Building Optimal Neural Architectures using Interpretable Knowledge

Keith G. Mills[1,2] Fred X. Han[2] Mohammad Salameh[2] Shengyao Lu[1]
Chunhua Zhou[3] Jiao He[3] Fengyu Sun[3] Di Niu[1]
[1]Dept. ECE, University of Alberta [2]Huawei Technologies Canada [3]Huawei Kirin Solution, China
{kgmills, shengyao, dniu}@ualberta.ca sunfengyu@hisilicon.com
{fred.xuefei.han1, mohammad.salameh, zhouchunhua, hejiao4}@huawei.com

Abstract

Neural Architecture Search is a costly practice. The fact that a search space can span a vast number of design choices with each architecture evaluation taking nontrivial overhead makes it hard for an algorithm to sufficiently explore candidate networks. In this paper, we propose AutoBuild, a scheme which learns to align the latent embeddings of operations and architecture modules with the ground-truth performance of the architectures they appear in. By doing so, AutoBuild is capable of assigning interpretable importance scores to architecture modules, such as individual operation features and larger macro operation sequences such that high-performance neural networks can be constructed without any need for search. Through experiments performed on state-of-the-art image classification, segmentation, and Stable Diffusion models, we show that by mining a relatively small set of evaluated architectures, AutoBuild can learn to build high-quality architectures directly or help to reduce search space to focus on relevant areas, finding better architectures that outperform both the original labeled ones and ones found by search baselines. Code available at https://github.com/Ascend-Research/AutoBuild

1. Introduction

Neural Architecture Search (NAS) is an AutoML technique for neural network design and has been widely adopted in computer vision. Given a task, most NAS methods involve exploring a vast search space of candidate architectures, e.g., [6, 24, 31, 41], using a search algorithm, to find the best architecture in terms of predefined performance and hardware-friendly metrics. However, NAS is a costly procedure mainly due to two reasons. First, the search space is formed by varying the operation configurations in each layer and is thus exponential to the number of configurations per layer in scale, which can easily exceed 10^{10} architectures [4, 19], making it hard to thoroughly explore the search space. Second, there is a large cost associated with each architecture evaluation, which requires full training before assessing performance on a validation set.

Several methods have been proposed in the literature to reduce the evaluation cost in NAS. For example, Once-for-All (OFA) [4] trains a weight-sharing supernetwork that can represent any architecture in the search space. Recently, SnapFusion [16], which is adopted by Qualcomm's latest Snapdragon 8 Gen 3 Mobile Platform, applies a robust training procedure to Stable Diffusion models, so that the U-Net is trained to be robust to architectural permutations and block pruning, which enables it to find a pruned efficient U-Net that speeds up text-to-image generation significantly. In fact, these methods still pay an initial, very high one-time computational cost to train the supernetworks or robust U-Net (which requires a tremendous investment into GPUs), such that during search individual candidate architectures can be assessed by pulling weights from the supernetwork. Another limitation is that supernetwork training is specific to the dataset used, e.g., ImageNet [7], and retraining may incur a significant cost. As foundational models [16, 29, 33] grow in complexity, NAS is becoming infeasible for organizations with limited computing resources.

In this paper, we propose AutoBuild, a method for automatically constructing optimal neural architectures from high-valued architecture building blocks discovered from only a handful of random architectures that have been evaluated. Unlike traditional NAS that considers how to best traverse a vast search space, AutoBuild focuses on discovering the features, operations and multi-layered subgraphs as well as their configurations–in general, interpretable knowledge–that are important to performance in order to directly construct architectures. The intuition is that although the size of the search space is exponential large, the actual number of configurations per layer can be small. We ask the question–could there be a method that learns how to build a good architecture rather than searching for one, thus circumventing the formidable search cost?

Specifically, we learn the importance of architecture

modules using a ranking loss mechanism applied to each hop-level of a Graph Neural Network (GNN) [3, 40, 45] predictor to align the learnt embeddings for each subgraph or node with the ground-truth predictor targets (e.g., accuracy). This ensures important subgraphs and nodes will have higher embedding norms (by the GNN) if they appear in architectures with higher performance. We propose methods to be able to rank operation-level features, e.g., convolution kernel size vs. input/output latent tensor size for a CNN, or rank operations, e.g., attention vs. residual convolutions in a diffusion model, or rank subgraphs of operation sequences. AutoBuild can then construct high-quality architectures by combining high valued modules in each layer or can greatly prune the size of original search space prior to performing NAS. Moreover, we can readily manipulate predictor labels to combine multiple performance metrics, e.g., both accuracy and latency, to focus on different regions of the search space.

Through extensive experiments performed on state-of-the-art image classification, segmentation, and Stable Diffusion models, we show that by mining a relatively small set of evaluated architectures, AutoBuild can learn to assign importance 'scores' to the building blocks such that a high-quality architecture can be directly constructed without search or searched for in a much reduced search space to generate better architectures.

To prove the effectiveness of AutoBuild, we show that by learning from 3000 labeled architectures for ImageNet, AutoBuild can focus on shortlisted regions of the vast search space of MobileNetV3 to generate superior Pareto frontier of accuracy vs. latency compared to other schemes that search the whole search space or reduced search space based on manual insights. A similar trend holds for panoptic segmentation [15]. Furthermore, in a search space consisting of variants of Stable Diffusion v1.4 [33] by tweaking the configurations of each layer of its U-Net, where we have access to only 68 labeled architectures, AutoBuild can quickly generate a architecture without search that achieves better image Inpainting performance as compared to the labeled architectures or a predictor-based exhaustive search strategy.

2. Related Work

Search Spaces fall into two categories: Macro-level and micro-level. Macro-level spaces represent architectures as a sequence of predefined structures, e.g., MBConv blocks in MobileNets [6, 12, 13, 36, 41], or even Attention vs. Convolution layers in Stable Diffusion U-Nets [32, 33]. Representing the network as a sequence implicitly encodes some level of spatial information, such as latent tensor channel size and stride, which can change across the depth of the network. By contrast, micro-level spaces, like many NAS-Benchmarks [24], designs the search space over a cell structure that is repeated multiple times to form the entire network. The cell takes the form of different DAGs over a predefined list of operations and typically do not encode spatial information. Although many of the techniques in AutoBuild apply to both macro and micro-search spaces, we focus on macro-level spaces like MobileNets and Stable Diffusion U-Nets in this paper.

Neural performance predictors [22, 26, 43] learn to estimate the performance of candidate architectures and overcome the computational burden of training them from scratch, in order to facilitate more lightweight NAS. By contrast, AutoBuild focuses on learning in a way that quantifies the importance of architecture modules. AutoBuild further leverages these insights to reduce the overall search space size, and thus, the necessity to perform NAS.

