Building Optimal Neural Architectures using Interpretable Knowledge

Supplementary Material

7. Supplementary Materials

We provide additional information and materials, e.g., predictor training, details of how ImageNet performance metrics are obtained, and further elaborate on our experimental setup, results and insights into Stable Diffusion U-Nets.

7.1. Predictor Training Setup

We implement AutoBuild using PyTorch [30] and PyTorch-Geometric [9]. We use a GATv2 [3] GNN backbone with a hidden size of 32. There are 4 message-passing layers for PN/MBv3 and 6 message-passing layers for SDv1.4. Message-passing layers include a ReLU activation and residual skip-connections between hop layers.

The FE-MLP applies a separate MLP to each node feature category to generate a single scalar per category. The only activation in the FE-MLP is an `abs` operation which is applied to the concatenation of each node feature scalar to produce the 0-hop embedding. Finally, graph embeddings at all hop-levels are generating using mean aggregation (Eq. 2).

We train AutoBuild using Equation 4 (differentiable SRCC provided by [2]) with a batch size of 128. We use a AdamW [21] optimizer with an initial learning rate of $1e-4$ and weight decay of $1e-5$. Also, we standardize prediction labels using the sample mean and variance computed from the training data split.

The number of epochs differs depending on task and the number of labeled samples we have. For the classification results in Sec. 5.1, we train for 200 epochs with $3k$ labeled architectures ($6k$ to generate Fig. 3) at a training speed of 1 epoch/GPU second. For panoptic segmentation results in Sec. 5.2, we train for 300 epochs using the $1.3k$ 'pseudo-labeled' MBv3 architectures from AIO-P [27], while comparing against Pareto frontier generated from the 118 fully fine-tuned MBv3 architecture samples they provide. Finally, for SDv1.4, we have 68 samples and train each predictor (AutoBuild and Exhaustive Search) for 1000 epochs.

7.2. Search Space Sizes

We provide formal mathematics to quantify the number of architecture modules present in each search space, as well as the size of each original macro-search space. Also, we calculate the size of the reduced search spaces used by **AutoBuild-NAS** and the **Layer-Insight** [25] from Sec. 5.1.

Unique Architecture Modules. PN and MBv3 both use MBConv blocks as layers. There are 9 unique MBConv configurations formed as the cartesian product of kernel sizes $k \in \{3, 5, 7\}$ and expansion ratios $e \in \{3, 4, 6\}$. Stages 1-5 of PN and MBv3 contain 2-4 layers, for a total of $9^2 + 9^3 + 9^4 = 81 + 729 + 6561 = 7371$ unique 2-4 modules. This is sufficient for MBv3.

PN has an additional 6th stage which always contain only a single layer, giving it 9 additional subgraphs for 7380 in total. However, in practice when enumerating over PN architecture modules and performing NAS, we merge the 5th and 6th stage for simplicity, meaning we consider 3-5 node subgraph modules of which there are $9^3 + 9^4 + 9^5 = 66339$.

Mathematically Calculating Search Space Size. Let u represent a given stage, l_u be the number of layers in stage u, and $\mathcal{B}_u$ be the set of unique layers available at stage u. Each stage is a subgraph that contains $l_u \in [l_u^{min}, l_u^{max}]$ layers, each sampled from $\mathcal{B}_u$ with replacement. The maximum number of unique stage subgraphs is therefore $|\mathcal{S}_u| = \sum_{l_u=l_u^{min}}^{l_u^{max}} |\mathcal{B}_u|^{l_u}$. Likewise, the total size of the search space is the cartesian product of all subgraphs across all stages, $\prod_{u=u^{min}}^{u^{max}} |\mathcal{S}_u|$.

For MBv3, these calculations work out to be $|\mathcal{S}_u| = \sum_{l_u=2}^{4} 9^{l_u} = 7371$, for $u \in U = \{1, 2, .., 5\}$ as previously computed. The overall size is therefore $\prod_1^5 7371 \approx 2.18 \times 10^{19}$. For PN, this number is multiplied by 9 due to the additional 6th stage, for approximately 1.96×10^{20} total architectures. It is infeasible if not intractable to consider all of these architectures, even using a predictor, yet by contrast the number of architecture module subgraphs we can consider is substantially smaller such that we can enumerate over them all.

Quantifying Reduced Search Spaces. Also, the search space can be greatly reduced by limiting the size of $\mathcal{S}_u$, the number of possible subgraphs per stage. We reduce $|\mathcal{S}_u|$ by directly selecting a subset of architecture module subgraphs using AutoBuild in a data-driven manner. Specifically, for each target equation in the upper row of Fig. 5, we select the top-K module subgraphs per unit where $K = 25$. There are a maximum of $P = 4$ target equations we consider at a time, for a total of $(4 \times 25)^5 = 10^{10}$ architectures, still significantly less than the original search space by a factor of over 2×10^9.

By contrast, **Layer-Insight** reduce $|\mathcal{B}_u|$ and l_u^{max} by hand using human expert knowledge. For 'MBv3-GPU', they restrict $l_u^{max} = 3$ for stages 2, 4 and 5 and remove 2 MBConv blocks from consideration at all, resulting in a search space containing more than 4.6×10^{14} architectures. Likewise, for 'MBv3-CPU', they only constrain $l_u^{max} = 3$ for stages 1, 2 and 3, which only reduces the search space to 2.9×10^{16} architectures. Finally, 'MBv3-NPU' removes

Table 2. Distribution statistics from 2.7k (90% of 3k) random MBv3 architectures. $\mathcal{N}(\mu^h_m, \sigma^h_m)$ is the distribution of embedding norms for an accuracy predictor, while $\mathcal{N}(\mu^y_{u,l}, \sigma^y_{u,l})$ is the distribution of ImageNet top-1 accuracies [%] per (u, l). **Note**: $l = m + 1$.

