., 2021. URL https://proceedings.neurips.cc/paper_files/paper/2021/file/958c530554f78bcd8e97125b70e6973d-Paper.pdf.

Bowen Baker, Otkrist Gupta, Nikhil Naik, and Ramesh Raskar. Designing neural network architectures using reinforcement learning. In *International Conference on Learning Representations*, 2017. URL https://openreview.net/forum?id=S1c2cvqee.

Gabriel Bender, Pieter-Jan Kindermans, Barret Zoph, Vijay Vasudevan, and Quoc Le. Understanding and simplifying one-shot architecture search. In Jennifer Dy and Andreas Krause (eds.), *Proceedings of the 35th International Conference on Machine Learning*, volume 80 of *Proceedings of Machine Learning Research*, pp. 550–559. PMLR, 10–15 Jul 2018. URL https://proceedings.mlr.press/v80/bender18a.html.

Andrew Brock, Theo Lim, J.M. Ritchie, and Nick Weston. SMASH: One-shot model architecture search through hypernetworks. In *International Conference on Learning Representations*, 2018. URL https://openreview.net/forum?id=rydeCEhs-.

Tianqi Chen and Carlos Guestrin. Xgboost: A scalable tree boosting system. In *Proceedings of the 22nd acm sigkdd international conference on knowledge discovery and data mining*, pp. 785–794, 2016.

Wuyang Chen. Darts evaluation: Train from scratch for architectures from darts space, 2022. URL https://github.com/chenwydj/DARTS_evaluation.

Wuyang Chen, Xinyu Gong, and Zhangyang Wang. Neural architecture search on imagenet in four gpu hours: A theoretically inspired perspective. In *International Conference on Learning Representations*, 2021a.

Xiangning Chen, Ruochen Wang, Minhao Cheng, Xiaocheng Tang, and Cho-Jui Hsieh. Drnas: Dirichlet neural architecture search. In *International Conference on Learning Representations*, 2021b. URL https://openreview.net/forum?id=9FWas6YbmB3.

Xiangxiang Chu, Bo Zhang, and Ruijun Xu. Multi-objective reinforced evolution in mobile neural architecture search. In *European Conference on Computer Vision*, pp. 99–113. Springer, 2020.

Jia Deng, Wei Dong, Richard Socher, Li-Jia Li, Kai Li, and Li Fei-Fei. Imagenet: A large-scale hierarchical image database. In *2009 IEEE Conference on Computer Vision and Pattern Recognition*, pp. 248–255, 2009. doi: 10.1109/CVPR.2009.5206848.

Prafulla Dhariwal and Alexander Nichol. Diffusion models beat gans on image synthesis. *Advances in Neural Information Processing Systems*, 34:8780–8794, 2021.

Xuanyi Dong and Yi Yang. Nas-bench-201: Extending the scope of reproducible neural architecture search. In *International Conference on Learning Representations (ICLR)*, 2020.

Vijay Prakash Dwivedi and Xavier Bresson. A generalization of transformer networks to graphs. *AAAI Workshop on Deep Learning on Graphs: Methods and Applications*, 2021.

Thomas Elsken, Jan Hendrik Metzen, and Frank Hutter. Neural architecture search: A survey. *Journal of Machine Learning Research*, 20(55):1–21, 2019. URL http://jmlr.org/papers/v20/18-598.html.

Jörg K.H. Franke, Gregor Koehler, André Biedenkapp, and Frank Hutter. Sample-efficient automated deep reinforcement learning. In *International Conference on Learning Representations*, 2021. URL https://openreview.net/forum?id=hSjxQ3B7GWq.

Nico Giambi and Giuseppe Lisanti. Conditioning diffusion models via attributes and semantic masks for face generation. *arXiv preprint arXiv:2306.00914*, 2023.

Jonathan Ho and Tim Salimans. Classifier-free diffusion guidance. In *NeurIPS 2021 Workshop on Deep Generative Models and Downstream Applications*, 2021.

