Towards Neural Architecture Search through Hierarchical Generative Modeling

Lichuan Xiang [1 *] **Łukasz Dudziak** [2 *] **Mohamed S. Abdelfattah** [3] **Abhinav Mehrotra** [2]
Nicholas D. Lane [4 5] **Hongkai Wen** [1 2]

Abstract

Neural Architecture Search (NAS) aims to automate deep neural network design across various applications, while a good search space design is core to NAS performance. A too-narrow search space may fail to cover diverse task requirements, whereas a too-broad one can escalate computational expenses and reduce efficiency. In this work, we aim to address this challenge by leaning on the recent advances in generative modelling – we propose a novel method that can navigate through an extremely large, general-purpose initial search space efficiently by training a two-level generative model hierarchy. The first level uses Conditional Continuous Normalizing Flow (CCNF) for micro-cell design, while the second employs a transformer-based sequence generator to craft macro architectures aligned with task needs and architectural constraints. To ensure computational feasibility, we pre-train the generative models in a task-agnostic manner using a metric space of graph and zero-cost (ZC) similarities between architectures. We show our approach can achieve state-of-the-art performance among other low-cost NAS methods across different tasks on CIFAR-10/100, ImageNet and NAS-Bench-360.

1. Introduction

Designing more accurate and efficient deep neural networks (DNNs) has been one of the primary areas of focus in machine learning research in recent years. However, developing a novel architecture is often accompanied by burdensome trial-and-error in an attempt to find an *optimal* DNN configuration. More recently, this process has been aided by a broad family of algorithms designed to automate optimizing a machine learning (ML) system.

Neural Architecture Search (NAS), as described by Zoph & Le (2017), is a process designed to automatically identify the most effective deep neural network configurations from a specified, discrete group of options, commonly known as a *search space*. NAS methodologies have demonstrated their efficacy and surpassed the performance of manually designed networks in many tasks. However, the practical application of NAS is constrained by various challenges, including search efficiency, robustness, and the ability to transfer findings effectively across diverse datasets or tasks, as noted by White et al. (2023)

In this paper, we focus on advancing NAS by tackling one of the challenges that remain somewhat overlooked, possibly due to its particular complexity – the challenge of being limited by the initial NAS search space, which is often rigid and hand-crafted. In practice, although carefully designed search spaces can improve NAS performance (Radosavovic et al., 2020), e.g. decreasing search time or improving the chance of finding good-performing architectures, it limits the applicability of NAS approaches to those tasks where strong/sensible search spaces are available. More importantly, by definition, the performance of any NAS approach is strictly associated with the quality of its search space – even the best searching algorithm can only produce results as good as the best candidate model from within the search space.

This inherent relationship between the statistical properties of a search space and the cost-performance trade-off in NAS is well-known. It has motivated researchers to explore the problem of designing better search spaces, resulting in several methods attempting to automate this process (Zhou et al., 2021; Hu et al., 2021; 2020). Generally speaking, automated search space design methods can be seen as mappings from one search space to another, $S \rightarrow A$, where typically $A \subset S$ and $|A| \ll |S|$. In other words, these methods focus on identifying a subset (A) of a larger search space (S) that exhibits desired statistical properties, which is then used by a chosen NAS algorithm to find individual models. Compared to performing NAS directly on S, they are more efficient due to the relaxed objective in the first stage, effectively resulting in hybrid systems combining

[*]Equal contribution [1]University of Warwick, UK [2]Samsung AI Centre Cambridge, UK [3]Cornell University, US [4]University of Cambridge, UK [5]Flower Labs, UK. Correspondence to: Hongkai Wen <hongkai.wen@warwick.ac.uk>.

Proceedings of the 41st International Conference on Machine Learning, Vienna, Austria. PMLR 235, 2024.

coarse and fine-grained searching.

However, these approaches still need to carefully design the initial search space S (e.g., as a pre-defined super network in Zhou et al. (2021); Hu et al. (2021; 2020)), start with a large enough initial search space that includes final optimal search space during shrink process (Hu et al., 2021; 2020), or begin with existing success network structure to ensure the final optimal search space can efficiently emerged, limiting the final NAS performance in a way similar to classical methods. On the other hand, existing approaches that focus on pruning or evolving a large initial S often rely on repeating a variation of the NAS process multiple times to provide feedback for updating S. This is also subject to bias, e.g. when assessing the fitness of different subspaces, and will almost certainly result in non-negligible additional cost (Zhou et al., 2021; Ci et al., 2021). In contrast, our goal is to scale the initial search space S to the sizes *unattainable* ($\sim 10^{390}$) by previous works, in an attempt to provide a more general-purpose mechanism with less dependence on the initial design of S. Simply put, we want to include as many models in S as possible without worrying too much about the resulting computational cost or human effort of doing so. Furthermore, we want to navigate a reasonable subspace from such an unconstrained initial search space based on the task conditions that provide an optimal configuration for the NAS process without repeating and feedback.

To navigate the resulting large space efficiently and in a context-dependant way, we are inspired by the recent advances in generative ML (Grathwohl et al., 2019), which shows impressive results in modelling highly-dimensional conditional distributions of various modalities such as images, text or speech. To extend this high-level idea to distributions of neural networks, we propose a hierarchical approach which breaks down the generation into several steps: *1)* first, we learn a reversible, continuous latent space of micro/cell designs in which "similar" designs cluster together – this is achieved by training a Graph Variational Autoencoder (G-VAE) regularized by the zero-cost (ZC) similarity of different graphs; *2)* then, we introduce a Conditional Continuous Normalizing Flow (CCNF) model as a way of finding "synonymous designs" in the G-VAE space efficiently; *3)* finally, a decoder-only sequence generator (SG) is trained to design macro architectures – for user-defined conditioning, it outputs a sequence of cells to form a full model, with architectural details of individual cells being abstracted away through the notion of a "cell type" obtained from the previous steps. The full process is schematically depicted in Fig. 1 with Sec. 4 discussing each step in detail. In summary, our work contributes in the following ways:

- We extend existing ZC proxies into a multidimensional vector form, studying their clustering behaviour on network graphs and their effectiveness.
- We employ semi-supervised learning for a G-VAE to achieve a reversible encoding of graphs into a latent space, preserving the property of the ZC space and we further perform clustering. This approach further enables us to quantize the graph design space into multiple "synonyms" sets without compromising graph performance and costs.
- We use a CCNF model to efficiently navigate the latent space of the G-VAE, which results in efficiently generated candidate cell (graph) architectures under specific conditions.
- We construct a decoder-only transformer to integrate with graph generation, forming a hierarchical network architecture generation mechanism that supports user-defined conditioning.

2. Related Work

Automated NAS Search Space Design. Optimizing NAS search spaces in an automated manner has attracted extensive interest recently, e.g. by progressively constraining the degree of freedom of the network design space (Radosavovic et al., 2020), interchangeably shrinking and expanding the initial search space (Ci et al., 2021), or employing evolutionary algorithms to evolve the initial space to an optimal subspace (Zhou et al., 2021). Schrodi et al. (2023) proposed a unifying framework to design hierarchical NAS search spaces with context-free grammars and used Bayesian optimization to search efficiently. Despite their differences in how to find the optimized subspaces, most of the existing approaches still need to iteratively evaluate the fitness of subspaces, i.e., assessing the performance of models within them, which often results in significant cost A large body of work has been developed to speed up model performance evaluation in general NAS context (Pham et al., 2018; White et al., 2021a; Dudziak et al., 2020), e.g., zero-cost proxies (Abdelfattah et al., 2021; Mellor et al., 2021; Xiang et al., 2023a) have been shown to correlate well with final trained accuracies, with negligible cost to compute.

Flow-based Models. Compared to other types of generative models (Creswell et al., 2018; Kingma & Welling, 2013), flow-based models are more efficient to sample, while being more stable to train and free of mode collapse and convergence issues, showcasing impressive performance in various generation tasks (Abdal et al., 2021; Yang et al., 2019; Li et al., 2022). Existing works often consider the continuous normalizing flows (CNF, Grathwohl et al. (2019)) based on neural ODEs (Chen et al., 2018), which are able to impose conditions during the sampling process, and thus control certain properties of the generated output, e.g., generating attribute-semantic edits (Abdal et al., 2021) or super-resolving images (Lugmayr et al., 2020). Our work is based on the existing works on flow-based models, but to the best

of our knowledge, we are the first to show that it is feasible to employ CCNFs to generate optimized neural architectures, balancing performance and efficiency.

Neural Architecture Generation. Neural architecture generation (NAG) has recently become a popular topic in the NAS community (Lee et al., 2021; Yu et al., 2023; An et al., 2023). Compared to conventional NAS, NAG methods often resort to some forms of generative models, such as diffusion models (An et al., 2023) or Generative Pre-Trained (GPT) models (Yu et al., 2023), aiming to generate candidate architectures with desired properties. Our work bears some resemblance to these methods, but is fundamentally different: *1)* instead of direct search/generation, we consider a hierarchical paradigm to efficiently explore the tremendously large initial search space; *2)* we consider a synonymous micro design approach leveraging the clustering properties of architectures, to effectively discover high-performing cells from given references; and *3)* we use a vector form of zero-cost proxies throughout the micro and macro design, achieving desired trade-off between cost to compute and transferability across different tasks.

3. Search Space Design and Challenge

As discussed in the introduction, the cost of the NAS process is strongly related to the search space size. In practice, to allow the search results to be suitable for a more diverse range of tasks, the previous search space design is typically large and suitable for specific NAS algorithms. This section starts by introducing the design details of our less-constraint initial search space, called **GraphNet**, and its overall search space size. Then, compared with large search space designs in previous NAS works designed for relatively general purposes, to show the challenge of the previous approach on GraphNet.