	$\mathcal{N}(\mu^h_m, \sigma^h_m)$	$\mathcal{N}(\mu^y_{u,l}, \sigma^y_{u,l})$				
l	$m = l - 1$	$u = 1$	$u = 2$	$u = 3$	$u = 4$	$u = 5$
1	$\mathcal{N}(2.24, 0.39)$	–	–	–	–	–
2	$\mathcal{N}(13.83, 7.14)$	$\mathcal{N}(76.77, 0.82)$	$\mathcal{N}(76.77, 0.81)$	$\mathcal{N}(76.69, 0.85)$	$\mathcal{N}(76.42, 0.75)$	$\mathcal{N}(76.61, 0.79)$
3	$\mathcal{N}(27.53, 8.67)$	$\mathcal{N}(77.00, 0.79)$	$\mathcal{N}(76.98, 0.78)$	$\mathcal{N}(77.02, 0.76)$	$\mathcal{N}(77.05, 0.70)$	$\mathcal{N}(76.96, 0.76)$
4	$\mathcal{N}(40.94, 9.73)$	$\mathcal{N}(77.08, 0.76)$	$\mathcal{N}(77.12, 0.78)$	$\mathcal{N}(77.13, 0.74)$	$\mathcal{N}(77.41, 0.62)$	$\mathcal{N}(77.26, 0.73)$

$k = 7$ from consideration so $|\mathcal{B}|_u = 6$, giving a search space of size of $\sim 9 \times 10^{15}$. All of these are substantially larger than the reduced search spaces of AutoBuild.

7.3. Distribution Shift Statistics

We elaborate on how to calculate the statistics from Section 4.2, i.e., $\mathcal{N}(\mu^h_m, \sigma^h_m)$ and $\mathcal{N}(\mu^y_{u,l}, \sigma^y_{u,l})$. We also provide some empirical measurements to show the necessity of these calculations.

Recall that $\mathcal{N}(\mu^h_m, \sigma^h_m)$ is the distribution of node embedding norms learned by the GNN predictor, where h refers to a hidden GNN embedding and m parameterizes the hop-level. For each hop-level m, the statistics of this distribution are calculated after predictor training by a double for-loop that enumerates over every graph in the training set, and even node within each graph.

By contrast, $\mathcal{N}(\mu^y_{u,l}, \sigma^y_{u,l})$ is the distribution of target labels $y_{\mathcal{G}}$ for graphs that have an architecture module subgraph with l nodes positioned at stage u. These statistics are calculated by enumerating over the training dataset. Moreover, these statistics are specific to sequence-graphs and calculating them requires a sufficient number of graphs per (u, l) tuple[1].

We tabulate these statistics for an MBv3 accuracy predictor in Table 2. Note how the values of μ^h_m and σ^h_m tend to grow monotonically with m. Also note how the accuracy distribution increases with l, yet differs for each stage u.

7.4. Computation of ImageNet Metrics

We describe the evaluation protocol details for retrieving accuracy and latency of for all architecture AutoBuild constructs in the top row of Fig. 5 or found by search in the bottom row of Fig. 5. Specifically, we obtain accuracy by splicing architecture subnets from the corresponding OFA [4] supernetwork and directly evaluating on ImageNet, which takes about 1 V100 GPU minute per architecture.

We obtain latency measurements using predictors. These pure latency predictors are distinct from those mentioned in Figs. 2 & 3 and the AutoBuild-NAS predictors described in Sec. 7.1 which we use to reduce the search space, but they are similar in design. Specifically, our latency predictors train on the OFA-MBv3/PN samples that [25] provide[2] for 500 epochs with a batch size of 32 to minimize the loss

$$\mathcal{L}_{Lat} = \texttt{MAE}(y_{\mathcal{G}}, y'_{\mathcal{G}}) + (1 - \rho(y_{\mathcal{G}}, y'_{\mathcal{G}})), \tag{6}$$

where MAE is the Mean Absolute Error and $y_{\mathcal{G}}$ is the ground-truth latency of architecture $\mathcal{G}$. Also different from the AutoBuild predictors, the latency predictors use k-GNN [28] message-passing layers with a hidden size of 128 and LeakyReLU activations. Finally, latency predictors compute graph embeddings using 'sum' aggregation [47] rather than the 'mean' aggregation of Eq. 2.

7.5. NAS Evolutionary Algorithm Implementation

We use the random mutation-based EA provided by [25]. This algorithm randomly samples an initial pool of architectures and establishes a population of the most Pareto-optimal architectures found. The algorithm then iteratively mutates architectures in the frontier by creating a predefined number of mutated architectures per iteration, and merging them into the existing Pareto frontier. The algorithm repeats for a set number of iterations. We calculate the total evaluation using $eval\_budget = initial\_archs + iter \times eval\_per\_iter$. We set a total evaluation budget of 250 by setting $initial\_archs = 50$, $iter = 4$ and $eval\_per\_iter = 50$ for each experimental setup.

By default, the algorithm performs NAS by mutating the layers of OFA-PN/MBv3 networks, e.g., adding, removing or altering the MBConv choice for a single layer per stage. Their code implements **Layer-Full** by default but also includes the search space restrictions required to run **Layer-Insight**, both of which we compare to in Fig. 5. We extend the code to provide support for **Stage-Full** and **AutoBuild-NAS**, both of which uses a different mutation mechanism where entire stages are changed at once rather than single layers. Note that both layer-wise and stage-wise mutation are equivalent in terms of evaluations. That is, whether a mutation changes one layer or one stage, the resultant architecture is still different from the original and must be evaluated separately to obtain a gauge performance.