Interpretable approaches generate insights on the relationship between search spaces and performance metrics. For example, NAS-BOWL [34] uses Weisfeiler-Lehman (WL) kernels to extract architecture motifs. However, NAS-BOWL mainly operates on micro search spaces as the WL-kernel does not efficiently scale with graph depth [35]. Sampling-based approaches [25] examine how architecture modules impact accuracy and latency. However, these methods rely on large amounts of data to generate statistical information. In contrast, AutoBuild is more data-driven and efficient - capable of learning insights from limited data.

Ranking losses have previously been used to structure GNN embedding spaces. CT-NAS [5] use a ranking loss to determine which architecture in a pair has higher performance. AutoBuild uses a Magnitude Ranked Embedding Space to generate importance scores, which is partially inspired by the Ordered Embedding Space that SPMiner [46] construct to perform frequent subgraph mining.

3. Background

This section provides background information on how architectures can be cast as graphs. Further, we elaborate on how node embeddings produced by GNNs are used to 'score' their corresponding rooted subgraphs.

3.1. Architecture Graph Encodings

A common way to represent candidate architectures for NAS is as a Directed Acyclic Graph (DAG). A DAG is a graph $\mathcal{G}$ with a node (vertex) set $\mathcal{V}_\mathcal{G}$, which is connected by a corresponding edge set $\mathcal{E}_\mathcal{G}$. These edges define the forward pass in the graph. The representation of a node differs based on the structure and design of the search space. Fine-grained methods like AIO-P [27] and AutoGO [35] use an Intermediate Representation (IR) extracted from frameworks like ONNX [1], where nodes represent single primitive operations like convolutions, adds and activations. On the other hand, cell-based NAS-Benchmarks [24] use nodes or edges to represent predefined operation sequences.

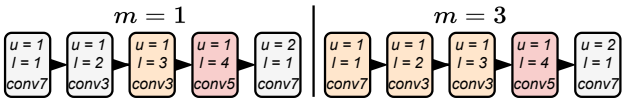


Figure 1. Visualization of a sequence-like architecture DAG. Nodes annotated with position and layer information: 'u' and 'l' refer to stage and layer position, while 'conv' refers to the type of layer. **Left graph:** Subgraph rooted at the red node induced by a 1-hop message passing layer. **Right graph:** Additional (orange) nodes incorporated into the subgraph for a 3-hop layer.

The coarsest node representation resides in macro-search spaces. Nodes could represent a whole residual convolution layer [10], a transformer block with multi-head attention [33, 39], or MBConv [6, 12, 13, 36, 38, 41] block structure. Nodes are stacked in a 1-dimensional sequence to form a network.

We primarily focus on macro-search sequence graphs in this paper. Generally, a macro-search space contains two tiers of granularity: Stages (also known as Units [4, 25]) and Layers. Architectures in a search space contain a fixed number of stages, each assigned a specific latent tensor Height-Width-Channel (HWC) dimension. Stages could contain a variable number of layers. Each layer constitutes a node that defines the specific type of computation such as convolution vs. attention in Stable Diffusion v1.4 or kernel size/channel expansion ratio if the layer is an MBConv block.

Each stage is denoted as a subgraph of its layers. The maximum number of unique subgraphs for a given stage depends on the number of layers it can contain as well as the set of layer type combinations that are permitted. Therefore, the size of the overall search space depends on the number of stages as well as the number of possible subgraphs for each stage. We provide an illustration of a sequence-like graph for a macro-search space in Figure 1, showing how nodes can be labeled with stage/layer positional features.

3.2. Induced Subgraph Embeddings

Graph Neural Networks (GNN) [47] facilitate message passing between nodes using a graph's adjacency matrix. Each GNN layer represents a 'hop' - where each node receives information from all immediate nodes in its neighborhood. Given node v, let $\mathcal{N}(v) = \{s \in \mathcal{V}_\mathcal{G} | (s, v) \in \mathcal{E}_\mathcal{G}\}$ denote the neighborhood of v. Furthermore, let $h_v^m \in \mathbb{R}^{1\times d}$, where d is the embedding vector length, represent the latent representation of v at layer m. Then, the message passing function [45] of a GNN layer m can be defined as:

$$h_v^m = \texttt{Combine}(\texttt{Agg}(h_v^{m-1}, \{h_s^{m-1} : s \in \mathcal{N}(v)\}) \quad (1)$$

where the `Combine` and `Agg` (Aggregate) functions are determined by GNN type, e.g., GAT [3, 40], GCN [42], etc.

With each successive GNN layer, the receptive field of v grows as features from more distant nodes are aggregated into h_v^m. As such, instead of interpreting h_v^m as the 'embedding of node v at layer m', we extend h_v^m to denote the 'embedding of the m-hop rooted-subgraph induced by node v'. Figure 1 illustrates how the subgraph encompasses information from all nodes within an m-hop distance of v.

Graph Aggregation and Prediction. In graph-level prediction problems, the GNN produces a single prediction per graph. Since graphs can have varying number of nodes and edges, this requires condensing the node embeddings from a message passing layer $H_{\mathcal{V}_\mathcal{G}}^m \in \mathbb{R}^{|\mathcal{V}_\mathcal{G}|\times d}$ into a single, fixed-length embedding representing the *whole* graph, $h_\mathcal{G}^m \in \mathbb{R}^{1\times d}$. Typically, this is performed by applying an arithmetic operation like 'mean' [47],

$$h_\mathcal{G}^m = \frac{1}{|\mathcal{V}_\mathcal{G}|} \sum_{v\in\mathcal{V}_\mathcal{G}} h_v^m. \quad (2)$$

A GNN can contain multiple layers $m \in [0, M]$. Usually, a single graph embedding is calculated from the output of the final message passing layer M, which is then fed into a simple Multi-Layer Perceptron (MLP) to produce a prediction, $y'_\mathcal{G} = \texttt{MLP}(h_\mathcal{G}^M)$. However, it is possible to calculate subgraph embeddings at *any* hop-level, e.g., $m < M$, including $m = 0$. In fact, it is this capability which allows AutoBuild to 'score' and compare subgraphs of different sizes as well as node feature categories, as we will next discuss.

4. Methodology

In this section, we elaborate on the design of AutoBuild, specifically the hop-level correlation loss, and how we can use a Magnitude Ranked Embedding Space to compare subgraphs induced by different hop levels. Additionally, we elaborate on our Feature Embedding MLP (FE-MLP).

4.1. Magnitude Ranked Embedding Space

Graph regressors learn to estimate their target values by minimizing a loss, such as the Mean-Squared Error (MSE) or Mean Absolute Error (MAE) between its predictions $y'_\mathcal{G}$ and targets $y_\mathcal{G}$. However, this approach does not impose any constraint upon the latent embedding space which makes the predictor a black box where estimations are hard to interpret. On the other hand, AutoBuild constrains the embedding space by associating high-performance subgraph stages with high scores, making it easier to interpret results.