Jonathan Ho, Ajay Jain, and Pieter Abbeel. Denoising diffusion probabilistic models. *Advances in neural information processing systems*, 33:6840–6851, 2020.

Emiel Hoogeboom, Vıctor Garcia Satorras, Clément Vignac, and Max Welling. Equivariant diffusion for molecule generation in 3d. In *International conference on machine learning*, pp. 8867–8887. PMLR, 2022.

Andrew Howard, Mark Sandler, Grace Chu, Liang-Chieh Chen, Bo Chen, Mingxing Tan, Weijun Wang, Yukun Zhu, Ruoming Pang, Vijay Vasudevan, et al. Searching for mobilenetv3. In *Proceedings of the IEEE/CVF international conference on computer vision*, pp. 1314–1324, 2019.

Sian-Yao Huang and Wei-Ta Chu. Searching by generating: Flexible and efficient one-shot nas with architecture generator. In *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition*, pp. 983–992, 2021.

Haifeng Jin, Qingquan Song, and Xia Hu. Auto-keras: An efficient neural architecture search system. In *Proceedings of the 25th ACM SIGKDD international conference on knowledge discovery & data mining*, pp. 1946–1956, 2019.

Diederik P Kingma and Max Welling. Auto-encoding variational bayes. *arXiv preprint arXiv:1312.6114*, 2013.

Nikita Klyuchnikov, Ilya Trofimov, E. Artemova, Mikhail Salnikov, Maxim Fedorov, and Evgeny Burnaev. Nas-bench-nlp: Neural architecture search benchmark for natural language processing. *IEEE Access*, PP: 1–1, 2020.

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

Hayeon Lee, Eunyoung Hyung, and Sung Ju Hwang. Rapid neural architecture search by learning to generate graphs from datasets. In *International Conference on Learning Representations*, 2021.

Chaojian Li, Zhongzhi Yu, Yonggan Fu, Yongan Zhang, Yang Zhao, Haoran You, Qixuan Yu, Yue Wang, Cong Hao, and Yingyan Lin. {HW}-{nas}-bench: Hardware-aware neural architecture search benchmark. In *International Conference on Learning Representations*, 2021.

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

Hanxiao Liu, Karen Simonyan, and Yiming Yang. DARTS: Differentiable architecture search. In *International Conference on Learning Representations*, 2019.

Ilya Loshchilov and Frank Hutter. Decoupled weight decay regularization. *arXiv preprint arXiv:1711.05101*, 2017.

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)*, pp. 1–8. IEEE, 2021.

Jovita Lukasik, Steffen Jung, and Margret Keuper. Learning where to look–generative nas is surprisingly efficient. In *European Conference on Computer Vision*, pp. 257–273. Springer, 2022.

Renqian Luo, Fei Tian, Tao Qin, Enhong Chen, and Tie-Yan Liu. Neural architecture optimization. *Advances in neural information processing systems*, 31, 2018.

Xiangzhong Luo, Di Liu, Hao Kong, Shuo Huai, Hui Chen, and Weichen Liu. Surgenas: a comprehensive surgery on hardware-aware differentiable neural architecture search. *IEEE Transactions on Computers*, 72(4):1081–1094, 2022.

Mitchell P. Marcus, Beatrice Santorini, and Mary Ann Marcinkiewicz. Building a large annotated corpus of English: The Penn Treebank. *Computational Linguistics*, 19(2):313–330, 1993. URL https://aclanthology.org/J93-2004.

Hieu Pham, Melody Guan, Barret Zoph, Quoc Le, and Jeff Dean. Efficient neural architecture search via parameters sharing. In *International conference on machine learning*, pp. 4095–4104. PMLR, 2018.

Esteban Real, Sherry Moore, Andrew Selle, Saurabh Saxena, Yutaka Leon Suematsu, Jie Tan, Quoc V. Le, and Alexey Kurakin. Large-scale evolution of image classifiers. In Doina Precup and Yee Whye Teh (eds.), *Proceedings of the 34th International Conference on Machine Learning*, volume 70 of *Proceedings of Machine Learning Research*, pp. 2902–2911. PMLR, 06–11 Aug 2017.