3.1. GraphNet Search Space Design

Micro search space. Our micro design space consists of any directed graph with up to 6 operation nodes, 1 input node and 1 output node. Operation nodes can be assigned one of 28 operations from the literature (9 op. types with 3 kernel size each + skip connection), and the n^{th} node can have up to n inputs – one for each previous node; values of different inputs are added before executing a selected operation. Each cell design includes the number of channels and strides as hyperparameters – operations within a cell are not allowed to change dimensions of intermediate results (unless internal to an operation), and a relevant projection is performed automatically before executing a cell to adjust input channels and possibly reduce spatial dimensions. Overall, without accounting for isomorphism and cell hyperparameters, the micro design space spans approximately up to 10^{17} configurations. Formally, the micro space is a set of pairs $\Omega = \{ (\mathbf{A}, \mathbf{F}) \}$ where $\mathbf{A} \in \{0,1\}^{8\times 8}$ is an adjacency matrix and $\mathbf{F} \in \{0,1\}^{8\times 13}$ is a feature matrix encoding operation type and kernel size (1-hot over the first 10 and last 3 dimensions). Details about operations can be found in App. C.

Macro search space. The macro search space is then defined as a sequence of up to 20 different cells $x_i \in \Omega$, with each cell being additionally parametrized by the number of channels $c_i \in C = \{8, 16, ..., 1024\}$ and stride $s_i \in \{1, 2\}$, for $i = 1, 2, ..., 20$. Suppose we assume different triples $t_i = [x_i, c_i, s_i]$ to be unique designs. In that case, the overall search space is a set of all sequences of any 20 such triples $S = (\Omega \times C \times \{1, 2\})^{20}$ and we can estimate the total number of points to be up to $(|\Omega| * 127 * 2)^{20} \approx 10^{390}$. We will use $\alpha(t_1, t_2, ..., t_{20}) \in S$ to refer to particular design points in the overall search space.

3.2. Challenges for NAS on GraphNet Design Space

We started to survey the previous NAS works that aimed to design a large search space for diverse NAS tasks, A detailed Table 9 is located in Appendix.G.

Previous efforts, such as those in the reinforcement learning approaches, used graph-based search spaces, such as Depict Network Topologies(Zoph & Le, 2017) and NASNet-A(Zoph et al., 2018), which required intensive computational resources. For instance, the Depict Network Topology requires approximately 537,600 GPU hours to explore a search space size of 1.12×10^{89}. On the other hand, Differentiable Architecture Search (DARTS) has significantly reduced the search cost by constraining the search space and adopting supernet strategies. However, these supernet strategies heavily rely on memory resources as a trade-off, posing significant challenges in scaling up for much larger search spaces. Recently, Zero-Cost NAS approaches, such as ZenNAS, have demonstrated substantial improvements in managing large search spaces with relatively lower evaluation costs. Nevertheless, they still require considerable repetitive evaluation during each NAS run under different conditions. This indicates a significant potential for optimizing costs through a more rational search space design.

Our GraphNet design space, as illustrated in the previous subsection, proposes a hierarchical search space that integrates sequential and graph-based elements, expanding the search space to an unprecedented scale of approximately 1×10^{390} and not specifically designed for weight sharing strategy that causes methods mentioned above can be extremely time-consuming and resource expensive in our GraphNet search space.

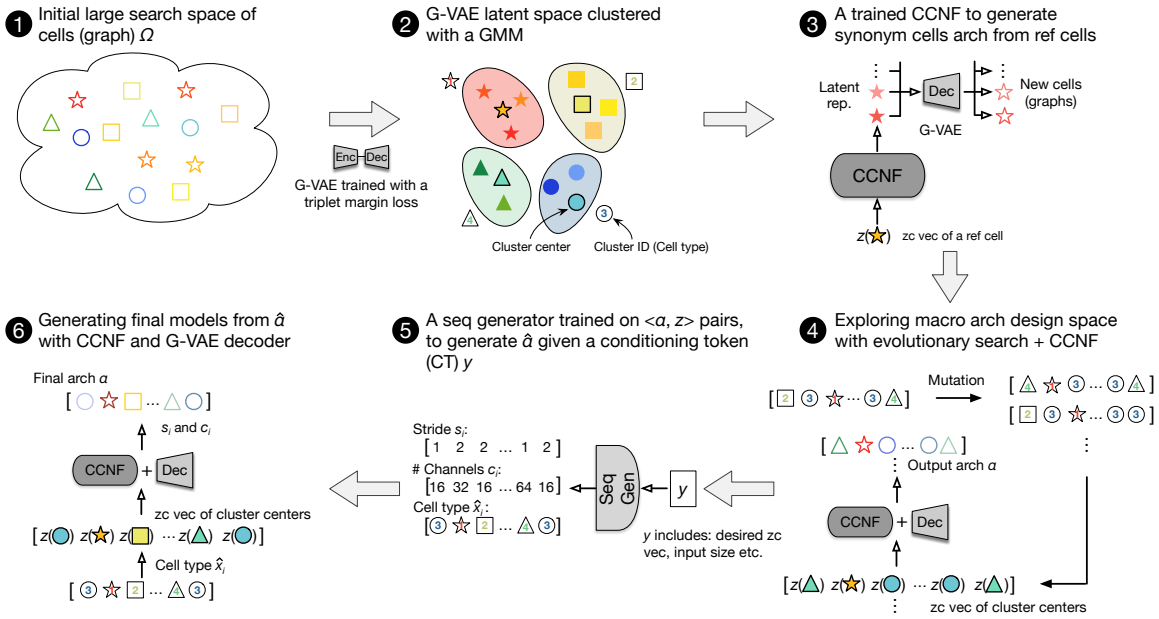

Figure 1. Overview of the proposed approach. ❶ We start from a large micro search space Ω, and ❷ train a G-VAE with triplet margin loss and use a GMM to partition the latent space into clusters. ❸ A CCNF model is then trained to generate synonym cells given a reference r. ❹ We explore the high-level search space $\hat{S}$ with ageing evolution and the trained CNF, and ❺ the set of explored architectures $\{\alpha\}$ is used to train a SG, which is able to generate the desired $\hat{\alpha}$ given a condition token (CT) y. ❻ The final architecture is then generated from $\hat{\alpha}$ using the CCNF and G-VAE decoder.

4. Method

This section outlines the technical details of our method and search space. We begin by discussing the clustering of similar micro designs in an invertible fashion in Sec. 4.1, then follow with synonym generation using a CCNF model in Sec. 4.2. Finally, we discuss full model generation with a decoder-only transformer in Sec. 4.3.

4.1. Clustering micro designs

Training a macro model generator to find high-quality models directly in such a large search space is obviously challenging. In order to simplify this task, we want to abstract away details of individual designs by automatically identifying families of similar cells. Later, the sequence generator only has to decide a family for each cell and the task of finding high-performance designs within each selected family is delegated to a lower-level search in the hierarchy.

Formally, given a metric space of micro designs (Ω, d), where $d(a, b)$ is a distance metric between designs a and b proportional to their relative quality, our goal is twofold: *1)* to partition Ω into a finite number of K clusters $\hat{\Omega}_{k=1,\dots,K}$ based on the metric d; *2)* to be able to sample new designs from each cluster easily.

The first important choice here is the metric d - naively relying on accuracy is undesired for two reasons: *1)* computing accuracy for a large number of models can be extremely expensive, a common challenge in NAS, but also *2)* it is known that different models can exhibit very different performance on different downstream tasks (Duan et al., 2021; Tu et al., 2022) – since we want to avoid bias towards any particular downstream task, we need something else.[1] To address this, we propose to use a generalized form of ZC proxies instead, by computing a vector of P proxies for a semi-fixed point from S. Specifically, for a micro design $x \in \Omega$, we define its ZC vector $z : \Omega \to \mathbb{R}^P$ as:

$$z(x) = \left[z^{(1)}_{\alpha_x}, z^{(2)}_{\alpha_x}, ..., z^{(P)}_{\alpha_x}\right], \quad (1)$$
$$\text{where: } \alpha_x = \alpha(t_1, t_2, ..., t_{17}) \quad \text{s.t.}$$
$$t_i = [x, c_i, s_i],\ c_i = 16 * 2^{\lfloor \frac{i}{6} \rfloor},\ s_i = \begin{cases} 2 & \text{if } i \in \{6, 12\} \\ 1 & \text{otherwise} \end{cases}$$

and $z^{(i)}_{\alpha_x}$ refers to computing the i^{th} zero-cost proxy for the model α_x obtained by stacking the same cell design into the above predefined macro-structure. Note that the ZC vector, while related to the original ZC metrics, serves a fundamentally different purpose – instead of strictly ordering models, which is a challenging task, we only use it to cluster designs that are likely to train to similar performance. Although the difference might seem nuanced, it is crucial for our method. Consider a simple proxy of the number of parameters. For example, sometimes very large models are desired if work-

[1] Conceptually, we could also consider a vector of accuracies on a diverse set of uncorrelated tasks, but that becomes even more expensive.

ing with large datasets, and sometimes small ones might be a better option. While there is no universal answer to choosing the right model size, we can reasonably expect if one small model works well, other small models should not be fundamentally wrong choices, and vice versa. This reasoning is the foundation for our design clustering. Given z, the metric d is then defined as $d(a,b) = L1(z(a), z(b))$. We will refer to this particular form as ZC-similarity[2]. In our experiments, we use a vector of 4 common proxies: *1)* NASWOT (Mellor et al., 2021), *2)* SNIP-SSNR (Xiang et al., 2023b), *3)* number of parameters (Ning et al., 2021), *4)* FLOPS.

Following the choice of the metric d, we employ a Gaussian Mixture Model (GMM) to identify different clusters in the ZC space. Specifically, we fit a mixture of multivariate normal distributions, parameterized with factors π_i and their means and variances μ_i, σ_i, to the observed values of z:

$$p(z) = \sum_{i=1}^{K} \pi_i \mathcal{N}(z|\mu_i, \sigma_i), \tag{2}$$

using the expectation-maximization algorithm. After that, models are assigned to clusters based on which component of the mixture they are most likely to belong to – this means one cluster is modelled by one component of the GMM with μ_i as its centre. The number of clusters is decided by plateauing Bayesian and Akaike information criteria.