Let $\mathcal{G}_1$ and $\mathcal{G}_2$ be two separate architecture graphs in a training set. In addition to a standard regression loss between labels and targets, AutoBuild incorporates an additional constraint into the learning process:

$$\forall m \in [0, M] \mid \texttt{if } y_{\mathcal{G}_1} > y_{\mathcal{G}_2}, \texttt{then } \left\|h_{\mathcal{G}_1}^m\right\|_1 > \left\|h_{\mathcal{G}_2}^m\right\|_1. \quad (3)$$

This constraint ensures that if the ground-truth label for $\mathcal{G}_1$ is higher than $\mathcal{G}_2$, then the norm of $\mathcal{G}_1$'s graph embedding

should be higher than that of $\mathcal{G}_2$ across all hop levels. In practice, we impose this constraint by augmenting the original regression loss with an additional ranking loss,

$$\mathcal{L} = \texttt{MSE}(y_\mathcal{G}, y'_\mathcal{G}) + \frac{1}{M+1}\sum_{m=0}^{M}(1 - \rho(y_\mathcal{G}, \left\|h^m_\mathcal{G}\right\|_1)), \quad (4)$$

where ρ is the Spearman's Rank Correlation Coefficient (SRCC). The value of $\rho = 1$ indicates perfect positive correlation, while $\rho = -1$ indicates negative correlation and $\rho = 0$ denotes no correlation. If the graph embedding norms are perfectly aligned with the targets, $\rho(y_\mathcal{G}, \left\|h^m_\mathcal{G}\right\|_1) = 1$, the loss term will be zeroed. Typically, ranking metrics like SRCC are generally considered non-differentiable and are therefore not usable as loss functions. However, we utilize the method of Blondel et al., 2020 [2], which provides an exact, differentiable computation to calculate SRCC across an entire batch of data.

Equation 4 forces the GNN to learn embeddings in a Magnitude Ranked Embedding Space - where graph embedding norms are correlated with ground-truth targets. Moreover, since graph embeddings $h^m_\mathcal{G}$ are directly computed from node embeddings h^m_v per Equation 2, the GNN is forced to learn which nodes can be attributed to these more prominent labels and subsequently assign them larger node embeddings. Thus, the norm of the node embedding, $\|h^m_v\|_1$, acts as a learned numerical 'score' for the node v, and by extension, its induced m-hop subgraph.

We further note that the node/subgraph scores do not require additional architecture annotations, e.g., explicit ground-truth subgraph labels. Rather subgraph scores are implicitly learned from the training data. Like other neural performance predictors [43], the only ground-truth architecture annotations AutoBuild requires to compute Eq. 4 is $y_\mathcal{G}$, the scalar label for architecture $\mathcal{G}$. $y_\mathcal{G}$ is typically an end-to-end architecture metric like accuracy, inference latency, or a combination thereof, as we demonstrate in Sec. 5.

4.2. Comparing Subgraphs at Different Hops

We can interpret the embedding norm $\|h^m_v\|_1$ as the score for the m-hop subgraph rooted at node v. However, this score is only comparable between subgraphs of the same hop level. While Equation 4 ensures ranking correlation between embedding norms and the ground-truth labels, the distribution of embedding norms can vary *between* hop levels. Therefore, additional computation is necessary to compare subgraphs of different sizes. We propose a mechanism that works in the case of macro-search spaces where architectures can consist of stages (Sec 3.1), since each stage can consist of one predefined subgraph.

Let μ^h_m and σ^h_m refer to the mean and standard deviation of node embedding norms produced by the GNN at hop-level m. Also, let $\mu^y_{u,l}$ and $\sigma^y_{u,l}$ refer to the mean and standard deviation of ground-truth labels for architectures that have l layers ($l = m+1$). To compare an arbitrary node embedding norm $\|h^m_v\|_1$ with a score from another hop-level, we shift it from $\mathcal{N}(\mu^h_m, \sigma^h_m)$ to $\mathcal{N}(\mu^y_{u,l}, \sigma^y_{u,l})$ as follows:

$$\left\|h^l_v\right\|^*_1 = (\|h^m_v\|_1 - \mu^h_m) * \frac{\sigma^y_{u,l}}{\sigma^h_m} + \mu^y_{u,l}, \quad (5)$$

where $\left\|h^l_v\right\|^*_1$ is the shifted embedding norm. Likewise, in order to compare this subgraph to another one at a different hop-level, e.g., $\left\|h^{m+1}_v\right\|_1$, we would in turn shift that score using $\mathcal{N}(\mu^h_{m+1}, \sigma^h_{m+1})$ and $\mathcal{N}(\mu^y_{u,l+1}, \sigma^y_{u,l+1})$ instead.

This distribution shift aligns the subgraph scores with the biases of the target data. While we could standardize all scores into a normal distribution $\mathcal{N}(0, 1)$, such a procedure assumes that subgraphs at different hop levels should be weighed equally. For example, standardizing prior to comparison assumes that the average l-hop subgraph should receive the same score as the average $l+1$-hop subgraph. However, if the GNN is an accuracy or latency predictor, using a bigger subgraph at a given stage means the overall architecture is larger, which on average entails higher accuracy and latency [25]. As such, we would expect $\mu^y_{u,l+1}$ to exceed $\mu^y_{u,l}$. We can interpret the difference $\mu^y_{u,l+1} - \mu^y_{u,l}$ as the performance *bias* an $l+1$-sized stage subgraph enjoys, and that a smaller l-hop stage subgraph must overcome, through superior combination of layers, to obtain a higher score. Therefore, we first standardize using $\mathcal{N}(\mu^h_m, \sigma^h_m)$, which is computed by passing the graph dataset over the GNN after training is complete. We then unstandardize using $\mathcal{N}(\mu^y_{u,l}, \sigma^y_{u,l})$, which can be statistically computed from the training set.

4.3. Ranking Individual Node Features

The first component of a GNN is typically the embedding layer, $m = 0$, which translates the original, human-readable features of graph nodes into numerical vectors that machine-learning models can better understand. The nodes can have various types of features, such as discrete operation types like 'convolution', or numerical vectors for tensor shapes. Positional encodings like the stage u and layer l are also considered. The GNN embedding layer is responsible for refining these different feature categories into a single continuous vector, $h^0_v \in \mathbb{R}^{1\times d}_+$.

AutoBuild learns to estimate the importance of individual node feature categories by incorporating a Feature Embedding Multi-Layer Perception (FE-MLP). The design of the FE-MLP is straightforward: each node feature category is assigned a separate feed-forward module that processes it into a continuous scalar. We then concatenate the scalar values for all feature categories, and apply an `abs` nonlinearity to form h^0_v, the '0-hop' node embedding.