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

Seyed Saeed Changiz Rezaei, Fred X Han, Di Niu, Mohammad Salameh, Keith Mills, Shuo Lian, Wei Lu, and Shangling Jui. Generative adversarial neural architecture search. *arXiv preprint arXiv:2105.09356*, 2021.

Robin Rombach, Andreas Blattmann, Dominik Lorenz, Patrick Esser, and Björn Ommer. High-resolution image synthesis with latent diffusion models. In *Proceedings of the IEEE/CVF conference on computer vision and pattern recognition*, pp. 10684–10695, 2022.

Chitwan Saharia, William Chan, Saurabh Saxena, Lala Li, Jay Whang, Emily L Denton, Kamyar Ghasemipour, Raphael Gontijo Lopes, Burcu Karagol Ayan, Tim Salimans, et al. Photorealistic text-to-image diffusion models with deep language understanding. *Advances in Neural Information Processing Systems*, 35:36479–36494, 2022.

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. URL https://openreview.net/forum?id=paGvsrl4Ntr.

Julien Niklas Siems, Lucas Zimmer, Arber Zela, Jovita Lukasik, Margret Keuper, and Frank Hutter. {NAS}-bench-301 and the case for surrogate benchmarks for neural architecture search, 2021.

Jasper Snoek, Oren Rippel, Kevin Swersky, Ryan Kiros, Nadathur Satish, Narayanan Sundaram, Mostofa Patwary, Mr Prabhat, and Ryan Adams. Scalable bayesian optimization using deep neural networks. In *International conference on machine learning*, pp. 2171–2180. PMLR, 2015.

Jascha Sohl-Dickstein, Eric Weiss, Niru Maheswaranathan, and Surya Ganguli. Deep unsupervised learning using nonequilibrium thermodynamics. In *International conference on machine learning*, pp. 2256–2265. PMLR, 2015.

Yuan Tian, Qin Wang, Zhiwu Huang, Wen Li, Dengxin Dai, Minghao Yang, Jun Wang, and Olga Fink. Off-policy reinforcement learning for efficient and effective gan architecture search. In *Computer Vision–ECCV 2020: 16th European Conference, Glasgow, UK, August 23–28, 2020, Proceedings, Part VII 16*, pp. 175–192. Springer, 2020.

Ashish Vaswani, Noam Shazeer, Niki Parmar, Jakob Uszkoreit, Llion Jones, Aidan N Gomez, Ł ukasz Kaiser, and Illia Polosukhin. Attention is all you need. In I. Guyon, U. Von Luxburg, S. Bengio, H. Wallach, R. Fergus, S. Vishwanathan, and R. Garnett (eds.), *Advances in Neural Information Processing Systems*, volume 30. Curran Associates, Inc., 2017a. URL https://proceedings.neurips.cc/paper_files/paper/2017/file/3f5ee243547dee91fbd053c1c4a845aa-Paper.pdf.

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

Clement Vignac, Igor Krawczuk, Antoine Siraudin, Bohan Wang, Volkan Cevher, and Pascal Frossard. Digress: Discrete denoising diffusion for graph generation. In *The Eleventh International Conference on Learning Representations*, 2023.

Junxiang Wang, Junji Jiang, and Liang Zhao. An invertible graph diffusion neural network for source localization. In *Proceedings of the ACM Web Conference 2022*, WWW '22, pp. 1058–1069, New York, NY, USA, 2022. Association for Computing Machinery. ISBN 9781450390965. doi: 10.1145/3485447.3512155. URL `https://doi.org/10.1145/3485447.3512155`.

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

Colin White, Sam Nolen, and Yash Savani. Exploring the loss landscape in neural architecture search. In *Uncertainty in Artificial Intelligence*, pp. 654–664. PMLR, 2021b.

Bichen Wu, Xiaoliang Dai, Peizhao Zhang, Yanghan Wang, Fei Sun, Yiming Wu, Yuandong Tian, Peter Vajda, Yangqing Jia, and Kurt Keutzer. Fbnet: Hardware-aware efficient convnet design via differentiable neural architecture search. In *Proceedings of the IEEE/CVF conference on computer vision and pattern recognition*, pp. 10734–10742, 2019.