The above provides us with a way of identifying families of cells, as shown in Fig. 2(a), but navigating the space Ω is still hard since we do not have a way of quickly providing designs that would belong to a certain family. To address this, we further learn an invertible mapping to a continues latent space $f : \Omega \to \mathbb{R}^m$ in which $L1$ distance approximates d, *i.e.*, $(F, L1) \approx (\Omega, q)$, where $F = \{f(x)\}_{x\in\Omega}$ is the set of embedded designs in this latent space and $\forall_{a,b\in\Omega} L1(f(a), f(b)) \propto d(a,b)$. This is done by training a G-VAE model (Kipf & Welling, 2016) additionally regularized with a triplet margin loss (Balntas et al., 2016) and an auxiliary ZC predictor q which tries to predict $z(x)$ from $f(x)$:

$$\begin{aligned}\mathcal{L}_{\text{VAE}} = &\overbrace{-\mathbb{E}_{p(f|x)} \log p(\tilde{x}|f)}^{\text{reconstruction (BCE)}} + \overbrace{\text{KL}[p(f|x)\,||\,p(f)]}^{\text{VAE regularization}} \\ &+ \underbrace{\mathbb{E}_{x,a,b} \max\left\{d(x,a)^2 - d(x,b)^2 + m, 0\right\}}_{\text{triplet margin}} \\ &+ \underbrace{\mathbb{E}_x (q(f(x)) - z(x))^2}_{\text{predictor (MSE)}},\end{aligned} \tag{3}$$

where $p(f|x)$ and $p(\tilde{x}|f)$ are distributions of encoded and reconstructed points, respectively; m is a margin term and triples $x, a, b \in \Omega^3$ are chosen s.t. for an anchor x, a and b are positive and negative examples, respectively. In practice, to ensure the decoder of the G-VAE can correctly decode as many points in the latent space as possible, it is important to scale its training set as much as possible. However, computing $z(x)$ and related distances $d(a,b)$ might easily become a bottleneck; therefore, we decided not to compute the regularization term in Eq. 3 all the time and only limit it to a relatively small random subset of all training points. After the G-VAE is trained, we can approximate f and f^{-1} with its encoder and decoder, respectively.

Given the G-VAE and the GMM, we can efficiently realize all required functionality of identifying clusters, finding a cluster to which a model belongs and sampling from clusters. The resultant latent space and its clustering properties w.r.t. z when trained naively vs. using our proposed regularization is illustrated in Fig. 2(b)-(c) – clearly, the regularized space preserves original clustering much better.

4.2. Generating synonymous micro designs

At this point, we have identified K different families of cells and have an easy way of sampling from Ω in a structured way thanks to our latent space. However, recall that we aim to be able to find high-performance designs in each family efficiently – this is still challenging since $K \ll |\Omega|$, meaning each cluster is still extremely big. To make the problem more approachable, for now, we will relax it to the problem of robustly finding synonyms of different designs – assuming we already know an example high-performance reference design r, we want $p(x|r)$ to be a distribution of designs with their accuracy centred around that of r. This problem is much simpler and should in fact be solvable by directly leveraging the properties of our latent space. In particular, $p(x|r)$ could be realized by sampling a neighborhood of r in the latent space: $f^{-1}(f(r) + \epsilon), \epsilon \sim \mathcal{N}$. However, we observe that even in a simple case of CIFAR-10 classification as a downstream task, a naive sampling strategy like this leaves some room for improvement, which we attribute to the sheer amount of possible models and imperfections of the VAE latent space.

To further improve sampling, let us first denote $p(x|r)$ in the latent space of the G-VAE as $p(f|r)$. We follow (Liu et al., 2019b) and utilize a CCNF formulation (Chen et al., 2018; Abdal et al., 2021) to map $\mathcal{N}$ to $p(f|r)$ by running forward in time from $u_0^{\rightarrow} = \epsilon$ to $u_T^{\rightarrow} = u_0^{\rightarrow} + \int_{t=0}^{T} g(u_t, t, z(r))dt$, according to the dynamics parameterized by a neural network g. By repeating the forward process for different samples ϵ and a given reference r we can obtain a distribution of generated designs: $p(u_T^{\rightarrow}|r)$. To ensure $p(u_T^{\rightarrow}|r)$ approximates $p(f|r)$, parameters of g are fit to minimize negative log-likelihood of generating $f(r)$ from $z(r)$, following the

[2]Strictly speaking, ZC-similarity should be understood as the opposite of the $L1$ distance since similar models should be close, *i.e.*, $\uparrow$ similarity $=\downarrow L1$.

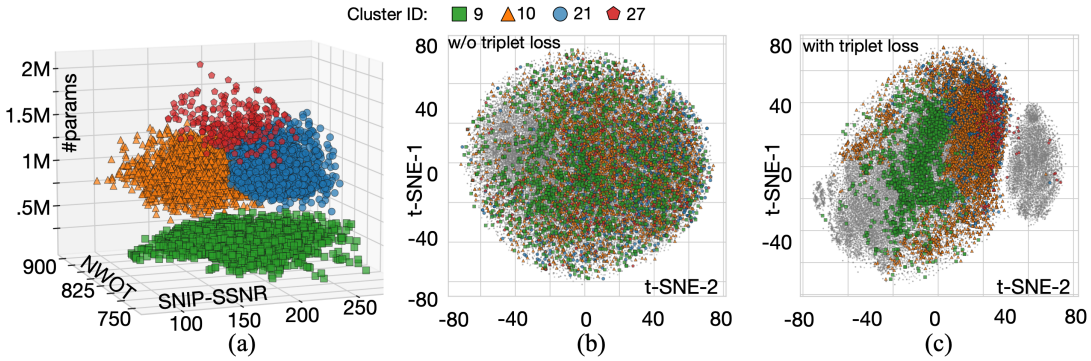

Figure 2. (a) Example of 4 clusters of micro designs identified with a GMM w.r.t. the ZC vectors z. (b) t-SNE visualization of the latent space obtained from a G-VAE trained naively vs. (c) using triplet loss regularization (the same 4 clusters highlighted). Unregularized space does not preserve ZC similarity of the models.

reverse process:

$$\mathcal{L}_{\text{CCNF}} = \mathbb{E}_{p(r)} - \log p(u_T^{\leftarrow}|r) = \tag{4}$$

$$\mathbb{E}_{p(r)}\Big[- \log p(u_0^{\leftarrow}) + \int_{t=0}^{T} \text{Tr}\Big(\mathbf{J}_{u_t} g(u_t, t, z(r))\Big) dt\Big],$$

$$\text{where:} \quad u_T^{\leftarrow} = f(r),\ u_0^{\leftarrow} = u_T^{\leftarrow} - \int_{t=0}^{T} g(u_t, t, z(r)) dt,$$

and $\mathbf{J}_{u_t} g$ is Jacobian of g w.r.t. u_t. Conditioning on the reference r is done using the same $z(r)$ encoding as in Sec. 4.1 for consistency with the G-VAE. We use Hutchinson's trace estimator to efficiently estimate the trace in Eq. 4 (Grathwohl et al., 2019) and optimize using gradient descent via the adjoint method (Chen et al., 2018). Furthermore, similar to training the G-VAE, to scale training of the CCNF without evaluating $z(r)$ for the excessive amount of models, we utilize the previously trained predictor q.

After training finishes, $p(x|r)$ is recovered by simply applying the G-VAE decoder to the samples generated by the CCNF. The overall training and sampling process is summarized in Appendix. B. We validate the quality of designs generated by the CCNF and compare it to naive sampling in the latent space in Apprndix. BTab. 8 and Tab. 7. Notably, by sampling with a CCNF, we are able to find a better design, while neighbourhood sampling fails to achieve this.

4.3. Macro architecture generation

Clustering the micro design space and being able to generate high-performance architectures from each cluster efficiently enables us to factorize the search space S into a two-level hierarchy by replacing individual points from Ω with identified clusters: $\hat{S} = (\{\hat{\Omega}_k\}_{k=1}^{K} \times C \times \{1, 2\})^{20}$. This, in turn, reduces the space of macro designs from c. 10^{390} to $(K|C||S|)^{20} \approx 10^{78}$. We will refer to $\hat{S}$ as a higher-level (HL) search space to distinguish it from the original S. Similarly, we will call elements of $\hat{S}$ HL designs/sequences and denote them with $\hat{\alpha}(\hat{t}_1, ..., \hat{t}_{20})$.

Within the HL search space, each triple $\hat{t}$ represents a large number of different possible triples t, depending on which design x is selected from a cluster $\hat{x} \in \{\hat{\Omega}_k\}_{k=1}^{K}$. We want to use our CCNF to resolve this, but we face the problem of selecting a reference point – if we already know a strong design point, we can use it and focus our search towards similar designs; this is related to summarizing each cluster with an approximation of $\min_{x \in \hat{x}} \mathcal{L}(\alpha_x)$. However, knowing good designs in advance is a strong assumption. Therefore, we opt for a simpler but more realistic approach of using the most representative models from each cluster as reference – this is conveniently done by conditioning the CCNF on each cluster's mean μ_i, and is more closely related to considering $\mathbb{E}_{x \in \hat{x}} \mathcal{L}(\alpha_x)$. Formally, for the HL design $\hat{\alpha} \in \hat{S}$, a random design $\alpha(\hat{\alpha}) \in S$ can be defined as:

$$\alpha(\hat{\alpha}) = \alpha(t_1, ..., t_{20}), \text{ where: } t_i = [\omega(\hat{x}_i), \hat{c}_i, \hat{s}_i], \tag{5}$$

$$\omega(\hat{x}) = f^{-1}\Big(\epsilon + \int_{t=0}^{T} g(u_t, t, \mu(\hat{x}))\Big), \tag{6}$$

$[\hat{x}_i, \hat{c}_i, \hat{s}_i] = \hat{t}_i$ are elements of HL design $\hat{\alpha}$ and $\mu(\hat{x})$ is the mean of cluster $\hat{x}$.