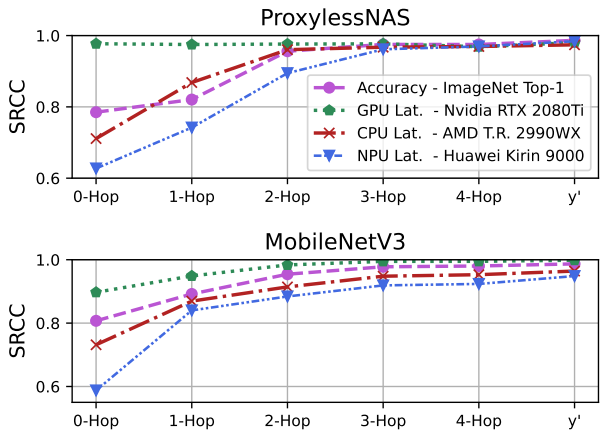


Figure 2. Test SRCC for PN and MBv3 test sets. Specifically, we train AutoBuild predictors using Eq. 4 for accuracy or inference latency on several hardware devices, then measure SRCC for every hop level $m \in [0, 4]$ and the predictor MLP (y').

Although h_v^0 consists of scalars from all feature categories, they do not interact with each other. By coupling this design choice with the hop-level ranking loss in Equation 4, the FE-MLP can identify which feature choices should be assigned high importance scores without any additional information about the given node. This allows us to easily determine the importance of each node feature choice once training is complete, without having to exhaustively evaluate every possible node feature combination.

5. Results

We gauge the effectiveness of AutoBuild in several domains. First, we find Pareto-optimal architectures on two ImageNet-based macro-search spaces from OFA. Then, we apply AutoBuild to two domains where we have access to a limited amount of labeled architectures: Panoptic Segmentation for MS-COCO [18] and Inpainting using SDv1.4. For all experiments, we provide a full description of the training setup and hyperparameters in the supplementary materials.

5.1. ImageNet Macro-Search Spaces

Our OFA macro-search spaces are ProxylessNAS (PN) and MobileNetV3 (MBv3). Specifically, each search space follows the stage-layer layout described in Sec 3.1. Both PN and MBv3 contain over 10^{19} architectures and use MBConv blocks as layers. There are 9 types with kernel sizes $k \in \{3, 5, 7\}$ and channel expansion ratio $e \in \{3, 4, 6\}$. We further enrich the node features by two position-based node features that encode a node stage u and layer l location. MBv3 contains 5 stages each with 2-4 layers. Thus, the total number of architecture module subgraphs for MBv3 is $9^2 + 9^3 + 9^4 = 7.4k$. PN has 9 more modules as it includes a final 6th stage that always has 1 layer.

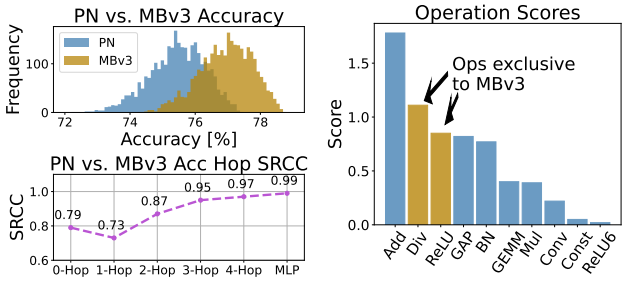


Figure 3. Results of training an accuracy predictor for PN and MBv3 using the 'IR' representation format. **Top-left**: Accuracy histogram. **Bottom-left**: Hop-wise AutoBuild test SRCC. **Right**: Operation-type importance scores from the 0-hop FE-MLP. Note: 'GAP' means Global Average Pool and 'BN' means Batch Norm.

To obtain accuracy labels, we sample and evaluate $3k$ architectures from each search space. Additionally, we obtain inference latency measurements from [25] for three devices: RTX 2080 Ti GPU, AMD Threadripper 2990WX CPU, and Huawei Kirin 9000 Mobile NPU [17].

We gauge the effectiveness of Equation 4 by training GNN predictors and then measuring the test set SRCC of the graph embeddings at each hop-level. For the training/testing data split, we use a ratio of 90%/10% and a batch size of 128. Figure 2 plots the SRCC values across each search space and target metric. We observe strong positive SRCC values across all hop levels for every search space and target metric. This demonstrates the feasibility of learning a magnitude-ranked embedding space using Eq. 4.

Cross-Search Space Comparison. To illustrate how the FE-MLP can assign human-interpretable importance scores to node-level features, we train an accuracy predictor for PN and MBv3 using the finer ONNX IR graph format mentioned in Sec. 3.1. Unlike sequence graphs, the IR format represents operation primitives such as conv or ReLU as nodes while operation type is a categorical node feature.

We present our findings in Figure 3. The 0-hop SRCC of the predictor is 0.79 so FE-MLP embedding norms are correlated with accuracy. First, note MBv3 has a higher accuracy distribution than PN. The FE-MLP captures this distinction, by assigning high importance to 'Division' and 'ReLU' operation nodes, which are part of the *h-swish* activation [12] and Squeeze-and-Excite module (SE) [14] exclusive to MBv3 since PN only uses the 'ReLU6' activation. Additionally, the operation with the highest FE-MLP score is the 'Add', due to its multi-input nature and the frequent use of long skip-connects. We provide a detailed analysis of this phenomenon in the supplementary material.

Architecture Construction. Next, we demonstrate how module subgraph-scoring can help us construct a small handful of high-efficiency architectures. Specifically, we build architectures that outperform the accuracy-latency

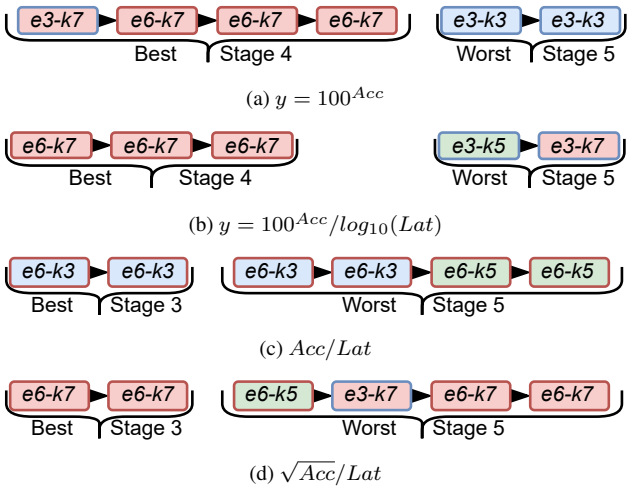


Figure 4. Best and worst MobileNetV3 MBConv layer subgraphs found by AutoBuild using different target equations. Specifically, we illustrate the best and worst subgraphs, by AutoBuild module score, when different equations are used to compute the predictor targets using accuracy and GPU latency.

Pareto frontier formed by the $3k$ random architectures we use to train and test our predictors.

First, we plot the accuracy vs. latency Pareto frontiers for existing architectures. Then, using these architectures, we train several AutoBuild predictors using different target label equations that combine both accuracy and latency. These equations aim to optimize different regions of the dataset Pareto frontier by emphasizing accuracy and latency to different degrees, e.g., $y = {}^{acc}/_{lat}$, $y = acc - lat$, etc.