Junru Wu, Xiyang Dai, Dongdong Chen, Yinpeng Chen, Mengchen Liu, Ye Yu, Zhangyang Wang, Zicheng Liu, Mei Chen, and Lu Yuan. Stronger nas with weaker predictors. *Advances in Neural Information Processing Systems*, 34:28904–28918, 2021.

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.

Shen Yan, Colin White, Yash Savani, and Frank Hutter. Nas-bench-x11 and the power of learning curves. *Advances in Neural Information Processing Systems*, 34:22534–22549, 2021.

Yibo Yang, Hongyang Li, Shan You, Fei Wang, Chen Qian, and Zhouchen Lin. Ista-nas: Efficient and consistent neural architecture search by sparse coding. *Advances in Neural Information Processing Systems*, 33:10503–10513, 2020.

Chris Ying, Aaron Klein, Eric Christiansen, Esteban Real, Kevin Murphy, and Frank Hutter. NAS-bench-101: Towards reproducible neural architecture search. In Kamalika Chaudhuri and Ruslan Salakhutdinov (eds.), *Proceedings of the 36th International Conference on Machine Learning*, volume 97 of *Proceedings of Machine Learning Research*, pp. 7105–7114, Long Beach, California, USA, 09–15 Jun 2019. PMLR.

Barret Zoph and Quoc Le. Neural architecture search with reinforcement learning. In *International Conference on Learning Representations*, 2017. URL `https://openreview.net/forum?id=r1Ue8Hcxg`.

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*, pp. 8697–8710, 2018.

# A Appendix

## A.1 Proofs and Derivations

**Derivation 1:** Let $q$ be the unconditional Markovian noise model and $\hat{q}$ be the conditional noising process similar to $q$. We define our aim as decomposing $\hat{q}(\mathbf{G}^{t-1}|\mathbf{G}^t, y_1, y_2, ..., y_k)$ and then deriving the score function

for the multi-conditioned diffusion process. We start by expanding the term:

$$\hat{q}(\mathbf{G}^{t-1}|\mathbf{G}^t, y_1, y_2, \ldots, y_k) = \frac{\hat{q}(\mathbf{G}^{t-1}, \mathbf{G}^t, y_1, y_2, \ldots, y_k)}{\hat{q}(\mathbf{G}^t, y_1, y_2, \ldots, y_k)} \tag{11}$$

$$= \frac{\hat{q}(\mathbf{G}^{t-1}, \mathbf{G}^t, y_1, y_2, \ldots, y_k)}{\hat{q}(y_1, y_2, \ldots, y_k|\mathbf{G^t})\hat{q}(\mathbf{G^t})} \tag{12}$$

$$= \frac{\hat{q}(y_1, \ldots, y_k|\mathbf{G}^{t-1}, \mathbf{G}^t)\hat{q}(\mathbf{G}^{t-1}, \mathbf{G}^t)}{\hat{q}(y_1, y_2, \ldots, y_k|\mathbf{G^t})\hat{q}(\mathbf{G^t})} \tag{13}$$

$$= \frac{\hat{q}(y_1, \ldots, y_k|\mathbf{G}^{t-1}, \mathbf{G}^t)\hat{q}(\mathbf{G}^{t-1}|\mathbf{G}^t)\hat{q}(\mathbf{G}^t)}{\hat{q}(y_1, y_2, \ldots, y_k|\mathbf{G}^t)\hat{q}(\mathbf{G}^t)} \tag{14}$$

$$= \frac{\hat{q}(y_1, \ldots, y_k|\mathbf{G}^{t-1}, \mathbf{G}^t)\hat{q}(\mathbf{G}^{t-1}|\mathbf{G}^t)}{\hat{q}(y_1, y_2, \ldots, y_k|\mathbf{G^t})} \tag{15}$$