Having chosen a strategy to sample from each cluster, we then start considering different HL designs for the purpose of training a macro architecture generator. Our goal is to train a sequence generator (SG) h that would propose different parametrized cells in an autoregressive fashion, given a starting condition y: $\hat{t}_i = h(..., \hat{t}_{i-1}, y)$. The conditioning token y is similar to the one used for CCNF and guides the generation process towards macro designs with high-level parameters similar to those in y. In particular, in our experiments, we use y to constrain FLOPs and parameters. We use decoder-only transformer (Radford et al., 2018) model for h. Specifically, we fine-tune a pretrained GPT-Neo-125M model provided by Black et al. (2021), by minimizing the negative log-likelihood of generating observed macro sequences from their relevant values of y. However, naively training the SG on a random sample of models would result in effectively performing a random search among the models meeting the requested y. To mitigate this, we run

an evolutionary search (ES) in the HL space (using a CCNF to construct cells) to find designs that maximize T-CET, a state-of-the-art zero-cost proxy (Xiang et al., 2023b), for various size constraints. We then train the SG on the history of the ES, resulting in a bias towards models navigating the trade-off between T-CET and parameters:

$$\mathcal{L}_{\text{SG}} = \mathbb{E}_{\hat{\alpha}\in H} \sum_i -\log\, p(\hat{t}_i|...,\hat{t}_{i-1}, y), \qquad (7)$$
$$\text{where: } y = z(\alpha(\hat{\alpha})),$$
$$H \approx \arg\max_{\hat{\alpha}}\ \text{T-CET}(\alpha(\hat{\alpha})),$$

$\hat{t}_i$ are elements of sequence $\hat{\alpha}$ and Eq. 4.3 refers to performing the ES (details of the HL-Evo algorithm can be found in Algo. 1). Overall, after the training has finished, the distribution of architectures generated for a given condition y can be summarized as:

$$p(\alpha|y) = p(\alpha(\hat{\alpha})), \qquad (8)$$
$$\text{where } \hat{\alpha} = [\hat{t}_1, ..., \hat{t}_{20}], \quad \hat{t}_i = h(..., \hat{t}_{i-1}, y),$$

and randomness is introduced through ϵ in $\alpha(\hat{\alpha})$ (see Eq. 5-6). Note that we employ deterministic sampling at the macro level.

Details of our HL-EVO Algorithm Alg. 1 summarizes our HL evolutionary search algorithm (HL-EVO), which combines evolutionary search with the pre-trained CCNF to discover good-performing macro architectures. Our HL-EVO considers only mutations – parents are randomly selected, while the offspring with low fitness are dropped. In practice, mutations may occasionally produce child architectures that do not satisfy the pre-defined constraints – they are simply discarded by the algorithm, which then moves forward to the next iteration. In particular, our HL-EVO operates in two phases, *exploration* and *exploitation*, with the equal budget on the number of iterations. In the exploration phase, we allow the search to mutate across different cell families for each $\hat{x}_i \in \hat{\alpha}(\hat{t}_1, ..., \hat{t}_{20})$, so it can explore various combinations of cell families in the macro sequence. On the other hand, in the later exploitation phase, we restrict the search to fix the current sequence of cell families $\hat{x}_i$, but only mutate channel $\hat{c}_i$ and stride $\hat{s}_i$, in hope to keep the already explored promising macro sequences, while relying on the CCNF to subsequently exploit good-performing final architectures from these macro sequences $\alpha(\hat{\alpha})$.

In our experiments, we use $T = 20000$ and $N = 512$. We choose a relatively small initial architecture α_0 as $[(\hat{x}^{(5)}, 32, 1), (\hat{x}^{(5)}, 64, 2), (\hat{x}^{(5)}, 64, 1)]$, where $\hat{x}^{(5)}$ is the mean of the 5-th cluster (cell family).

Final model selection. The above sampling procedure is supposed to produce a small, optimized search space for a given conditioning y. As such, it should be used with a separate searching algorithm on top of the generated models. Although many approaches could be tried for simplicity and to keep the cost low, we opted to use a simple selection based on T-CET in our experiments. Different searching strategies are compared in App. F.1.

Algorithm 1 HL-Evo

input Search space S (with related $\hat{S}$ and $\{\hat{\Omega}_k\}_{k=1}^K$),
objective $\mathcal{L}$, total number of iteration T,
population size N, initial architecture α_0,
a trained CCNF model g and a G-VAE decoder f^{-1}
output A selected architecture $\alpha^\star$, search history H

function `GenCell`($\hat{\Omega}$)
  $\epsilon \sim \mathcal{N}$
  $x \leftarrow f^{-1}\big(\epsilon + \int_{t=0}^{T} g(u_t, t, \mu(\hat{\Omega}))dt\big)$
  return x
end function
function `Mutate`(α, exploration)
  take random cell $t = [x, c, s] \in \alpha$
  let $\hat{x}$ be the cluster to which x belongs
  select random mutation m
  if m = change channels **then**
    sample new $c \sim \mathcal{U}\{C\}$
  else if m = change stride **then**
    sample new $s \sim \mathcal{U}\{1, 2\}$
  else if m = change cell design **then**
    generate new $x \leftarrow$ `GenCell`($\hat{x}$)
  else if exploration **and** m = change cell type **then**
    sample new cluster $\hat{x} \sim \mathcal{U}\{\hat{\Omega}_k\}$
    generate new cell $x \leftarrow$ `GenCell`($\hat{x}$)
  end if
  replace t in α with its modified version
  return modified α
end function
$H = \{\alpha_0\}$
$P \leftarrow H$
for e $\in$ { `true`, `false` } **do**
  for $i \leftarrow 0$ **to** $T/2$ **do**
    take random model from the population $\alpha \sim \mathcal{U}\{P\}$
    $\alpha' =$ `Mutate`(α, e)
    if $|P| \geq N$ **then**
      $\alpha'' \leftarrow \arg\max_{x\in P} \mathcal{L}(x)$
      $P \leftarrow P \setminus \{\alpha''\}$
    end if
    $P \leftarrow P \cup \{\alpha'\}$
    $H \leftarrow H \cup \{\alpha'\}$
  end for
end for
$\alpha^\star \leftarrow \arg\min_{x\in H} \mathcal{L}(x)$

5. Evaluation

We evaluate our method on the typical datasets: CIFAR-10, CIFAR-100 (Krizhevsky, 2009) and ImageNet-1k (Deng et al., 2009), abbreviated C-10, C-100 and IN-1k, respectively. To test the generalizability of our method to less-studied tasks, we also evaluate the recently proposed NAS-Bench-360 (NB360, (Tu et al., 2022)). Details about pre-training architecture, hyperparameters and cost can be found

Table 1. Comparison with SOTA ZC NAS. All methods were run using the same protocol: up to "ZC evals." models were scored using a selected ZC proxy, where the model selection was guided with either evolution, HL evolution, or our SG. The top-N models according to each proxy were trained, and the best one is reported – for CIFARs, N=10, for IN-1k, N=1. Constraints were denoted under each dataset's name, and models that violated these were discarded. "Cost" refers to searching cost in GPU hours.

Design Space	Method	Pretraining time	ZC evals.	C-10 (1M param.)		C-100 (1M param.)		IN-1k (450M FLOPs)	
				Acc.(%)	Cost	Acc.(%)	Cost	Acc.(%)	Cost
ZenNet‡	Evo(Zen)	-	480,000	96.2	14	80.1	14	-	-
	Evo(ZiCo)	-	480,000	97.0	10	80.2	10	-	-
	Evo(T-CET)	-	480,000	97.2	19	80.4	19	-	-
GraphNet	Evo(T-CET)	-	480,000	96.4	22	79.6	22	77.6	20
	HL-Evo(T-CET)+CCNF	5	20,000	97.4	3.6	80.9	3.6	78.4	6
	SG+CCNF	30	100	97.6	0.08	81.0	0.08	78.5	0.08

in App. A.

5.1. Systematic comparison on CIFAR and ImageNet

We begin by comparing our method to other approaches based on zero-cost proxies in a controlled setting where all models are trained using exactly the same protocol, architecture constraints and (max) search budget. We also ablate our method by progressively adding different components and quantifying their effects when performing a search in our search space (which we call **GraphNet**). The results are presented in Tab. 1.

We compare to evolution-based ZC NAS approaches: ZenNAS (Lin et al., 2021), ZiCo (Li et al., 2023) and T-CET (Xiang et al., 2023b). We do not compare to supernet-based ZC NAS, such as TE-NAS (Chen et al., 2021) or Zero-Cost-PT (Xiang et al., 2023a), since constructing a supernet for those methods is impossible in our search space. Overall, we can see that models designed in the GraphNet search space using our proposed generative models achieve superior performance – we attribute this simply to the fact that our search space is much larger, *i.e.*, it is a strict superset of ZenNet design space. However, we can also see that simply increasing the search space size is not enough – when naively using existing methods in our large space, the performance is likely to drop even for a fairly large computational budget (*e.g.*, Evo(T-CET), which is the best on ZenNet, drops by 0.8 percentage points on both C-10 and C-100 when run in the GraphNet space). Compared to these methods, by utilizing our G-VAE and CCNF and running evolution in the HL space, we can already improve upon all existing methods while being significantly cheaper (HL-Evo) – this is primarily due to the informative organization of the search space (Sec. 4.1). Finally, we can amortize the cost of running the HL ES each time by training our SG, which then allows us to achieve comparable, if not slightly better, performance in a matter of minutes – this property is desired since it allows us to mitigate the "cold start" problem in NAS, addressed by some recent work (Zhao et al., 2023). More experiments using different algorithms on our GraphNet space can be found in App. F.2.

5.2. Open-world comparison on ImageNet

Here we compare to other NAS approaches that can be found in the literature without requiring them to run in an aligned setting – the only common trait is the models should be (roughly) smaller than 450M FLOPs. Our focus is on positioning our work w.r.t. the state-of-the-art approaches in the 3 most relevant lines of work: *1)* zero-cost and low-cost NAS, *2)* automated search space design, and *3)* generative NAS. We present a detailed comparison in Tab. 2, including our best effort to highlight any differences in search spaces and training schemes. We can see that our method achieves very strong performance while requiring minimal computation resources. The only two methods that achieve better results are OFA (when initialized with weights from the supernet) and GPT NAS. However, the former requires extensive pretraining while the latter is almost 33$\times$ larger (the authors do not report FLOPs so we included the model size, but it probably violates the FLOPs constraint as well). The results of our method are averaged by searching 3 times, each time training the resulting network once.The architecture and further discussion can be find in App. F

5.3. Architecture Discovered and Further Discussion on Performance

We present our SG+CCNF architecture for ImageNet 1k tasks with constrained budgets of 450M and 600M FLOPs, as shown in Fig. 4 and 5 at App.F. For the macro-structures satisfying FLOP constraints, most blocks use stride 2 to reduce feature map size. In the 600M model, the last block uses stride 1, leveraging the higher FLOP allowance.