[1] We have around 1k for Sec. 5.1 and around 430 for Sec. 5.2.

[2] $50k/15k/5k$ unique architectures for the GPU/CPU/NPU devices with latency measurements for three input image sizes: $\{192^2, 208^2, 224^2\}$

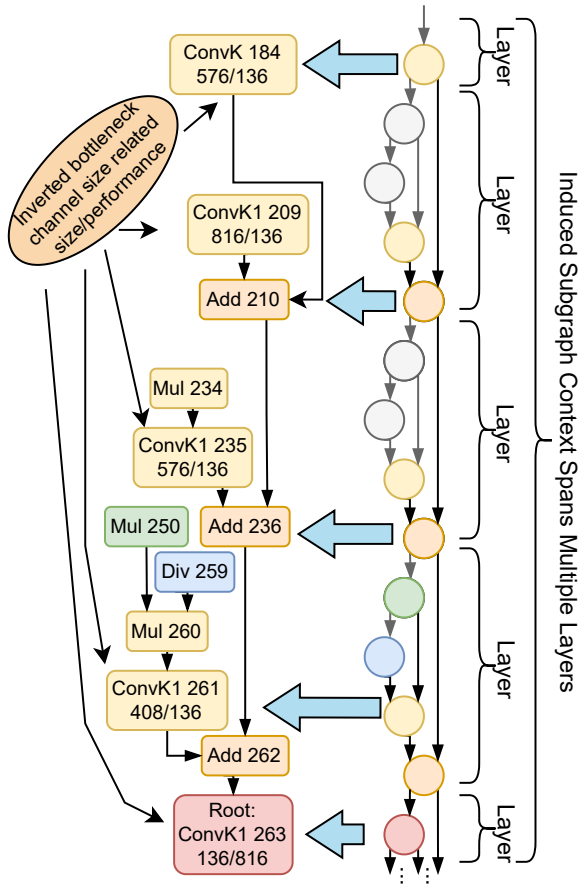


Figure 8. Annotated illustration of the 4-hop subgraph which achieved the highest score when training a AutoBuild accuracy predictor for PN/MBv3 using the 'IR' format. Specifically, this subgraph comes from a large MBv3 architecture and is rooted at the first convolution in an MBConv block with $e = 6$. We annotate each node by operation type and topological sort index. Convolution nodes are further labeled with 'Input Channel/Output Channel' numbers. Individual nodes are color-coded to match with the general area of the larger network they belong to.

7.6. Additional FE-MLP Results

We elaborate on why the operation type with the highest FE-MLP score in the right subfigure of Fig. 3 is 'Add'. Further, we also provide additional FE-MLP results for PN and MBv3 that could not be included in the main manuscript due to space constraints.

Importance of 'Add'. Both PN and MBv3 use long residuals to connect the start and end of MBConv blocks. These residual connections play a key role in network accuracy improvement and the add operator is often used to implement such skip-connections. As such, with sufficient hop-levels, subgraphs which incorporate add operations can span large sections of the overall network in order to accrue information from critical nodes that are far apart. For example, consider the MBv3 graph in Figure 8. This graph contains several hundred nodes, yet the highest-ranked subgraph is rooted at 'Conv 263'. This is a pointwise convolution[3] which has input and output channels of 136 and 816, respectively, meaning that its channel expansion ratio is $e = {}^{816}/_{136} = 6$, the highest possible e value in PN/MBv3. Since the input channel size is 136, 'Conv 263' is located in Stage 4 of MBv3, where changes in the number of layers/block choice can have a great impact on performance (see Fig. 5 of [25]). Additionally, it is directly downstream from 'Add 262', which receives input from 'Conv 261' which has $e = 3$ and 'Add 236'. 'Add 236' is fed by 'Conv 235' that has $e = 4$ and 'Add 210', in turn fed by 'Conv 209' ($e = 6$) and 'Conv 184' ($e = 4$)[4].

This long chain of operation spans almost 80 node indices by topological sort, and the inclusion of 'Div 259' signals that this subgraph is part of an MBv3 network, not PN. However, more importantly, the subgraph captures multiple pointwise convolutions which control the channel ratios in PN/MBv3. The number of channels directly affects the number parameters and FLOPs required to perform a forward pass - both of which are highly correlated with accuracy performance [25]. Of the 5 pointwise convolutions captured in the subgraph, only 1 of them has $e = 3$, the worst option, while the remaining either have $e = 4$ or $e = 6$. Thus, not only does the subgraph belong to an MBv3 network, but the operations within it correspond to MBConv layers that perform heavy computations and likewise, help attain high performance. Since the formation of this subgraph fully depends on the residual 'Add' operations, that operation type is assigned the highest FE-MLP score in Fig. 3.

FE-MLP Sequence Graph Scores. The FE-MLP of AutoBuild provides a direct mean of measuring the scalar values for each choice of node feature, such as discrete kernel size, and interpreting these values as importance scores. Higher values correspond to higher target values through Eq. 4. PN and MBv3 use MBConv blocks as layers, with a total of 9 types spanning kernel sizes $k \in \{3, 5, 7\}$ and channel expansion ratios $e \in \{3, 4, 6\}$. We further enrich the node features by including the network input resolution and two position-based node features that encode a node stage u and layer l location. PN architectures contain 6 stages. Stages 1 to 5 contain 2 to 4 layers, while stage 6 always contains 1 layer. We perform an analysis of the feature importance for the PN and MBv3 macro-search spaces. Specifically, scores provided in the following figures stem from the same exper-

[3]The depthwise convolutions with $k \in \{3, 5, 7\}$ are located in the middle of a block. The subgraph in Fig. 8 does not contain any of them.