We then compute the AutoBuild score for each architecture module subgraph in the search space. Figure 4 illustrates the best and worst subgraphs for MBv3 on GPU latency for different label equations. These results are intuitive, e.g., for $y = 100^{Acc}$, the best subgraph has 4 layers (the maximum), convolving over many channels ($e6$) using the largest possible kernel size ($k7$), while the worst subgraph is small with few channels and low kernel size. When latency is factored in, the best subgraphs contain fewer layers yet maintain high expansion e values, while bad subgraphs are long sequences containing inefficient blocks. These results align with [25] who found that MBv3 GPU latency was primarily driven by the number of layers per stage rather than layer type.

Next, using the AutoBuild scores, we iterate across each stage u and select the top-K subgraphs with the highest scores, forming a reduced search space with size $K^{|u|}$. Individual architectures that optimize the predictor target can be quickly constructed by sampling one module subgraph per stage. If K is small, e.g., $K = 3$ while $|u| = 5$, then $K^{|u|} = 243$. Using OFA [4] supernetworks, a search space of this size can be exhaustively evaluated in a few hours on a single GPU. Afterwards, we plot the accuracy and latency of the constructed architectures against the existing architecture Pareto frontier.

The top row of Figure 5 illustrates our findings for MBv3 and PN. On MBv3, PN-GPU and PN-NPU we consistently craft target equations that can focus on regions that contain architectures beyond the Pareto frontier. On MBv3-CPU, MBv3-NPU and PN-NPU, we achieve better performance then architectures on the high-accuracy tail of the frontier using significantly less latency. On PN-CPU, only two of the four reduced search spaces outperform the data Pareto frontier. We note that our equations are manually inferred from the data which is a limitation. Automating the equation-creation process is a future research direction.

Enhancing Neural Architecture Search. AutoBuild is not limited to simply constructing a handful of specialized architectures. By combining the top-K architecture module subgraphs from predictors trained using different target equations, we can craft a reduced search space. Further, we can then use a NAS algorithm to find a Pareto frontier of high-efficiency architectures.

Specifically, for target equations in the top row of Fig. 5, we set $K = 25$, then combine the set of module subgraphs selected for each stage to form our search space. We perform NAS using a multi-objective evolutionary search algorithm [20, 23, 37] which performs mutation by randomly swapping unit subgraphs. We denote our approach as **AutoBuild-NAS** and compare it to several baselines:

- **Stage-Full** uses the whole search space. This baseline performs mutation by changing one of the stage subgraphs in an architecture.
- **Layer-Full**: Similar to Stage-Full with a different mutation mechanism that adds, removes or edits a single layer at a time, instead of changing whole stage subgraphs.
- **Layer-Insight**: Layer-Full mutation using the reduced search spaces of [25], who generate insights by statistically sampling the impact of each *stage-layer* combination using many architectures. Manual, human-expert knowledge is applied to restrict layer choice and the maximum number of layers per stage.

We set a search budget of 250 architectures. In the supplementary materials, we provide additional search details and calculate the size of each method's search space. The bottom row of Figure 5 illustrates our search results. We note that AutoBuild-NAS leads to superior Pareto frontiers compared to other methods across all search spaces and hardware devices, specifically all MBv3 tests and PN-GPU. In fact, while both AutoBuild-NAS and Stage-Full can break 79% accuracy on MBv3-CPU and MBv3-NPU, only AutoBuild-NAS achieves this at less than 5ms latency on MBv3-NPU. Also, at the lower-latency end of the Pareto frontiers, the strongest competitor is generally Layer-Insight, who also shrink the search space. However, gains attained by Layer-Insight are quite small and incon-

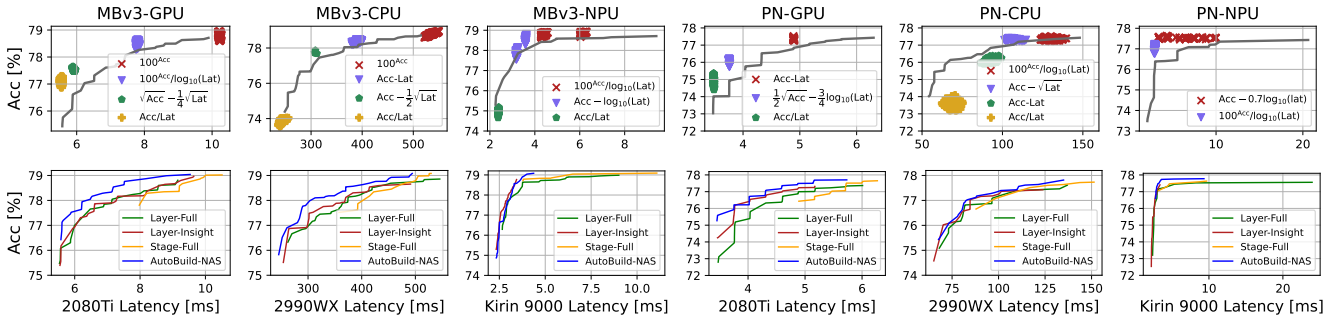


Figure 5. **Top row:** Comparing reduced search spaces produced by AutoBuild (colored clusters; K=3) to the accuracy-latency Pareto frontier of the architectures used to train the predictor (grey line). Each cluster corresponds to a specific target equation designed to improve upon a specific region of the frontier. **Bottom row:** Evolutionary search results comparing the reduced AutoBuild-NAS search space (K=25) and unit-level mutation to other search spaces and mutation techniques. Best viewed in color.

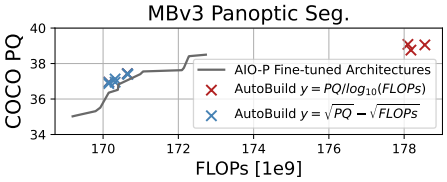


Figure 6. Results comparing AutoBuild MBv3 architectures to the PQ-FLOPs Pareto frontier of fine-tuned architectures from [27].

sistent. Moreover, while Layer-Insight sampled many thousands of architectures to build insights and relies on upon human knowledge to decide how to reduce the search space, AutoBuild-NAS is data-driven and only relies on $3k$ labeled samples to train each AutoBuild predictor.

5.2. Application to Panoptic Segmentation

AutoBuild can construct architectures for higher-resolution computer vision tasks, like Panoptic Segmentation (PS) [15, 44] on MS-COCO [18]. However, evaluation on this task is costly compared to classification as it requires fine-tuning on over 50GB of VRAM for over 24 V100 GPU hours per MBv3 architecture. AIO-P [27] provides performance annotations for $\sim$100 fully fine-tuned and $\sim 1.3k$ 'pseudo-labeled' MBv3 architectures, respectively.