From a microstructure perspective, mbconv is the predominant choice, with dil_conv also frequently used. Dense residual connections are prevalent, ensuring gradient flow during training. Each graph in the proposed networks is sampled from the Pareto Front of zc scores to the flops/param cost,

Table 2. Comparison with various NAS approaches on ImageNet-1k for budgets of 450M, 600M and 1000M FLOPs. All numbers other than "Ours" are directly taken from relevant papers. OS – One-Shot, SN – supernet, ZC – zero-cost. Cost in GPU hours.

Model	Design space	Training scheme	Pre-cost	Search cost	Acc.(%)	Params.(M)	FLOPs(M)	Approach
OFA (Cai et al., 2020)	MobileNetv3	Custom	1,200	40	79.1	?	389	OS + predictor
OFA scratch (Cai et al., 2020)	MobileNetv3	Custom	1,200	40	77.0	?	389	OS + predictor
OFA scratch (Moons et al., 2021)	MobileNetv3	EfficientNet	1,200	40	77.7	?	389	OS + predictor
GPT NAS (Yu et al., 2023)	Custom	?	?	96	79.1	110.9	?	GPT gen.
NSENet† (Ci et al., 2021)	Custom	EfficientNet	?	c. 4,000	77.3	7.6	333	SN evo. + OS
NSENet-GPU (Ci et al., 2021)	Custom	EfficientNet	?	c. 4,000	77.9	15.7	?	SN evo. + OS
AutoSpace† (Zhou et al., 2021)	Custom	?	960	?	77.5	5.7	380	SN evo. + OS
TE-NAS (Chen et al., 2021)	DARTS	?	0	4.1	75.5	5.4	<600	ZC + SN
Zero-Cost-PT (Xiang et al., 2023a)	Proxyless	Proxyless	0	1.0	76.4	8.0	?	ZC + SN
ZenNet (400M-SE) (Lin et al., 2021)	MobileNetv2	ZenNAS	0	12	78.0	5.7	410	ZC evo.
ZiCo (450M) (Li et al., 2023)	MobileNetv2	ZenNAS	0	9.6	78.1±0.3	?	448	ZC evo.
Ours	GraphNet	ZenNAS	30	0.08	78.3±0.2	4.3	429	ZC gen. + ZC sel.
ZiCo (600M) (Li et al., 2023)	MobileNetv2	ZenNAS	0	9.6	79.4±0.3	?	603	ZC evo.
Ours	GraphNet	ZenNAS	30	0.08	79.5±0.1	7.0	593	ZC gen + ZC sel.
ZiCo (1000M) (Li et al., 2023)	MobileNetv2	ZenNAS	0	9.6	80.5±0.2	?	1005	ZC evo.
Ours	GraphNet	ZenNAS	30	0.08	80.6±0.4	8.2	964	ZC gen + ZC sel.

aiming to maximize zc scores while minimizing cost. The cluster distribution in the zero-cost vector space is illustrated in Fig. 6 at App.F .

While our method may not significantly improve accuracy alone, it effectively amortizes search cost and enhances accuracy, a challenging achievement. Considering ImageNet is a well-studied task, we adhered to previous training procedures for fair comparison, acknowledging that substantial improvements are not expected.

5.4. NAS-Bench-360

Table 3. Results on the tasks from NB360. Metrics and baselines follow what is reported by (Shen et al., 2022). Lower is better for all tasks. Best in bold, second best underlined. We use WRN (Zagoruyko & Komodakis, 2016) as a reference and consider Ours better than DASH (Shen et al., 2022) for the ECG task due to the lower search cost.

Method	Spherical	Darcy Flow	PSICOV	Cosmic	NinaPro	ECG	Satellite	DeepSEA
Expert	67.41	0.008	3.35	**0.13**	8.73	**0.28**	19.8	0.30
WRN	85.77	0.073	3.84	0.24	6.78	0.43	15.49	0.40
DASH	71.28	**0.008**	3.30	0.19	6.60	0.32	**12.28**	**0.28**
Ours	**65.72**	0.011	**3.02**	0.14	**6.30**	0.32	18.11	0.50

To run on NB360, we first have to decide on sensible conditioning for our SG. Unfortunately, unlike the previous benchmarks, NB360 does not have well-established constraints that we could use and compare to other methods with a similar budget — to resolve this, we decided to use #params and #FLOPs of the Wide-ResNet (WRN) as this is the most basic baseline considered by (Tu et al., 2022). Specifically, we use WRN for each task backbone structure (backbone is different for each task, on hyper-parameters like stride size, etc.) to CIFAR100 task and calculate zero-cost metrics for WRN. Then, for each task, use network generate to generate candidates' networks based on the zero-cost metrics we have as reference conditions. Afterwards, we adopted the candidates' networks with current task output layers, calculated zero cost metrics for each task data, ranked the candidates with zero cost metrics, and selected and trained top-10 models. We report the best model we obtained from validation. We compare to DASH (Shen et al., 2022), a recent NAS method designed specifically with NB360 in mind, and expert-designed networks in Tab. 3. We can see our generative approach achieves very competitive performance. Significantly, we can improve upon the WRN reference in 6/8 cases. Moreover, we improve upon DASH in 4/8 cases, despite DASH performing supernet training and HPO for each task, whereas we do not use any feedback. Note that due to the amortised cost of training the SG, we are also much faster — generating models for new tasks takes minimal time. Still, there is room for improvement, considering our method fails noticeably on DeepSEA and Satellite tasks. We leave investigating those failure cases for future work.

6. Conclusion

We have presented a novel method to efficiently generate high-performance DNN architectures for user-defined architectural constraints by using an elaborated hierarchy of generative models guided by the clustering properties of zero-cost proxies. Our approach achieves state-of-the-art performance on common tasks, including avg. 78.3%, 79.5% and 80.6% top-1 accuracy on ImageNet-1k for the budgets of 450M, 600M and 1000M FLOPs. We hope our work serves as a proof of concept that generative methods can be successfully used for NAS without incurring prohibitive costs. We include more discussion on limitations of our approach and future work in App. H.

Impact Statement

This paper presents work whose goal is to advance the field of Machine Learning. There are many potential societal consequences of our work, none which we feel must be specifically highlighted here.

Acknowledgements

Part of the experiments were performed using the Sulis Tier 2 HPC platform hosted by the Scientific Computing Research Technology Platform at the University of Warwick. Sulis is funded by EPSRC Grant EP/T022108/1 and the HPC Midlands+ consortium.

References

Abdal, R., Zhu, P., Mitra, N. J., and Wonka, P. Styleflow: Attribute-conditioned exploration of stylegan-generated images using conditional continuous normalizing flows. *ACM Transactions on Graphics (ToG)*, 40(3):1–21, 2021.

Abdelfattah, M. S., Mehrotra, A., Dudziak, Ł., and Lane, N. D. Zero-cost proxies for lightweight NAS. In *International Conference on Learning Representations (ICLR)*, 2021.

An, S., Lee, H., Jo, J., Lee, S., and Hwang, S. J. DiffusionNAG: Task-guided neural architecture generation with diffusion models, 2023.

Ba, J. L., Kiros, J. R., and Hinton, G. E. Layer normalization, 2016.

Balntas, V., Riba, E., Ponsa, D., and Mikolajczyk, K. Learning local feature descriptors with triplets and shallow convolutional neural networks. In *British Machine Vision Conference (BMVC)*, 2016.

Black, S., Gao, L., Wang, P., Leahy, C., and Biderman, S. GPT-Neo: Large Scale Autoregressive Language Modeling with Mesh-Tensorflow, 2021. URL https://doi.org/10.5281/zenodo.5297715.

Brown, T., Mann, B., Ryder, N., Subbiah, M., Kaplan, J. D., Dhariwal, P., Neelakantan, A., Shyam, P., Sastry, G., Askell, A., Agarwal, S., Herbert-Voss, A., Krueger, G., Henighan, T., Child, R., Ramesh, A., Ziegler, D., Wu, J., Winter, C., Hesse, C., Chen, M., Sigler, E., Litwin, M., Gray, S., Chess, B., Clark, J., Berner, C., McCandlish, S., Radford, A., Sutskever, I., and Amodei, D. Language models are few-shot learners. In *Advances in Neural Information Processing Systems (NeurIPS)*, 2020.

Cai, H., Gan, C., Wang, T., Zhang, Z., and Han, S. Once for all: Train one network and specialize it for efficient deployment. In *International Conference on Learning Representations (ICLR)*, 2020.

Chen, R. T., Rubanova, Y., Bettencourt, J., and Duvenaud, D. K. Neural ordinary differential equations. *Advances in Neural Information Processing Systems (NeurIPS)*, 31, 2018.

Chen, W., Gong, X., and Wang, Z. Neural architecture search on imagenet in four gpu hours: A theoretically inspired perspective. In *International Conference on Learning Representations (ICLR)*, 2021.

Ci, Y., Lin, C., Sun, M., Chen, B., Zhang, H., and Ouyang, W. Evolving search space for neural architecture search. In *IEEE/CVF International Conference on Computer Vision (ICCV)*, 2021.

Creswell, A., White, T., Dumoulin, V., Arulkumaran, K., Sengupta, B., and Bharath, A. A. Generative adversarial networks: An overview. *IEEE signal processing magazine*, 35(1):53–65, 2018.

Deng, J., Dong, W., Socher, R., Li, L.-J., Li, K., and Fei-Fei, L. ImageNet: A Large-Scale Hierarchical Image Database. In *IEEE/CVF Conference on Computer Vision and Pattern Recognition (CVPR)*, 2009.

Dong, X., Liu, L., Musial, K., and Gabrys, B. NATS-Bench: Benchmarking NAS algorithms for architecture topology and size. *IEEE Transactions on Pattern Analysis and Machine Intelligence (TPAMI)*, 44, 2021.

Duan, Y., Chen, X., Xu, H., Chen, Z., Liang, X., Zhang, T., and Li, Z. TransNAS-Bench-101: Improving transferability and generalizability of cross-task neural architecture search. In *IEEE/CVF Conference on Computer Vision and Pattern Recognition (CVPR)*, 2021.

Dudziak, Ł., Chau, T., Abdelfattah, M. S., Lee, R., Kim, H., and Lane, N. D. BRP-NAS: Prediction-based NAS using GCNs. In *Advances in Neural Information Processing Systems (NeurIPS)*, 2020.