Dhariwal & Nichol (2021) prove that the classification term (in our case $\hat{q}(y_1, \ldots, y_k|\mathbf{G}^{t-1}, \mathbf{G}^t)$) does not depend on the noisier version of G (i.e. $\mathbf{G^t}$) and can be rewritten as $\hat{q}(y_1, \ldots, y_k|\mathbf{G}^{t-1})$. Furthermore, they also show that $\hat{q}$ behaves the same as $q$ when not conditioned on the classification targets $y_1, \ldots, y_k$. We use these findings to further simplify Eq. 15 to:

$$\hat{q}(\mathbf{G}^{t-1}|\mathbf{G}^t, y_1, y_2, \ldots, y_k) = \frac{\hat{q}(y_1, \ldots, y_k|\mathbf{G}^{t-1})q(\mathbf{G}^{t-1}|\mathbf{G}^t)}{\hat{q}(y_1, y_2, \ldots, y_k|\mathbf{G^t})} \tag{16}$$

We assume that the generative model $p_\theta(\mathbf{G}^{t-1}|\mathbf{G}^t)$ approximates $q(\mathbf{G}^{t-1}|\mathbf{G}^t)$ and the classifier $p_\psi(y_1, \ldots, y_k|\mathbf{G}^{t-1})$ approximates $\hat{q}(y_1, \ldots, y_k|\mathbf{G}^{t-1})$. By substituting the distributions with their approximations, taking the logarithm and calculating the gradients w.r.t. $\mathbf{G}^{t-1}$, we obtain the following score function:

$$\nabla_{\mathbf{G}^{t-1}} \log p_{\theta,\psi}(\mathbf{G}^{t-1}|\mathbf{G}^t, y_1, ..., y_k) = \nabla_{\mathbf{G}^{t-1}} \log p_\theta(\mathbf{G}^{t-1}|\mathbf{G}^t) + \nabla_{\mathbf{G}^{t-1}} \log p_\psi(y_1, ..., y_k|\mathbf{G}^{t-1}), \tag{17}$$

where $\mathbf{G^t}$ and $\mathbf{G}^{t-1}$ represent the DAG $\mathbf{G}$ at time step $t$ and $t-1$ respectively, $\psi$ are the classifier parameters and $\theta$ are the generative model parameters. Similar to the standard classifier-based guidance (Dhariwal & Nichol, 2021), we multiply the conditioning term by a factor of $\gamma$ . Thus, we can express the reverse denoising process as a weighted sum of the unconditional score function $\nabla_{\mathbf{G}^{t-1}} \log p_\theta(\mathbf{G}^{t-1}|\mathbf{G}^t)$ and the conditioning term $\nabla_{\mathbf{G}^{t-1}} \log p_\psi(y_1, \ldots, y_k|\mathbf{G}^{t-1})$, given by:

$$\nabla_{\mathbf{G}^{t-1}} \log p_{\theta_\gamma,\psi}(\mathbf{G}^{t-1}|\mathbf{G}^t, y_1, ..., y_k) = \nabla_{\mathbf{G}^{t-1}} \log p_\theta(\mathbf{G}^{t-1}|\mathbf{G}^t) + \gamma\nabla_{\mathbf{G}^{t-1}} \log p_\psi(y_1, \ldots, y_k|\mathbf{G}^{t-1}), \tag{18}$$

where $\gamma$ is the guidance scale. Then, by substituting the conditioning term $\nabla_{\mathbf{G}^{t-1}} \log p_\psi(y_1, \ldots, y_k|\mathbf{G}^{t-1})$ from Eq. 17 and removing the classifier parameters $\psi$, we obtain:

$$\begin{aligned}\nabla_{\mathbf{G}^{t-1}} \log p_{\theta_\gamma}(\mathbf{G}^{t-1}|\mathbf{G}^t, y_1, ..., y_k) = (1-\gamma)\nabla_{\mathbf{G}^{t-1}} \log p_\theta(\mathbf{G}^{t-1}|\mathbf{G}^t) \\ + \gamma\nabla_{\mathbf{G}^{t-1}} \log p_\theta(\mathbf{G}^{t-1}|\mathbf{G}^t, y_1, ..., y_k).\end{aligned} \tag{19}$$

Hence, as we can observe from Equation 19, we have successfully derived the score function of multi-conditioned diffusion guidance.