We train AutoBuild predictors to find MBv3 module subgraphs that provide superior and efficient Panoptic Quality (PQ). Specifically, we calculate module scores by training predictors on the 'pseudo-labeled' architectures using two target equations: $PQ/log_{10}(FLOPs)$ aims to maximize PQ while not finding overtly large architectures, while $\sqrt{PQ} - \sqrt{FLOPs}$ strikes a balance between performance and computation cost. For each equation, we evaluate a small handful of constructed architectures, then compare their PQ and FLOPs to the Pareto frontier derived from the fully fine-tuned architectures.

Figure 6 presents our findings. First, we note that module subgraphs selected using $y = PQ/log_{10}(FLOPs)$ obtain high PQ, generally at or above 39 which is better than any of the fully fine-tuned architectures from [27]. However, the FLOPs of these architectures is quite high and generally overshoots the high-FLOPs tail of the Pareto frontier by about 5 billion FLOPs. By contrast, architectures constructed using $y = \sqrt{PQ} - \sqrt{FLOPs}$ strike a balance between performance and the computational burden and outperform the existing Pareto frontier. Thus, Fig. 6 demonstrates the applicability of AutoBuild in scenarios where evaluation is costly and the ability to use 'pseudo-labeled' samples to provide accurate architecture module scores.

5.3. Generative AI with Limited Evaluations

AutoBuild is applicable in emerging Generative AI tasks, such as Inpainting based on Stable Diffusion, under very limited architecture evaluations. In this task, we mask out certain parts of an image and rely on Stable-Diffusion models to refill the missing part. Training diffusion models involves a forward pass where Gaussian noises are added to the input image in a step-wise fashion. The masked area is replaced during inference with Gaussian noise, so only the reverse process is required to generate new content in that region. The baseline Inpainting model, SDv1.4 [33], consists of a VQ-GAN [8] encoder/decoder and a denoising U-Net which has 4 stages, each consisting of a series of residual convolution and attention layers. We illustrate the whole model structure in the supplementary.

Our goal is to find a high-performance variant of the original SDv1.4 U-Net by altering the number of channels in each stage and enabling/disabling attention or convolution layers in each stage. Similar to SnapFusion [16], we remove attention from the early stages of the network, as they require too many FLOPs and do not provide much performance gain. The above pruning and channel altering forms a macro-search space containing $\sim 800k$ architectures.

To apply AutoBuild, we first generate a set of 68 random architectures from this search space including SDv1.4 U-Net, each inheriting weights from the original SDv1.4 U-

Table 1. Comparing AutoBuild to an Exhaustive evaluation technique on SDv1.4 and the FID scores (lower is better) in the predictor dataset. Brackets () denote architecture set size.

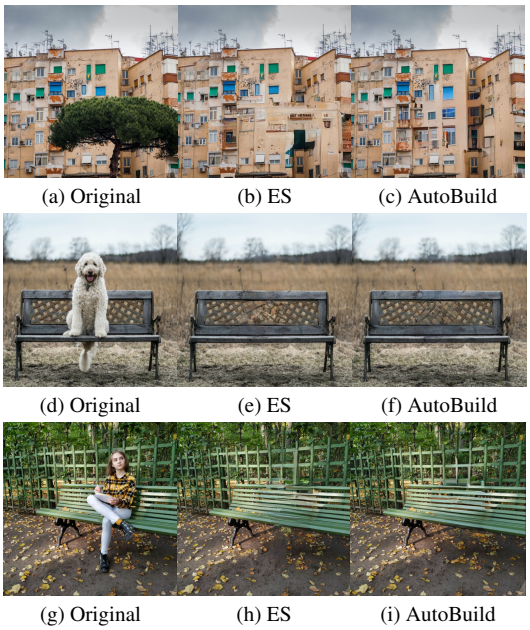
Arch Set	Eval Archs (68)	Exhaustive Search (4)	AutoBuild (4)
Ave. FID	22.13	10.82	**10.13**
Best FID	10.54	10.29	**9.96**

Net. We fine-tune architectures on the Places365 [48] training set for 20k steps, then measure the Fréchet Inception Distance (FID) [11] using 3k images from the validation set with predefined masks. Each architecture fine-tuning (and the evaluation) takes 1-2 days and over 60GB of VRAM. As such, traditional NAS methods would require several dozens of GPUs to explore 0.1% of 800k architectures. We show that just based on these 68 evaluated architectures, AutoBuild can directly build better architectures.

To deal with a small number of evaluated architectures, we train an ensemble of 20 AutoBuild predictors, each trained on 4/5 of the 68 architectures under a 5-fold split. This procedure is repeated under 4 random seeds (to shuffle data), leading to 20 predictors. Then, each subgraph, which is a specific sequence of convolution and attention layers with a stage label and other features including number of channels, is scored by a weighted sum of this subgraph's embedding norms under the 20 predictors, where the weighting is given by the test SRCC between the subgraph embedding norms by a predictor and ground-truth labels at the hop-level of the subgraph in question. Thus, we can use the test SRCC on the remaining 1/5 held-out architectures as a confidence measure to weigh a predictor's estimation of subgraph importance. For AutoBuild, we fine-tune and evaluate the top-4 architectures according to the summation of weighted subgraph scores over each stage.

Additionally, we also consider a baseline search strategy, which applies Exhaustive Search (ES) to all 800k architectures, with performance estimation provided by an ensemble of 20 predictors trained in the same way as above except that only the MSE loss (between predicted and ground-truth FID) is used in training predictors since no node/subgraph embeddings are generated. The other difference is that the predictor weight is the test SRCC between FID predictions and the ground-truth. We then fine-tune and evaluate the top-4 architectures selected by ES.

Table 1 shows the true FID scores for the top-4 architectures selected by each method. While both methods can find architectures that are better than the 68 evaluated architectures, AutoBuild finds much better architectures than ES does, on average, and in terms of the best architecture found (with a much lower FID). This demonstrates the power of AutoBuild in constructing good architectures through interpretable subgraph mining, which cannot be achieved by ensembled predictors trained on the same 68 evaluated architectures. In addition, ES (with ensembled predictions) is only possible due to the search space being still relatively small ($800k$). Since feeding each candidate architecture into the predictor will still cause overhead, for even larger search spaces, ES becomes undesirable or infeasible. We provide additional details and results in the supplementary.



(a) Original (b) ES (c) AutoBuild

(d) Original (e) ES (f) AutoBuild

(g) Original (h) ES (i) AutoBuild

Figure 7. Visual comparison of the best architectures found using Exhaustive Search and AutoBuild on sample images from SDv1.4.

Finally, we visually compare the inpainting outcomes of the best architectures found by AutoBuild and ES. Figure 7 provides Inpainting samples from both U-Net variants. The AutoBuild architecture clearly produces more accurate content, e.g., faithfully representing the building structure and the backrests of both benches whereas the ES variant fails to do so. Thus, these findings demonstrate the robust utility of AutoBuild and architecture module scoring.