[4]'Conv 184' is not preceded by an add node as it belongs to the first layer in the stage ('Conv 261'/'Add 262' is part of the 4th and final layer of the 4th MBv3 stage, while 'Conv 263' is the first node in the 5th stage.), where downsampling occurs, and thus, no residual connection is included.

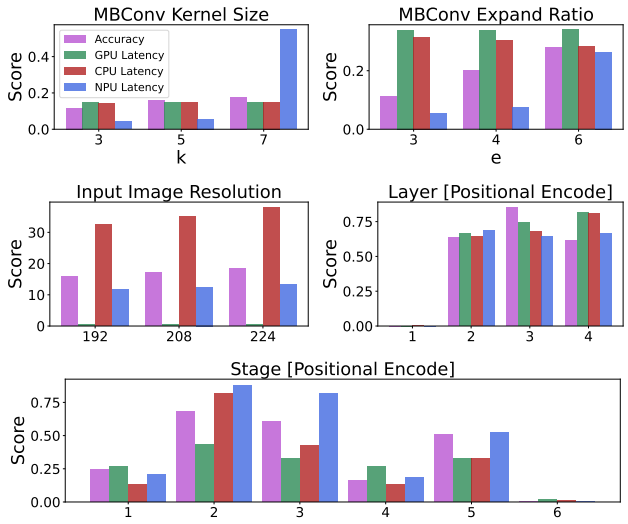


Figure 9. FE-MLP 0-hop feature importance scores for PN. We train a AutoBuild predictor using the 'custom' graph representation which provides 5 feature categories: Kernel, Expand, Resolution, Layers and Stages.

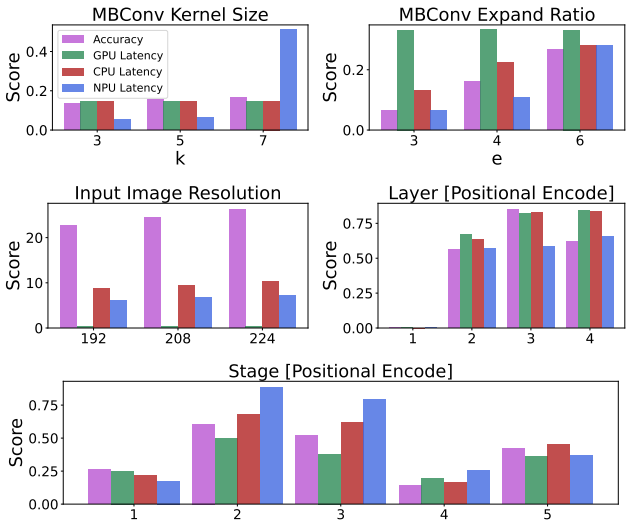


Figure 10. FE-MLP 0-hop feature importance scores for MBv3. Same setup as Fig. 9.

iments performed to generate Fig. 2 in the main manuscript.

Figures 9 and 10 illustrates the range of feature scores for the PN and MBv3 search spaces, respectively. Specifically, we consider accuracy, GPU latency, CPU latency and NPU latency metrics. We note that in both cases AutoBuild assigns disproportionate importance to kernel size 7 for NPU latency, which aligns with [25] who found that $k = 7$ disproportionately increases that metric. In other cases, kernel size, expansion ratio, and resolution generally follow linear trends or are mostly constant, e.g., expansion ratio e for GPU latency on both search spaces, and PN CPU latency. Also, for PN, the resolution feature overall is essential when determining CPU latency, moderately important for accuracy and NPU latency, but has little to no effect in determining GPU latency. Yet, for MBv3, the resolution feature is emphasized more for accuracy than CPU latency.

In terms of positional features, we observe that a near-zero weight is universally assigned for $l = 0$. Intuitively, this makes sense as the first layer always exists, while the 3rd and 4th layers are assigned higher values due to being optional. In [25], architectures with stages containing 3 or more layers had higher GPU and CPU latency, especially if these layers were allocated to stages 2, 3 and 5, which obtained high scores across all metrics. Finally, for PN, stage $u = 6$ is assigned a near-zero weight as it always exists and always contains 1 layer.

7.7. Additional Panoptic Segmentation Results

Additionally, we also investigated the performance of AutoBuild on the ProxylessNAS macro-search space for Panoptic Segmentation. Much like MBv3, AIO-P [27] also provide sets of fully fine-tuned and 'pseudo-labeled' architecture samples. As such, we apply AutoBuild to PN, using the same procedure detailed in Sec. 5.2. The only difference is we differ the target label equations as the PN PQ-FLOPs Pareto frontier differs from that of MBv3.

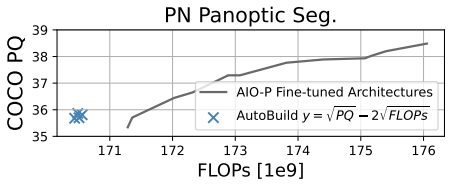


Figure 11. Results comparing AutoBuild PN architectures to the PQ-FLOPs Pareto frontier of fine-tuned architectures from [27].

Figure 11 illustrates our findings for a single target equation, $y = \sqrt{PQ} - 2\sqrt{FLOPs}$. While we are able to outperform the existing Pareto frontier, we originally crafted this target equation to focus on the center of the Pareto frontier. Specifically, we crafted the equation by considering three points on the Pareto frontier (Fig. 11 grey line):

- The 'high' point: $(FLOPs, PQ) = (175.1, 37.9)$
- The 'ideal' point: $(FLOPs, PQ) = (172.9, 37.4)$.
- The 'low' point: $(FLOPs, PQ) = (171.3, 35.7)$.