Grathwohl, W., Chen, R. T., Bettencourt, J., Sutskever, I., and Duvenaud, D. Ffjord: Free-form continuous dynamics for scalable reversible generative models. In *International Conference on Learning Representations (ICLR)*, 2019.

Hairer, E., Norsett, S., and Wanner, G. *Solving Ordinary Differential Equations I: Nonstiff Problems*, volume 8. 1993. ISBN 978-3-540-56670-0.

He, K., Zhang, X., Ren, S., and Sun, J. Deep residual learning for image recognition. In *IEEE/CVF Conference on Computer Vision and Pattern Recognition (CVPR)*, 2016.

Hendrycks, D. and Gimpel, K. Gaussian error linear units (GELUs). *arXiv preprint arXiv:1606.08415*, 2016.

Hu, Y., Liang, Y., Guo, Z., Wan, R., Zhang, X., Wei, Y., Gu, Q., and Sun, J. Angle-Based search space shrinking for neural architecture search. In *European Conference on Computer Vision (ECCV)*, 2020.

Hu, Y., Wang, X., Li, L., and Gu, Q. Improving One-Shot NAS with Shrinking-and-Expanding supernet. *Pattern Recognit.*, 118:108025, October 2021.

Ioffe, S. and Szegedy, C. Batch normalization: Accelerating deep network training by reducing internal covariate shift. In *International Conference on Machine Learning (ICML)*, 2015.

Kingma, D. P. and Ba, J. Adam: A method for stochastic optimization. In *International Conference on Learning Representations, (ICLR)*, 2015.

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

Kipf, T. N. and Welling, M. Variational graph auto-encoders. *NIPS Workshop on Bayesian Deep Learning*, 2016.

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

Krizhevsky, A. Learning Multiple Layers of Features from Tiny Images, 2009. URL https://www.cs.toronto.edu/~kriz/cifar.html.

Lee, H., Hyung, E., and Hwang, S. J. Rapid neural architecture search by learning to generate graphs from datasets. In *International Conference on Learning Representations (ICLR)*, 2021.

Li, G., Yang, Y., Bhardwaj, K., and Marculescu, R. ZiCo: Zero-shot NAS via inverse coefficient of variation on gradients. In *International Conference on Learning Representations (ICLR)*, 2023.

Li, Y.-k., Lien, Y.-H., and Wang, Y.-S. Style-structure disentangled features and normalizing flows for diverse icon colorization. In *IEEE/CVF Conference on Computer Vision and Pattern Recognition (CVPR)*, 2022.

Lin, M., Wang, P., Sun, Z., Chen, H., Sun, X., Qian, Q., Li, H., and Jin, R. Zen-NAS: A zero-shot nas for high-performance deep image recognition. In *IEEE/CVF International Conference on Computer Vision (ICCV)*, 2021.

Liu, H., Simonyan, K., and Yang, Y. DARTS: Differentiable architecture search. In *International Conference on Learning Representations (ICLR)*, 2019a.

Liu, J., Kumar, A., Ba, J., Kiros, J., and Swersky, K. Graph normalizing flows. In *Advances in Neural Information Processing Systems (NeurIPS)*, 2019b.

Loshchilov, I. and Hutter, F. SGDR: Stochastic gradient descent with warm restarts. In *International Conference on Learning Representations (ICLR)*, 2017.

Loshchilov, I. and Hutter, F. Decoupled weight decay regularization. In *International Conference on Learning Representations (ICLR)*, 2019.

Lugmayr, A., Danelljan, M., Van Gool, L., and Timofte, R. SRFlow: Learning the super-resolution space with normalizing flow. In *European Conference on Computer Vision (ECCV)*, 2020.

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

Moons, B., Noorzad, P., Skliar, A., Mariani, G., Mehta, D., Lott, C., and Blankevoort, T. Distilling optimal neural networks: Rapid search in diverse spaces. In *IEEE/CVF International Conference on Computer Vision (ICCV)*, 2021.

Narayanan, D., Shoeybi, M., Casper, J., LeGresley, P., Patwary, M., Korthikanti, V. A., Vainbrand, D., Kashinkunti, P., Bernauer, J., Catanzaro, B., Phanishayee, A., and Zaharia, M. Efficient large-scale language model training on gpu clusters using megatron-lm. In *Supercomputing Conference (SC)*, 2021.

Niklaus, S., Mai, L., and Liu, F. Video frame interpolation via adaptive separable convolution. In *IEEE International Conference on Computer Vision (ICCV)*, 2017.

Ning, X., Tang, C., Li, W., Zhou, Z., Liang, S., Yang, H., and Wang, Y. Evaluating efficient performance estimators of neural architectures. In *Advances in Neural Information Processing Systems (NeurIPS)*, 2021.

Pham, H., Guan, M., Zoph, B., Le, Q., and Dean, J. Efficient neural architecture search via parameters sharing. In *International Conference on Machine Learning (ICML)*, 2018.

Radford, A., Narasimhan, K., Salimans, T., Sutskever, I., et al. Improving language understanding by generative pre-training. 2018.

Radosavovic, I., Kosaraju, R. P., Girshick, R., He, K., and Dollár, P. Designing network design spaces. In *IEEE/CVF Conference on Computer Vision and Pattern Recognition (CVPR)*, 2020.

Schrodi, S., Stoll, D., Ru, B., Sukthanker, R., Brox, T., and Hutter, F. Construction of hierarchical neural architecture search spaces based on context-free grammars. In *Advances in Neural Information Processing Systems (NeurIPS)*. (to appear), 2023.

Shen, J., Khodak, M., and Talwalkar, A. Efficient architecture search for diverse tasks. In *Advances in Neural Information Processing Systems (NeurIPS)*, 2022.

Tu, R., Roberts, N., Khodak, M., Shen, J., Sala, F., and Talwalkar, A. NAS-bench-360: Benchmarking neural architecture search on diverse tasks. In *Advances in Neural Information Processing Systems (NeurIPS)*, 2022.

Vaswani, A., Shazeer, N., Parmar, N., Uszkoreit, J., Jones, L., Gomez, A. N., Kaiser, Ł., and Polosukhin, I. Attention is all you need. *Advances in Neural Information Processing Systems (NeurIPS)*, 2017.

White, C., Neiswanger, W., and Savani, Y. Bananas: Bayesian optimization with neural architectures for neural architecture search. In *AAAI Conference on Artificial Intelligence (AAAI)*, 2021a.

White, C., Zela, A., Ru, R., Liu, Y., and Hutter, F. How powerful are performance predictors in neural architecture search? *Advances in Neural Information Processing Systems (NeurIPS)*, 2021b.

White, C., Safari, M., Sukthanker, R., Ru, B., Elsken, T., Zela, A., Dey, D., and Hutter, F. Neural architecture search: Insights from 1000 papers. *arXiv preprint arXiv:2301.08727*, 2023.

Xiang, L., Dudziak, Ł., Abdelfattah, M. S., Chau, T., Lane, N. D., and Wen, H. Zero-cost operation scoring in differentiable architecture search. In *AAAI Conference on Artificial Intelligence (AAAI)*, 2023a.

Xiang, L., Hunter, R., Xu, M., Dudziak, Ł., and Wen, H. Exploiting network compressibility and topology in zero-cost NAS. In *AutoML Conference*, 2023b.

Yang, G., Huang, X., Hao, Z., Liu, M.-Y., Belongie, S., and Hariharan, B. PointFlow: 3d point cloud generation with continuous normalizing flows. In *IEEE/CVF International Conference on Computer Vision (ICCV)*, 2019.

Yu, C., Liu, X., Feng, W., Tang, C., and Lv, J. GPT-NAS: Evolutionary neural architecture search with the generative pre-trained model, 2023.

Yu, F. and Koltun, V. Multi-scale context aggregation by dilated convolutions. In *Internation Conference on Learning Representations (ICLR)*, 2016.

Zagoruyko, S. and Komodakis, N. Wide residual networks. In *British Machine Vision Conference (BMVC)*, 2016.

Zhao, J., Ning, X., Liu, E., Ru, B., Zhou, Z., Zhao, T., Chen, C., Zhang, J., Liao, Q., and Wang, Y. Dynamic ensemble of low-fidelity experts: Mitigating nas "cold-start". *AAAI Conference on Artificial Intelligence (AAAI)*, 2023.

Zhou, D., Zhou, X., Zhang, W., Loy, C. C., Yi, S., Zhang, X., and Ouyang, W. Econas: Finding proxies for economical neural architecture search. In *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition (CVPR)*, 2020.

Zhou, D., Jin, X., Lian, X., Yang, L., Xue, Y., Hou, Q., and Feng, J. AutoSpace: Neural architecture search with less human interference. In *IEEE/CVF International Conference on Computer Vision (ICCV)*, 2021.

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

Zoph, B., Vasudevan, V., Shlens, J., and Le, Q. V. Learning transferable architectures for scalable image recognition. In *Proceedings of the IEEE conference on computer vision and pattern recognition*, pp. 8697–8710, 2018.

A. Pretraining: architectures, hyperparameters and cost

The total pretraining cost is around 30 GPU hours on a single V100 GPU, and the most time-consuming part is running the ES for training the SG (14 GPU hours). On the other hand, training the G-VAE and the CCNF takes only 5.5 GPU hours in total. Note that the cost is paid once for all reported experiments.

Hidden cost. Additionally, to the NAS-specific cost above, our method relies on having access to a trained GPT. Although GPTs have become increasingly available, if a model like that had to be trained from scratch (e.g., due to legal issues), it would incur relatively high additional costs. Specifically, (Brown et al., 2020) report total compute to train GPT-3 125M to be 2.25×10^{20} FLOPs, which translates to approx. 910 GPU hours using a A100 GPU (assuming FP32 with 44% of the peak tensor utilisation, c.f. Tab. 1 by (Narayanan et al., 2021)).

In this section, we furnish an overview of the hyperparameters employed in each component, along with the training parameters utilized during their respective training procedures. We also provide a detailed overview of the cost of different steps of our method in Tab. 5.