### A.2 Implementation Details

Our proposed method involves training a Graph Transformer network, proposed by Dwivedi & Bresson (2021), for denoising. This network comprises of an input node/edge wise MLP layer, followed by 5 Graph Transformer layers and, node/edge wise MLP as the output layer. Each Graph Transformer layer consists

of three main parts: a self-attention module similar to the one found in the standard Transformer model (Vaswani et al., 2017a), a fully connected layer, and layer normalisation. All the models were trained for 100 epochs with a learning rate of 0.0002, batch-size of 16, weight decay of $10^{-12}$, and using AdamW optimiser (Loshchilov & Hutter, 2017) on a single NVIDIA A6000 GPU with 48GB VRAM. For noising, we use cosine noise schedule for $T = 500$ time-steps. The hyperparameters used in our method were derived from the code from Vignac et al. (2023).

For the evaluation on ImageNet, we employ the same training pipeline and code as AG-Net (Lukasik et al., 2022) and TENAS (Chen et al., 2021a), taken from Chen (2022). We train the best generated architecture in terms of validation accuracy from NAS-Bench-301 on ImageNet for 250 epochs. The initial learning rate is set to 0.5 with a cosine learning rate scheduler and the batch size is set to 1024. The ImageNet training is performed on 3 NVIDIA V100 GPUs parallelly in a distributed manner.

### A.3 Evaluation protocols

**NAS-Bench 101 and NAS-Bench 201** We generate a fixed number of architectures (equal to the respective number of queries) and query them on both the benchmarks to find the maximum validation accuracy and its corresponding test accuracy. This process is repeated 10 times to calculate the mean maximum validation accuracy and mean corresponding test accuracy. Note that we use $f = 99$ for NAS-Bench-101 experiments and $f = 95$ for NAS-Bench-201 experiments.

**NAS-Bench-301** To evaluate our approach on NAS-Bench-301, a random subset of 100,000 architectures is selected from the DARTS search space. As surrogate benchmarks do not provide accuracy, the accuracy of the selected architectures are calculated using a pre-trained surrogate predictor XG-Boost (Chen & Guestrin, 2016) provided with NAS-Bench-301. Next, the network is trained using normal cells from this dataset, producing 10 normal cells from $> f_{th}$ class. Next, this process is repeated to produce 10 reduction cells. The evaluation involves 100 queries, considering all possible combinations of the 10 generated normal and 10 reduction cells. For each query, the highest validation accuracy and its corresponding test accuracy are recorded. This entire process is iterated 10 times, yielding mean values for these recorded accuracies. We use $f = 99$ for this benchmark.

**NAS-Bench-NLP** Given the enormity of this search space, we employ NAS-Bench-X11 (Yan et al., 2021) as a surrogate predictor to obtain accuracy for these architectures, trained specifically on the Penn TreeBank dataset (Marcus et al., 1993). However, it should be noted that NAS-Bench-X11 is only capable of handling graphs with up to 12 nodes, which filters our dataset to include the total of 7,258 architectures. After training, we generate 304 architectures and estimate their accuracy using NAS-Bench-X11. The process is repeated 10 times. We set $f = 99$ for this benchmark.

**HW-NAS-Bench** For this evaluation, we consider 12 distinct cases for latency and device constraints. Upon each training, our method generates 200 architectures which are then queried on the benchmark. We adopt two conditions simultaneously, namely the accuracy should be in $> f_{th}(= 95)$ percentile class and the latency should satisfy the given constraint. We repeat the generation process for 10 runs and report the mean of the validation accuracy, along with the feasibility and number of queries.

**ImageNet** We start by generating 100 architectures through our approach trained on NAS-Bench-301. Then, we select the best architecture in terms of validation accuracy. Next, we train the network using the same training pipeline and code as TENAS (Chen et al., 2021a). Finally, we save the weights from the epoch where the top-1 and top-5 validation errors are minimum and report the top-1 and top-5 errors in Table 5.