The idea is to craft an equation where the 'ideal' point receives a higher target than the 'high' and 'low' points. Specifically, using $y = \sqrt{PQ} - 2\sqrt{FLOPs}$, the predictor targets would be -20.31, -20.18 and -20.20 for the 'high', 'ideal' and 'low' points respectively. While the 'ideal' point has the highest target, it is closer to the 'low' point than the 'high' point, which influenced the architecture subgraph components found by AutoBuild. As such, and as stated in Sec. 5.1, improving on the target design scheme is definitely a direction for future work.

7.8. Stable Diffusion 1.4 Inpainting Background

We provide additional details and experimental results regarding the Inpainting task described in Section 5.3 and elaborate on the AutoBuild macro-search space. Further, we provide illustrations of the best architectures found by AutoBuild and insights on the search space.

In the Inpainting task, we mask out certain parts of an image and rely on Stable Diffusion models to re-create the missing content. The baseline Inpainting model consists of two sub-modules: a VQ-GAN [8] and a U-Net. The VQ-GAN is responsible for encoding/decoding images in the RGB domain to/from the latent domain. The U-Net is responsible for removing noises from the input image. The training of Diffusion models involves a forward diffusion process, where Gaussian noises are added to the input image in a step-wise fashion until the entire image becomes pure noise. And a reverse diffusion process, where the U-Net model attempts to remove the added noise and iteratively recovers the original image. During inference, the U-Net generates new content from pure Gaussian noise.

In our experiments we use the open-source Stable Diffusion v1.4 [33] VQ-GAN and U-Net model, but we only optimize the U-Net architecture. To train the U-Net model, the original image, the masked image and the mask are first encoded into latent space by either the VQ-GAN encoder or interpolation. Next, random Gaussian noise is added, and the U-Net is trained to recover the noise-free image in the latent space. An overview of our Inpainting model is illustrated in Figure 12. The U-Net model can be further dissected into input, middle and output branches. The input and output branches contains multiple stages with residual connections in-between (e.g., the output of stage 2 from the input branch will be added to the output of stage 2 from the output branch). Each stage further partitions into Residual Convolution and Attention Blocks. At the end of input stages 1, 2 and 3, there are downsampling blocks. Conversely, at the end of output stages 2, 3 and 4, there are upsampling blocks. The downsampling/upsampling blocks adjust the latent tensor height, width and channels (HWC). Figure 13 provides a compact view of the SDv1.4 U-Net.

7.9. AutoBuild for FID Minimization

We consider two distinct SDv1.4-based search spaces and fine-tuning regimes. The first of which pertains to all results listed and illustrated in Sec. 5.3 and is further detailed in this subsection. Here, the objective is to minimize the FID. The second search space aims to find Pareto-optimal architectures where a secondary hardware metric, e.g., latency, is taken into account. All results and discussion pertaining to the second search space can be found in Sec. 7.10.

FID Minimization Search Space & Evaluation. Our search space is designed around the SDv1.4 U-Net. We always remove the attention blocks from Stage 2, and then configure the rest of the U-Net according to several searchable attributes, given below:

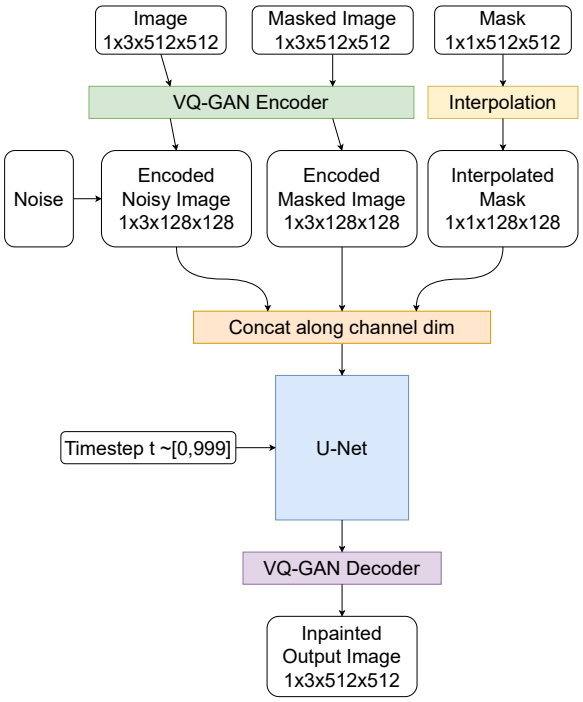


Figure 12. Overview of the Stable Diffusion 1.4 Inpainting model used in our experiments. Note that the U-Net model is our main focus for NAS-based optimization. The U-Net often need to run for multiple steps during inference.

- **U-Net global channel size** C_m: Baseline has 256 channels, our searchable candidates are [128, 192, 256].
- **Stage channel multipliers**: Baseline has multipliers [1, 2, 3, 4] corresponding to channels 256, 512, 768, 1024 for stage 1, 2, 3, 4 of the input/output branches. We add 7 other combinations as searchable candidates (e.g., [1, 1, 2, 4], [1, 3, 4, 2]).
- **Number of residual blocks**: Baseline has 2 residual blocks per stage for the input branch and 3 for the output branch. We allow the optional removal of 1 residual block for any number of stages.
- **Number of attention blocks**: We allow the optional removal of all attention blocks in stages 3 and/or 4.