6. Conclusion

We propose AutoBuild, to measure and rank the importance of the architecture modules that exist within search spaces as opposed to focusing on traversing the whole design space as traditional NAS does. AutoBuild applies a hop-level ranking correlation loss to learn graph embedding norms that are correlated with architecture performance labels. This method allows us to assign numerical scores to architecture modules of different sizes. We then use these scores to quickly construct high-quality classification and segmentation models to improve or circumvent NAS by reducing the search space. Finally, we show that AutoBuild is applicable in scenarios, where few architectures have been evaluated, as we use it to find a high-performance variant of Stable Diffusion v1.4 for Inpainting based on only 68 labeled architectures.

References

[1] Junjie Bai, Fang Lu, Ke Zhang, et al. Onnx: Open neural network exchange. https://github.com/onnx/onnx, 2019. 2

[2] Mathieu Blondel, Olivier Teboul, Quentin Berthet, and Josip Djolonga. Fast differentiable sorting and ranking. In *International Conference on Machine Learning*, pages 950–959. PMLR, 2020. 4, 1

[3] Shaked Brody, Uri Alon, and Eran Yahav. How attentive are graph attention networks? In *International Conference on Learning Representations*, 2022. 2, 3, 1

[4] Han Cai, Chuang Gan, Tianzhe Wang, Zhekai Zhang, and Song Han. Once for all: Train one network and specialize it for efficient deployment. In *International Conference on Learning Representations*, 2020. 1, 3, 6, 2

[5] Yaofo Chen, Yong Guo, Qi Chen, Minli Li, Wei Zeng, Yaowei Wang, and Mingkui Tan. Contrastive neural architecture search with neural architecture comparators. In *Proceedings of the IEEE/CVF conference on computer vision and pattern recognition*, pages 9502–9511, 2021. 2

[6] Xiaoliang Dai, Alvin Wan, Peizhao Zhang, Bichen Wu, Zijian He, Zhen Wei, Kan Chen, Yuandong Tian, Matthew Yu, Peter Vajda, and Joseph E. Gonzalez. Fbnetv3: Joint architecture-recipe search using predictor pretraining. In *IEEE Conference on Computer Vision and Pattern Recognition, CVPR 2021, virtual, June 19-25, 2021*, pages 16276–16285. Computer Vision Foundation / IEEE, 2021. 1, 2, 3

[7] 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*, pages 248–255. Ieee, 2009. 1

[8] Patrick Esser, Robin Rombach, and Bjorn Ommer. Taming transformers for high-resolution image synthesis. In *Proceedings of the IEEE/CVF conference on computer vision and pattern recognition*, pages 12873–12883, 2021. 7, 5

[9] Matthias Fey and Jan E. Lenssen. Fast graph representation learning with PyTorch Geometric. In *ICLR Workshop on Representation Learning on Graphs and Manifolds*, 2019. 1

[10] Kaiming He, Xiangyu Zhang, Shaoqing Ren, and Jian Sun. Deep residual learning for image recognition. In *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*, pages 770–778, 2016. 3

[11] Martin Heusel, Hubert Ramsauer, Thomas Unterthiner, Bernhard Nessler, and Sepp Hochreiter. Gans trained by a two time-scale update rule converge to a local nash equilibrium. *Advances in neural information processing systems*, 30, 2017. 8

[12] 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*, pages 1314–1324, 2019. 2, 3, 5

[13] Andrew G Howard, Menglong Zhu, Bo Chen, Dmitry Kalenichenko, Weijun Wang, Tobias Weyand, Marco Andreetto, and Hartwig Adam. Mobilenets: Efficient convolutional neural networks for mobile vision applications. *arXiv preprint arXiv:1704.04861*, 2017. 2, 3

[14] Jie Hu, Li Shen, and Gang Sun. Squeeze-and-excitation networks. In *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*, pages 7132–7141, 2018. 5

[15] Alexander Kirillov, Ross Girshick, Kaiming He, and Piotr Dollár. Panoptic feature pyramid networks. In *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition*, pages 6399–6408, 2019. 2, 7

[16] Yanyu Li, Huan Wang, Qing Jin, Ju Hu, Pavlo Chemerys, Yun Fu, Yanzhi Wang, Sergey Tulyakov, and Jian Ren. Snapfusion: Text-to-image diffusion model on mobile devices within two seconds. In *Advances in Neural Information Processing Systems*, pages 20662–20678. Curran Associates, Inc., 2023. 1, 7

[17] Heng Liao, Jiajin Tu, Jing Xia, Hu Liu, Xiping Zhou, Honghui Yuan, and Yuxing Hu. Ascend: a scalable and unified architecture for ubiquitous deep neural network computing : Industry track paper. *2021 IEEE International Symposium on High-Performance Computer Architecture (HPCA)*, pages 789–801, 2021. 5

[18] Tsung-Yi Lin, Michael Maire, Serge Belongie, Lubomir Bourdev, Ross Girshick, James Hays, Pietro Perona, Deva Ramanan, C. Lawrence Zitnick, and Piotr Dollár. Microsoft coco: Common objects in context. In *Computer Vision – ECCV 2014*, pages 740–755, Cham, 2014. Springer International Publishing. 5, 7

[19] Hanxiao Liu, Karen Simonyan, and Yiming Yang. Darts: Differentiable architecture search. In *International Conference on Learning Representations (ICLR)*, 2019. 1

[20] Yuqiao Liu, Yanan Sun, Bing Xue, Mengjie Zhang, Gary G Yen, and Kay Chen Tan. A survey on evolutionary neural architecture search. *IEEE transactions on neural networks and learning systems*, 2021. 6

[21] Ilya Loshchilov and Frank Hutter. Decoupled weight decay regularization. In *7th International Conference on Learning Representations, ICLR 2019, New Orleans, LA, USA, May 6-9, 2019*. OpenReview.net, 2019. 1

[22] Shun Lu, Yu Hu, Peihao Wang, Yan Han, Jianchao Tan, Jixiang Li, Sen Yang, and Ji Liu. Pinat: A permutation invariance augmented transformer for nas predictor. In *Proceedings of the AAAI Conference on Artificial Intelligence (AAAI)*, 2023. 2

[23] Zhichao Lu, Ian Whalen, Vishnu Boddeti, Yashesh Dhebar, Kalyanmoy Deb, Erik Goodman, and Wolfgang Banzhaf. Nsga-net: neural architecture search using multi-objective genetic algorithm. In *Proceedings of the genetic and evolutionary computation conference*, pages 419–427, 2019. 6

[24] Yash Mehta, Colin White, Arber Zela, Arjun Krishnakumar, Guri Zabergja, Shakiba Moradian, Mahmoud Safari, Kaicheng Yu, and Frank Hutter. Nas-bench-suite: NAS evaluation is (now) surprisingly easy. In *The Tenth International Conference on Learning Representations, ICLR 2022, Virtual Event, April 25-29, 2022*. OpenReview.net, 2022. 1, 2