### A.4 Benchmark Descriptions

#### A.4.1 NAS-Bench-101

NAS-Bench-101 (Ying et al., 2019) is a cell-based tabular benchmark, comprising a large collection of 423,624 distinct architectures represented as cells. These architectures are also mapped to their respective validation

and test accuracy metrics, evaluated on CIFAR-10 image classification task. In this benchmark, the cells are constrained to have a maximum of 7 nodes and 9 edges. Specifically, the first and last nodes within these cells serve as input and output nodes. Intermediate nodes within the cells can take on one of three possible operations: 1x1 convolution, 3x3 convolution, or 3x3 max-pooling. Furthermore, it is important to note that each convolutional operation is preceded by batch normalisation, followed by a Rectified Linear Unit (ReLU) activation function.

#### A.4.2 NAS-Bench-201

Another cell-based tabular benchmark is NAS-Bench-201 (Dong & Yang, 2020), which contains data for 15,625 architectures (cells) trained on 3 datasets- CIFAR-10, CIFAR-100 (Krizhevsky et al., 2009) and ImageNet16-120 (Deng et al., 2009). In contrast to NAS-Bench-101, each edge of a cell in NAS-Bench-201 is associated to an operation drawn from a predefined operation set $\mathcal{O} = \{$1x1 convolution, 3x3 convolution, 3x3 avg pooling, skip, zero$\}$. In our training and experiments on NAS-Bench-201, we convert the edge based representation to node-based representation, where each node is associated with an operation, similar to NAS-Bench-101. This conversion is in line with the conversion in Arch2Vec (Yan et al., 2020). Each cell comprises 4 nodes and 6 edges and the adjacency matrices are identical to one another. The existence of operations like zero and skip enforces the structural diversity in different architectures.

#### A.4.3 NAS-Bench-301

NAS-Bench-301 (Siems et al., 2021) is a surrogate benchmark that trains and evaluates several performance predictors on 60,000 sampled architectures from DARTS search space (Liu et al., 2019). These learned performance (surrogate) predictors are then able to predict the accuracy of architectures in DARTS search space (comprising $10^{18}$ architectures). The architectures in DARTS comprise of a normal cell and a reduction cell. Each cell has a maximum of 7 nodes and 12 edges with each edge associated with an operation drawn from the set $\mathcal{O} = \{$ 3x3 sep. conv., 3x3 dil. conv., 5x5 sep. conv, 3x3 average pooling, identity, zero $\}$. We utilise a pretrained XGBoost (Chen & Guestrin, 2016) provided by Siems et al. (2021) as the surrogate predictor for our experiments.

#### A.4.4 NAS-Bench-NLP

NAS-Bench-NLP is the first NAS benchmark designed for Natural Language Processing tasks (Klyuchnikov et al., 2020). While its search space is extremely large with the total of $10^{53}$ architectures, NAS-Bench-NLP provides 14,322 architectures trained on Penn TreeBank dataset (Marcus et al., 1993). Each cell in the search space has a maximum of 24 nodes, 3 hidden states and 3 linear input vectors. The nodes in each cell depict the operations drawn from the set $\mathcal{O} = \{$Linear, blending, product, sum, tanh, sigmoid, LeakyRELU $\}$.We utilise the surrogate predictor provided by NAS-Bench-X11 (**?**) for this benchmark.

#### A.4.5 HW-NAS-Bench

HW-NAS-Bench is a unique benchmark that provides hardware-specific details, including latency and energy cost, across various devices along with their respective accuracy. These devices encompass a diverse set of hardware platforms, including EdgeGPU, Raspi4, Pixel3, EdgeTPU, Eyeriss, Pixel3, and FPGA. Crucially, HW-NAS-Bench operates within two distinct search spaces: NAS-Bench-201 (Dong & Yang, 2020) and FB-Net (Wu et al., 2019). In our experiments, we utilise latency information as hardware constraint, within the context of the NAS-Bench-201 search space.