An illustration of the search space choices is provided by Figure 14. To represent candidate architectures as sequence graphs, we encode the following node features:

- **Operation type**, which is one of: 'Residual Convolution', 'Attention', 'Downsample' or 'Upsample'.
- **Channel size**. **Note:** To instantiate an SDv1.4 architecture variant as a functional neural network, the global channel size C_m must be consistent across all nodes. Also, the channel multiplier values for nodes in an input stage and its corresponding output stage must be the

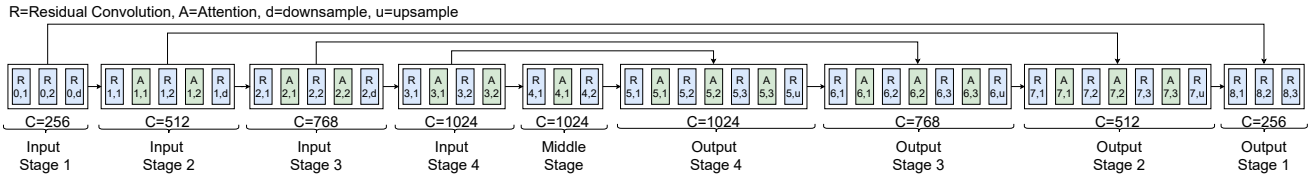


Figure 13. Structure of the baseline U-Net model for Inpainting. E.g., '0,1' means the first block of the first stage.

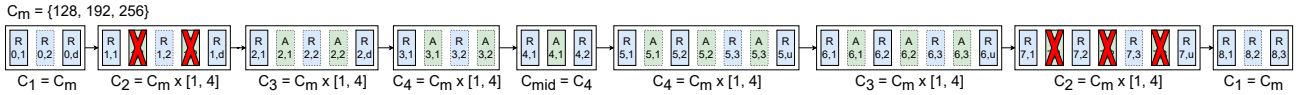


Figure 14. Illustration of the searchable attributes in the SDv1.4 U-Net. Long residual connections are removed from this drawing but still present in all candidate architectures. The attention blocks in Stage 2 are always removed. We do not alter the middle stage but note that its channels must match that of the stage 4 input/output modules. We allow for a global channel value $C_m \in \{128, 192, 256\}$. The number of channels in stages 2-4 is equal to C_m times a multiplier in $[1, 2, 3, 4]$, while the channels of Stage 1 are always C_m. We can also remove attention blocks from the input/output stages 3 and 4 independently of each other, as well as remove at least one residual convolution block per input/output stage. Blocks with dotted borders denote optional layers, while blocks bound by solid lines are mandatory.

same, e.g., if the multiplier for Input Stage 2 is 3, the multiplier for Output Stage 2 must also be 3.

- **Positional embedding** describing which section (input, middle, or output) and stage a node is located in.

After combining the searchable parameters and applying the constraints on C_m and stage-channel multipliers, we are left with a search space containing approximately 800k U-Net candidates. To evaluate a candidate, we first inherit weights from the original SDv1.4, then fine-tune for 20k steps on the Places365 [48] training set, which contains 1.7 million images. We use 2 Nvidia Tesla V100 GPUs with 32GB of VRAM each to fine-tune, with a batch size of 7 per GPU. Then we compute its FID on a held-out Places365 validation subset containing 3k images where the Inpainting masks are predefined and consistent across different architecture evaluations. The whole process can take between 1-2 days depending on candidate architecture. We fine-tune 68 architectures for AutoBuild and Exhaustive Search to learn from.

Prediction with <100 Samples. We provide additional details on how we use AutoBuild to select high-performance architectures. Specifically, each predictor in the ensemble trains for 1000 epochs with batch gradient descent, e.g., the batch size is equal to the size of the training set. The GNN predictor contains 6 message passing layers to accommodate output stages 2, 3 and 4 which can span up to 7 nodes. Since we only have 68 samples and the number of hops can vary between 0 and 6, we do not have enough data to compute the statistics required for Eq. 5. Instead, we normalize subgraph scores to a normal distribution $\mathcal{N}(0, 1)$, then take the top-5 subgraphs per input/output stage[5], dividing $K = 5$ between different subgraph sizes. For example, when selecting subgraphs for Stage 1, which can have 1-2 hops, we will select two 1-hop subgraphs and three 2-hop subgraphs, while for Output Stages 2, 3 and 4, which can span 2-6 hops, we select the best subgraph for each hop-size.

[5]We ignore the Middle Stage from consideration and simply select the channel figuration that will match with the Stage 4 Input/Output subgraphs.

Table 3. Distribution and range of feature categories for SDv1.4 generated by the AutoBuild predictor that attained the highest FE-MLP 0-hop SRCC.

Feature	Distribution	Range
Operation	$\mathcal{N}(0.53, 0.21)$	$[0.00, 0.77]$
Channels	$\mathcal{N}(0.48, 0.04)$	$[0.38, 0.53]$
Section Idx.	$\mathcal{N}(0.52, 0.60)$	$[0.02, 1.30]$
Stage Idx.	$\mathcal{N}(0.43, 0.22)$	$[0.26, 0.79]$

The top-4 architectures are then constructed by finding the top-4 subgraph combinations whose sum of scores is the highest, provided they abide by the aforementioned channel constraints to ensure network functionality. Figures 15 and 16 illustrate the top-4 architectures found by AutoBuild and Exhaustive Search, respectively, annotated with FID.