The G-VAE is configured with four graph convolution layers (Kipf & Welling, 2017), utilizing a hidden state of size 512 and projecting to an output latent space of size 256 for the metrics predictor. This metrics predictor is structured as a two-layer MLP with ReLU activation. The first layer serves as a hidden state, mirroring the size of the latent space at 256, and the subsequent layer produces four conditions, aligning with our target outcomes.

In terms of training, the metrics predictor undergoes joint training with the G-VAE. The AdamW optimizer (Loshchilov & Hutter, 2019), combined with a cosine annealing schedule (Loshchilov & Hutter, 2017), is employed, initiating with a learning rate of 1e-3 and decaying to a minimum value of 0. A weight decay is also incorporated, set at 5e-4. The training procedure is constrained by a maximum of 500,000 steps and incorporates an early stopping mechanism. This mechanism ceases training if a reduction in loss is not observed over a span of 10 epochs.

The Continuous Conditional Normalizing Flow (CCNF) incorporates nine Concat+Squash+Linear layers, each having a hidden dimension of 512. The input layer is structured to accommodate a latent feature size of 256 and is engineered to project these latent features into a Gaussian distribution, preserving the identical dimensional size of 256.

For training, we utilize 400,000 unlabeled graphs as our dataset. These graphs generate latent features through G-VAE, which are further predicted into ZC vectors by the metrics predictor. The CCNF is trained with a fixed learning rate of 1e-3 and a batch size of 256. Weight decay is set at 0.01, serving as the default value for the Adam optimizer (Kingma & Ba, 2015). The latent features are then transformed into a standard Gaussian distribution, which acts as the prior distribution by minimizing log-likelihood. Notably, the network employs the "dopri5" ODE solver (Hairer et al., 1993) with both absolute and relative tolerance set to 1e-5.

For training the sequence generator, GPT-Neo-125M (Black et al., 2021) is employed as the foundation through pretrained checkpoints. The model consists of 12 transformer blocks (Vaswani et al., 2017) using GELU (Hendrycks & Gimpel, 2016) and LayerNorm (Ba et al., 2016). We finetune this model with a consistent learning rate of 1e-3, utilizing the 60,000 networks and zc-vectors gathered from the HL-Evo procedure and maintaining a batch size of 1. To augment the dataset and mitigate the risk of overfitting, we enhance our condition token by randomly dropping tokens from the set that includes Param, Flops, and ZC values.

For details of our hyperparameters, please refer to Tab. 6.

Table 4. Detailed hyperparameters of each component.

	G-VAE	Predictor	CCNF	Seq. Gen.
Data gen.	——— 5 ———			14
Training	—— 0.5 ——		0.5	11

Table 5. Detailed cost of running each step of our method All cost in GPU hours assuming execution on a V100 GPU.

B. CCNF Sample Experiments Details

The overall training and sampling process is summarized in Fig. 3. We validate the quality of designs generated by the CCNF and compare it to naive sampling in the latent space in Tab. 8. We can see that samples from the CCNF are focused closer to the reference compared to naive sampling of the neighbourhood – this results in almost 1 percentage point higher average accuracy of the sampled models and better best- and worst-case performance.

To demonstrate the rationale of clustering micro designs, we choose to sample 10 graphs from each of the following clusters:

- **cluster 9**: containing graphs with the lowest params/Flops and zc-scores
- **cluster 1** containing graphs with the highest nwot/SNIP-SNR scores
- **cluster 2** containing graphs neighbour to those in cluster 1in zc-vector space with larger param/flops sizes but relatively smaller nwot/SNIP-SNR sco

Table 6. Detailed hyperparameters of each component.

		G-VAE	Metrics Predictor	CCNF	Seq. Gen.
ARCH	Main Operation	GCN	Linear	Cat + Squash + Linear	Transformer
	Number of layers	4	2	9	12
	Input dimension	$[9 \times 9, 15 \times 9]$	256	256	64
	Hidden dimension	512	256	512	2048
	Output dimension	256	4	256	768
	Activation fn	ReLU	ReLU	tanh	GELU
	Normalisation	LayerNorm	-	Moving BatchNorm	LayerNorm
	Prior	Gaussian	-	Gaussian	-
TRAIN	Tr. samples	400K unlabelled + 50k labelled		400k	60k
	Batch size	64		256	1
	Learning rate	3.5e4		1e3	1e3
	Learning rate schedule	CosineAnnealingLR		fixed	fixed
	Weight decay	5e-4		0.01	0.05
	Optimizer	AdamW		Adam	Adam
	Loss	Triplet + BCE + KLD + MSE		log$p$	CE
	Augmentations	-		-	randomly drop condition tokens from set of [param, flops zc]
	Others	Trained jontly		solver='dopri5' atol=1e-5 rtol=1e-5	GPT-neo-125

- **cluster 17**: containing graphs with similar param/flops sizes with those in cluster 1 but significantly lower zc-scores
- **cluster 30**: containing graphs with similar zc scores with those in cluster 2 but significantly small param/flops sizes

The statistical properties of sampled graphs from each cluster are shown in Tab. 7. As we cluster each cluster based on zc-vector space, we also plot the graphs scatter plot in zc-vector space to illustrate their relative information in Figure D. To investigate the relationship between performance and clusters; we train the architecture across three distinct tasks: CIFAR100, Ninapro and DarcyFlow; the statistics are shown as follows:

From the results, we can see that the performance across different clusters has followed a relative relationship learned from zero-cost space, which followed previous research on zero-cost metrics that zero-cost metrics correlate with model performance. Specifically, cluster 1 has better results while neighbour cluster 2 performs relatively similarly over three tasks compared to other apart clusters. Compared with cluster 17, even though graph 17 has a similar cost compared with cluster 1, the performance is significantly different since their NWOT/SNIP-SSNR are different from 794.68 compared to cluster 1 at 918.51 on NWOT while also for SNIP SSNR cluster 1 has 259.23 compared clusters average at 98.27. As for cluster 30, even though it has a much smaller cost since its NWOT/SNIP-SSNR is relatively higher than cluster 17, it has a slightly better average performance than cluster 17. A significantly worse cluster is cluster 9, with the smallest param/flops and smallest ZC-scores, the performance ranked bottom overall clusters we have selected.

C. Operations included in the design sapce

This section introduces the foundational operations incorporated in our graph design space. Our design space consists of nine basic operations, listed as follows:

- **avg_pool**: Average Pooling is a downsampling technique utilized to reduce the spatial dimensions of a feature map. It calculates the average value of the elements within a specified local region determined by the kernel size and stride.
- **conv**: The Convolutional Layer starts with a ReLU preactivation and ends with a batch normalization (Ioffe & Szegedy, 2015) layer. This operation is pivotal for learning spatial hierarchies of features and is widely employed in deep learning models.
- **sep_conv** (Niklaus et al., 2017): Separable Convolution is an efficient variant of the standard convolution. It factorizes a standard convolution into a depthwise spatial convolution followed by a pointwise convolution, reducing the computational costs.
- **dil_conv** (Yu & Koltun, 2016): Dilated Convolution introduces gaps to the convolutional kernel, effectively

Table 7. Experiments on Sampling Overall Different Clusters and Trained Diverse Tasks

Cluster	Params (1e6)	Flops (1e9)	SNIP-SNR	NWOT	C100	NinaPro	DarcyFlow
9	0.09 ± 0.09	0.02 ± 0.03	3.84 ± 0.79	712.9 ± 81.27	45.33 ± 10.46	70.13 ± 8.37	0.098 ± 0.017
1	1.44 ± 0.18	0.47 ± 0.06	259.23 ± 12.09	918.51 ± 8.58	75.33 ± 1.56	92.37 ± 1.14	0.014 ± 0.005
2	1.99 ± 0.2	0.59 ± 0.05	190.16 ± 37.5	864.17 ± 15.54	71.13 ± 2.04	91.23 ± 1.55	0.027 ± 0.006
17	1.59 ± 0.18	0.46 ± 0.05	98.27 ± 20.88	794.68 ± 39.61	59.47 ± 11.34	77.33 ± 9.14	0.056 ± 0.019
30	0.153 ± 0.05	0.05 ± 0.03	129.57 ± 42.36	797.32 ± 21.8	63.56 ± 15.19	79.15 ± 10.77	0.051 ± 0.021

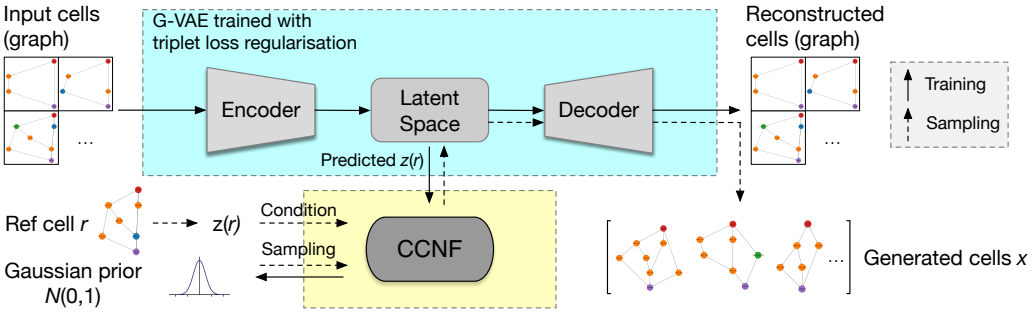

Figure 3. Training and sampling processes of the proposed G-VAE and CCNF models.

increasing the receptive field without increasing the number of parameters, making it suitable for tasks requiring the incorporation of larger contextual information.

- **mbconv** (Yu & Koltun, 2016): The MobileNetV2-inverted bottleneck convolution has three variants with expand sizes at [2, 3, 6]. It employs an inverted residual structure, aiding in building efficient and compact models suitable for mobile applications.
- **bconv** (He et al., 2016): The Bottleneck Convolution comes in two types with shrink rates in the bottleneck at [0.5, 0.25]. It utilizes a bottleneck structure to reduce dimensionality, thereby decreasing computational requirements.

In addition to the basic operation types introduced above, we also offer three choices of kernel sizes – $\{1, 3, 5\}$ – for each operation. This variety ensures coverage of the most prevalent network design paradigms, allowing for versatile adaptability and exploration within our graph design space.

D. Code

Code will be available upon acceptance.