[25] Keith G. Mills, Fred X. Han, Jialin Zhang, Seyed Saeed Changiz Rezaei, Fabian Chudak, Wei Lu, Shuo Lian, Shangling Jui, and Di Niu. Profiling neural blocks and design

spaces for mobile neural architecture search. In *Proceedings of the 30th ACM International Conference on Information & Knowledge Management*, page 4026–4035, New York, NY, USA, 2021. Association for Computing Machinery. 2, 3, 4, 5, 6, 1

[26] Keith G. Mills, Fred X. Han, Jialin Zhang, Fabian Chudak, Ali Safari Mamaghani, Mohammad Salameh, Wei Lu, Shangling Jui, and Di Niu. Gennape: Towards generalized neural architecture performance estimators. *Proceedings of the AAAI Conference on Artificial Intelligence*, 37(8):9190–9199, 2023. 2

[27] Keith G. Mills, Di Niu, Mohammad Salameh, Weichen Qiu, Fred X. Han, Puyuan Liu, Jialin Zhang, Wei Lu, and Shangling Jui. Aio-p: Expanding neural performance predictors beyond image classification. *Proceedings of the AAAI Conference on Artificial Intelligence*, 37(8):9180–9189, 2023. 2, 7, 1, 4

[28] Christopher Morris, Martin Ritzert, Matthias Fey, William L Hamilton, Jan Eric Lenssen, Gaurav Rattan, and Martin Grohe. Weisfeiler and leman go neural: Higher-order graph neural networks. In *Proceedings of the AAAI Conference on Artificial Intelligence*, pages 4602–4609, 2019. 2

[29] OpenAI. Gpt-4 technical report, 2023. 1

[30] Adam Paszke, Sam Gross, Francisco Massa, Adam Lerer, James Bradbury, Gregory Chanan, Trevor Killeen, Zeming Lin, Natalia Gimelshein, Luca Antiga, Alban Desmaison, Andreas Kopf, Edward Yang, Zachary DeVito, Martin Raison, Alykhan Tejani, Sasank Chilamkurthy, Benoit Steiner, Lu Fang, Junjie Bai, and Soumith Chintala. Pytorch: An imperative style, high-performance deep learning library. In *Advances in Neural Information Processing Systems 32*, pages 8024–8035. Curran Associates, Inc., 2019. 1

[31] Hieu Pham, Melody Guan, Barret Zoph, Quoc Le, and Jeff Dean. Efficient neural architecture search via parameters sharing. In *International Conference on Machine Learning*, pages 4095–4104. PMLR, 2018. 1

[32] Dustin Podell, Zion English, Kyle Lacey, Andreas Blattmann, Tim Dockhorn, Jonas Müller, Joe Penna, and Robin Rombach. Sdxl: Improving latent diffusion models for high-resolution image synthesis. *arXiv preprint arXiv:2307.01952*, 2023. 2

[33] 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 (CVPR)*, pages 10684–10695, 2022. 1, 2, 3, 7, 5

[34] Binxin Ru, Xingchen Wan, Xiaowen Dong, and Michael A. Osborne. Interpretable neural architecture search via bayesian optimisation with weisfeiler-lehman kernels. In *ICLR*, 2021. 2

[35] Mohammad Salameh, Keith G. Mills, Negar Hassanpour, Fred Han, Shuting Zhang, Wei Lu, Shangling Jui, Chunhua Zhou, Fengyu Sun, and Di Niu. Autogo: Automated computation graph optimization for neural network evolution. In *Advances in Neural Information Processing Systems*, pages 74455–74477. Curran Associates, Inc., 2023. 2

[36] Mark Sandler, Andrew Howard, Menglong Zhu, Andrey Zhmoginov, and Liang-Chieh Chen. Mobilenetv2: Inverted residuals and linear bottlenecks. In *Proceedings of the IEEE conference on computer vision and pattern recognition*, pages 4510–4520, 2018. 2, 3

[37] Yanan Sun, Handing Wang, Bing Xue, Yaochu Jin, Gary G. Yen, and Mengjie Zhang. Surrogate-assisted evolutionary deep learning using an end-to-end random forest-based performance predictor. *IEEE Transactions on Evolutionary Computation*, 24(2):350–364, 2020. 6

[38] Mingxing Tan and Quoc Le. Efficientnet: Rethinking model scaling for convolutional neural networks. In *International Conference on Machine Learning*, pages 6105–6114. PMLR, 2019. 3

[39] 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 *Advances in Neural Information Processing Systems*. Curran Associates, Inc., 2017. 3

[40] Petar Velickovic, Guillem Cucurull, Arantxa Casanova, Adriana Romero, Pietro Liò, and Yoshua Bengio. Graph attention networks. In *6th International Conference on Learning Representations, ICLR 2018, Vancouver, BC, Canada, April 30 - May 3, 2018, Conference Track Proceedings*. OpenReview.net, 2018. 2, 3

[41] Alvin Wan, Xiaoliang Dai, Peizhao Zhang, Zijian He, Yuandong Tian, Saining Xie, Bichen Wu, Matthew Yu, Tao Xu, Kan Chen, et al. Fbnetv2: Differentiable neural architecture search for spatial and channel dimensions. In *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition*, pages 12965–12974, 2020. 1, 2, 3

[42] Max Welling and Thomas N Kipf. Semi-supervised classification with graph convolutional networks. In *J. International Conference on Learning Representations (ICLR 2017)*, 2016. 3

[43] Colin White, Arber Zela, Robin Ru, Yang Liu, and Frank Hutter. How powerful are performance predictors in neural architecture search? *Advances in Neural Information Processing Systems*, 34:28454–28469, 2021. 2, 4

[44] Yuxin Wu, Alexander Kirillov, Francisco Massa, Wan-Yen Lo, and Ross Girshick. Detectron2. https://github.com/facebookresearch/detectron2, 2019. Accessed: 2022-08-15. 7

[45] Keyulu Xu, Weihua Hu, Jure Leskovec, and Stefanie Jegelka. How powerful are graph neural networks? In *7th International Conference on Learning Representations, ICLR 2019, New Orleans, LA, USA, May 6-9, 2019*, 2019. 2, 3

[46] Rex Ying, Tianyu Fu, Andrew Wang, Jiaxuan You, Yu Wang, and Jure Leskovec. Representation learning for frequent subgraph mining, 2024. 2

[47] Jiaxuan You, Zhitao Ying, and Jure Leskovec. Design space for graph neural networks. *Advances in Neural Information Processing Systems*, 33, 2020. 3, 2

[48] Bolei Zhou, Agata Lapedriza, Aditya Khosla, Aude Oliva, and Antonio Torralba. Places: A 10 million image database for scene recognition. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, 2017. 8, 6