SDv1.4 Search Space Insights. We provide some additional search space insights from the FE-MLP of the AutoBuild predictor which achieved the highest 0-hop SRCC (0.6464). Table 3 quantifies the distribution and range of importance scores for each feature category. First, we note that the feature which attains the highest value is Section Index. Specifically, input stages are assigned a high score of 1.3, output stages are assigned 0.15 while the Middle Stage is assigned a low value of 0.08, notable as the best architecture found (AutoBuild Architecture 4) allows attention in every input stage it can. In terms of Stage Index, importances are given in descending order as Stage $4 = 0.79$, Stage $2 = 0.41$, Stage $3 = 0.26$ and Stage $1 = 0.26$ which emphasizes the input/output stages closest to the middle, as well as Stage 2, where attention is part of the original model but not permitted in our search space.

Operation importance scores are as follows, in descend-

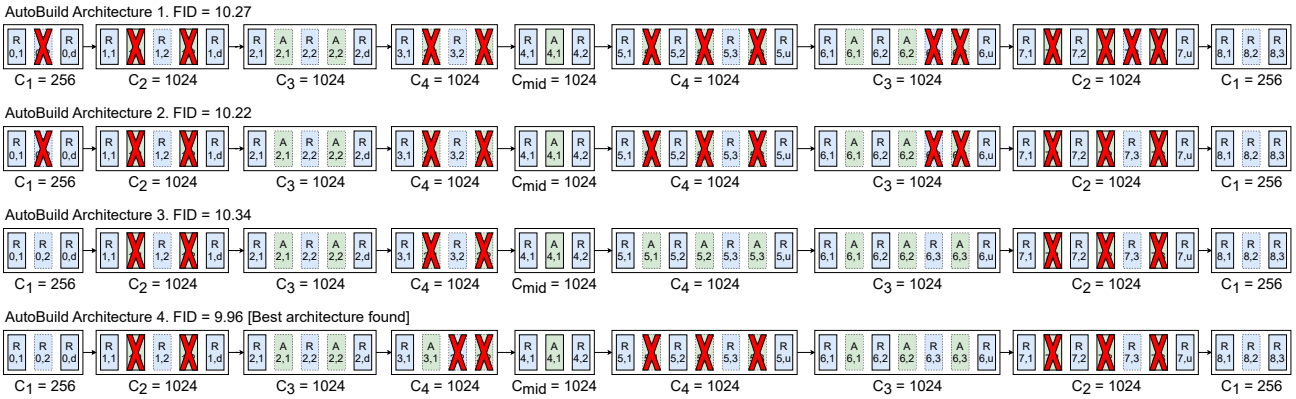


Figure 15. Architectures found by AutoBuild, annotated with FID performance.

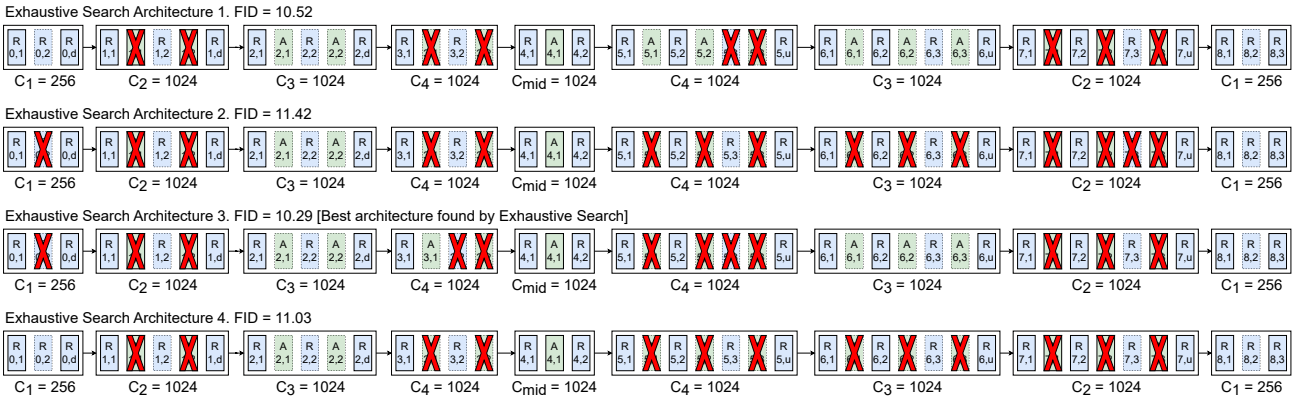


Figure 16. Architectures found by Exhaustive Search, annotated with FID performance.

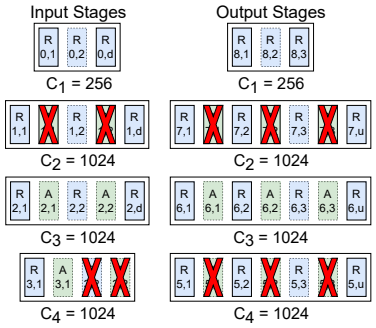


Figure 17. Best AutoBuild SDv1.4 input/output subgraphs.

ing order: Upsample $= 0.77$, Residual Convolution $= 0.63$, Attention $= 0.33$ and Downsample $= 0.00$. This may be a way of compensating for the high score assigned to nodes in the input stages while not placing importance on Output Stage 1, as it is the only output stage that lacks an upsampling operation. Finally, channel size is correlated with importance score: more channels is better, as evidenced by all architectures in Figs. 15 and 16 setting $C_m = 256$ and using the highest-possible multipliers in stages 2, 3 and 4.

Finally, Figure 17 illustrates the best input and output architecture module subgraphs found by AutoBuild for our SDv1.4 macro-search space.