E. More experiments

F. ImageNet-1k Architectures and Micro-design cluster Proposed

This section presents architectures for the ImageNet-1k task under different Flops constraints. Figure 4 illustrates the architecture searched with a 450M flops constraint, while Figure 5 depicts the architecture searched with a 600M flops constraint. Additionally, Figure 6 presents the cluster proposed by the SG.

F.1. Alternative final search algorithm

As mentioned at the end of Section 4, after our method generates a small number of promising models, there remains the question of the final selection. Although we opted for simple selection using T-CET in our main experiments, other approaches are also possible, although they are likely to increase the overall cost. To further explore this direction, we also include an approach of running a typical baseline — evolutionary search with reduced-training proxy (Zhou et al., 2020). For this purpose, we utilised training scheme similar to the one used by NATS-Bench-201 (Dong et al., 2021), with batch size 256 and 50 epochs on CIFAR-100. Using this scheme, training a single model takes approx. 0.5h, so we are able to train 40 models for a common budget of 20h presented in Tab. 1. Given the budget of 40 models, we run evolution using a population size of 8, to allow for a reasonable number of mutations. We run this baseline on the optimised search space obtained from generating 5 models from the same macro sequence and mixing different instances of cells between the 5 models. In our experiment, this resulted in the space of 5^4 models. The best model according to the reduced-training proxy was trained fully using the same training procedure as the rest of methods in Tab. 1. The final performance turned out to be 97.3 — this is somewhat noticeably lower than selection with T-CET (97.6), especially considering much higher searching cost, but also much higher compared to using a similar reduced-

Table 8. Comparison of different approaches of sampling the G-VAE latent space. Neighbourhood and CCNF are reference-based (see Sec. 4.2) and we test them by giving the best cell from NATS-Bench-TSS (Dong et al., 2021) as a reference. All macro models constructed using α_x from Eq. 1.

Sampling method	Gen. success	ZC vec.				C-10 Acc.		
		NASWOT	SNIP	Params. (M)	FLOPs (M)	Min	Mean	Max
Random	9/20	793$\pm$49	104$\pm$64	0.3$\pm$0.3	111$\pm$113	10.00	75.00$\pm$28.37	93.86
NATS-Bench ref.		783	80	1.15	327		94.24	
Neighbourhood	20/20	793$\pm$49	92$\pm$29	0.7$\pm$0.3	220$\pm$86	89.76	92.15$\pm$1.13	94.13
CCNF	20/20	788$\pm$10	78$\pm$13	0.6$\pm$0.3	189$\pm$100	91.89	93.08$\pm$0.85	94.55

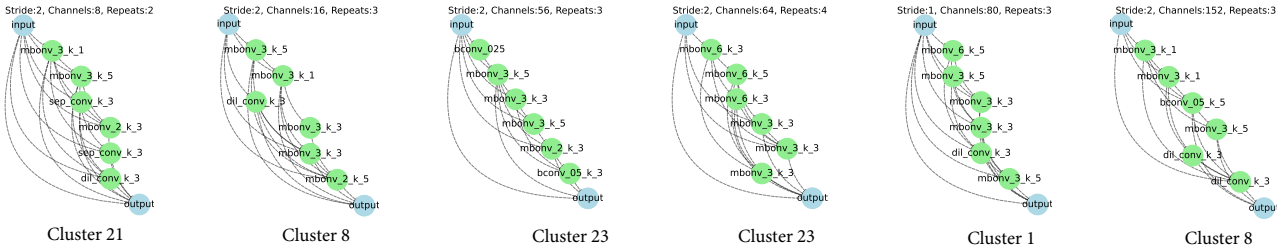

Figure 4. Architeture searched on ImageNet-1k task under 450M flops constraint

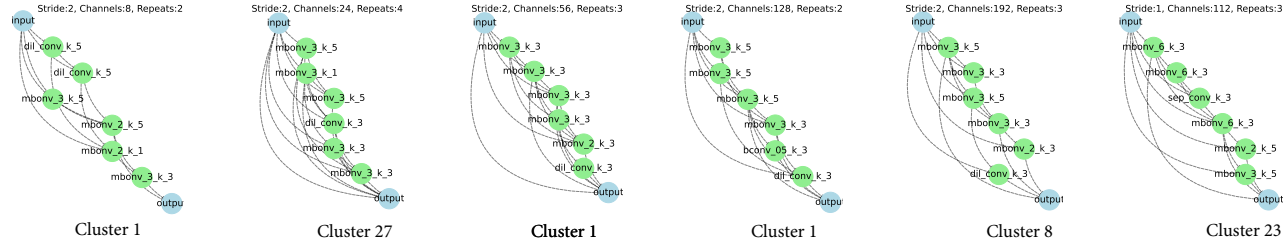

Figure 5. Architeture searched on ImageNet-1k task under 600M Flops constraint

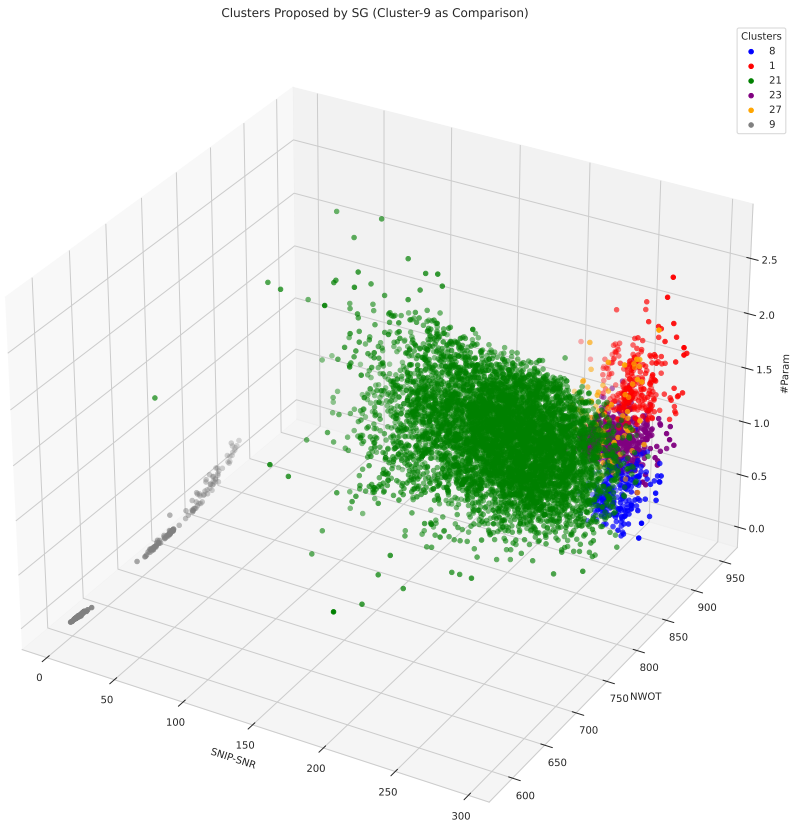

Figure 6. Cluster that Proposed by the SG

training proxy directly on the large GraphNet space (see below). In general, the results suggest our SG+CCNF can indeed help a searching a lot by shortlisting promising architectures, although it is probably not the best idea to use reduced training to select the final model.

F.2. More baselines on the big GraphNet space

In section 5 we show that Evo(T-CET) on the GraphNet space achieves inferior performance. But what about other search strategies? Here, we again include a common baseline of running a standard evolutionary algorithm with the same reduced-training proxy as above. Given the same searching budget of around 20 GPU hours, this baseline achieved an accuracy of 90.32, a significant shortcoming compared to 96.4 of Evo(T-CET) and 97.6 of our SG+CCNF. This showcases the challenges of using a popular off-the-shelf black-box optimiser in very large search spaces like ours.

G. Survey on Other Search Space and Typical NAS Cost

Table.9 compares diverse NAS methods based on their search space characteristics, including size and type, typical optimization techniques employed, evaluation strategies, the scale of model evaluations, and the associated computational expense measured in GPU hours. GraphNet, our proposed method, is highlighted, which introduces a novel hierarchical search space combining sequential and graph-based elements, illustrating its unprecedented scale and complexity.

H. Limitations, discussion and future work

Reliance on ZC proxies. Our method heavily relies on ZC proxies – we do that in order to organise a large search space like ours in a meaningful way without incurring a prohibitively large comp. cost; to the best of our knowledge, there is no better alternative at the moment. Having said that, ZC proxies are known to not always produce faithful scoring of networks (White et al., 2021b; Ning et al., 2021). Although we try to avoid the most common failure cases by relying on clustering rather than strict ordering, our assumptions might still not generally hold. On the other hand, due to the paradigm shift in our work, we show a new direction for future research on ZC proxies focused on relative performance – our work shows that this approach can be more powerful even in settings where existing proxies are known to work well (e.g., CIFAR). Another possible direction would be to incorporate a feedback mechanism to adjust network generation online efficiently.

Conditioning. Currently, microcells are generated based on cluster centres – this is simple and shown to work well in our case, but in the end, it leaves room for improvement. The method could also be extended to include more explicit conditioning on the target task (currently #FLOPs and #params).

On-device performance. On-device performance is important for the practical deployment of models, but we did not consider it a direct objective for simplicity, instead relying on device-agnostic metrics such as a number of parameters and FLOPs. Extending our work to be able to design networks optimized for a particular device is a very sensible direction for future work.

Transformers. Although transformers keep playing an increasingly important role, for the time being, we do not consider them in our search space. Extending our codebase to support hybrid convolution-attention models is an exciting direction for future work.

Table 9. Comparative Overview of Neural Architecture Search Methods.

SearchSpace	Search Space type	SearchSpace Size	Typical Methods	Evaluation Types	Model Evaluated	GPU-Cost(GPU-hours)
Depict Network Topologies-39 layers (Zoph & Le, 2017)	Graph	1.12e89	Reinforce Learning	Trained	12800	537600
NASNet-A(Zoph et al., 2018)	Graph	1.99e28	Reinforce Learning	Trained	20,000	2000
DARTS (Liu et al., 2019a)	Graph	1e19	Differentiable	Trained	None	96
ZenNet-16 layers (Lin et al., 2021)	Sequential	9.6e22	Aging Evolution	Zero-cost NAS	480000	12
GraphNet(our)	Hierachical(Sequential + Graph)	$\sim$1e390	/	/	/	/