7.10. AutoBuild for Pareto-optimal U-Nets

In addition to the search space and AutoBuild results present in Sections 5.3 and 7.9, we designed a second search space and training regime. The objective of this search space and experiments was to find smaller SDv1.4 U-Net variants that achieved Pareto-optimal performance by minimizing the FID and a hardware-friendliness metric, e.g., GPU latency.

Hardware-aware Search Space & Evaluation. The second U-Net search space is a subset of the first, only containing approximately $100k$ architectures compared to the $800k$ in the original. We now iterate the changes made to differentiate the second search space:

- **U-Net global channel size** C_m: Fixed to always be 256.
- **Stage channel multipliers**: We fix these so that the number of channels is never more than the original SDv1.4 U-Net, e.g., Stage 4 can choose from multipliers in

$[1, 2, 3, 4]$, but Stage 2 can only choose from $[1, 2]$.

- **Number of residual/attention blocks**: Unchanged.

These changes produce a new search space where the original SDv1.4 U-Net is the largest architecture. In terms of fine-tuning and evaluation, we still use the Places365 dataset. Specifically, we use the same $3k$ masked images from the validation set to generate FID scores as before. However, FID for architectures in the first and second search spaces are not strictly comparable as we increased the number of training steps to $30k$, up from $20k$ before.

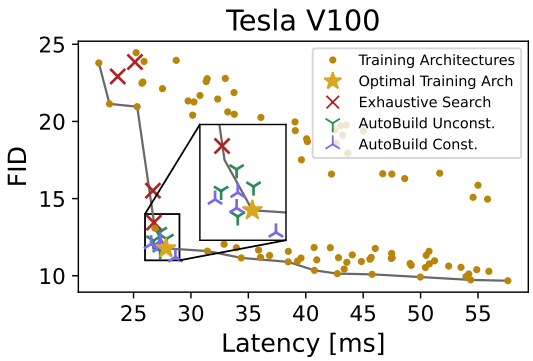
We fine-tune and evaluate 90 random architectures from the second search space. In addition to measuring the FID score, we also consider three hardware metrics: U-Net inference latency in an Nvidia V100 GPU, the number of Floating Point Operations (FLOPs) required for a forward pass, and the number of U-Net parameters.

Evaluation and Results. Our experimental procedure is largely unchanged from Sections 5.3 and 7.9 in that we still consider an **Exhaustive Search** baseline method. However, this time we evaluate two variants of AutoBuild on the second search space, **AutoBuild Unconstrained** and **AutoBuild Constrained**. We differentiate these methods by how they account for module subgraphs with different sizes when constructing high-performance architectures:

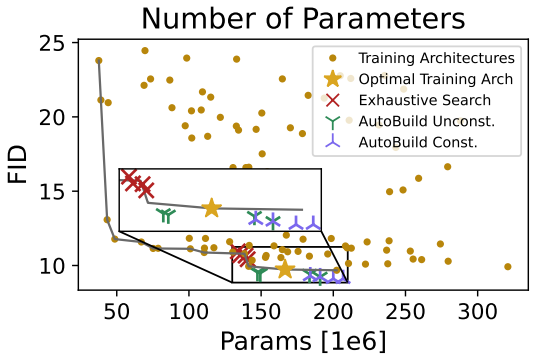
- **AutoBuild Constrained** follows the procedure from Sec. 7.9, ensuring that a variety module subgraphs with different hop sizes are selected when composing the top-K set for a given stage.
- **AutoBuild Unconstrained** does not follow this procedure. Instead, we just normalize all subgraph scores for a given stage into $\mathcal{N}(0, 1)$ and then take the top subgraphs per stage.

Figure 18 illustrates the resultant Pareto frontiers for a trio of hardware metrics. Note how in every case AutoBuild is able to consistently find better U-Net architectures that push beyond the original Pareto frontier formed by the 90 random training architectures. In addition, the architectures found by AutoBuild are generally much closer to the 'Optimal Training Arch', which is the training architecture that maximizes the target objective, than those found by Exhaustive Search.

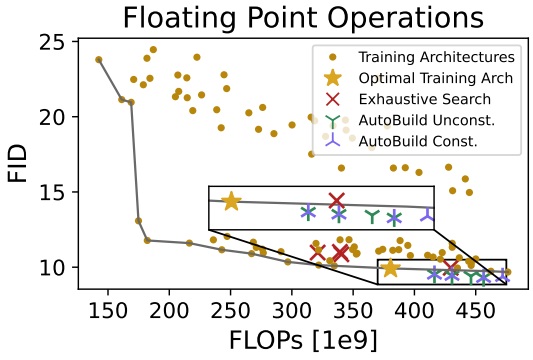
When comparing AutoBuild Constrained and AutoBuild Unconstrained, we find that the 'Constrained' version can more consistently construct architectures that surpass the original Pareto frontier, e.g., on the V100 GPU. A downside of this approach is that architectures found by AutoBuild Constrained actually deviant further away from the Optimal Training Architecture than those constructed by AutoBuild Unconstrained, e.g., in the FLOPs and parameter frontiers. Finally, also common to the FLOPs and parameter Pareto frontiers, is not uncommon for both of these methods to construct the same architectures.



(a) $y = -FID - Latency$



(b) $y = -FID - \frac{Params}{200}$



(c) $y = -FID - \frac{FLOPs}{400}$

Figure 18. FID vs. Hardware-friendliness Pareto frontiers (lower values are better for both) for the second SDv1.4 search space (Sec. 7.10). Subfigure captions refer to the target equations that AutoBuild and Exhaustive Search aim attempt to maximize. 'Training Architectures' refers to the 90 random fine-tuned U-Nets, while the 'Optimal Training Arch' gold star indicates the training architecture that maximizes the target equation.