SPACEEVO: SEARCHING HARDWARE-FRIENDLY SEARCH SPACE FOR EFFICIENT INT8 INFERENCE

Anonymous authors
Paper under double-blind review

ABSTRACT

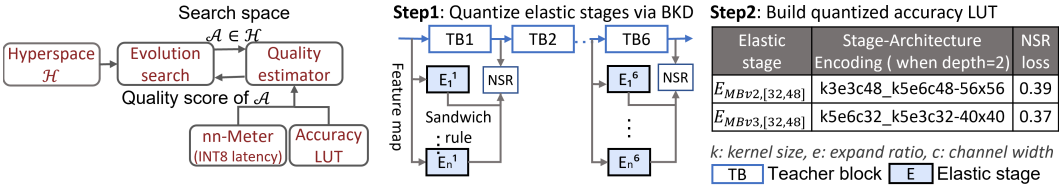
INT8 quantization is an essential compression tool to deploy a deep neural network (DNN) on resource-limited edge devices. While it greatly reduces model size and memory cost, current edge-regime DNN models cannot well utilize INT8 quantization to reduce inference latency. In this work, we find that the poor INT8 latency performance is due to the quantization-unfriendly issue: the operator and configuration (e.g., channel width) choices in a normal model design space lead to *diverse* quantization efficiency and can slow down the INT8 latency. To alleviate this issue, we propose SpaceEvo to efficiently search a novel hardware-aware, quantization-friendly search space, where its top-tier sub-networks achieve both superior quantization efficiency and accuracy. The key idea is to automatically evolve hardware-preferred operators and configurations guided by a search space quality metric, called *Q-T score*. However, naively training a candidate space from scratch for Q-T score evaluation brings prohibitive training cost, making it difficult to evolve search space on large-scale tasks (e.g., ImageNet). We further propose to conduct block-wise training and build INT8 accuracy lookup table to greatly reduce the cost. On diverse devices, SpaceEvo consistently outperforms existing manually-designed search spaces by producing both tiny and large quantized models with superior ImageNet accuracy and hardware efficiency. The discovered models, named SeqNet, achieve up to 10.1% accuracy improvement under the same latency. Our study addressed the hardware-friendly search space design challenge in NAS and paved the way for searching the search space towards efficient deployment.

1 INTRODUCTION

Edge AI, where the deep neural networks (DNNs) inference is fully executed on mobile and client devices, shows rapid growth (blog, 2020) and many advantages such as privacy protection. To deploy DNN models on these resource-limited devices, many quantization approaches have been proposed for different bit-width compression. Among them, INT8 quantization has been widely supported and becomes the dominant on edge devices, reducing 4× model size and memory cost compared to full-precision (FP32) models. However, existing state-of-the-art (SOTA) DNNs often receive little benefit from INT8 quantization, which yields limited speedup on real-world devices. (Fig. 2(ab)).

Conventional deployment process adopts a two-stage design-quantize scheme. It first designs an efficient model with low FLOPs and then quantizes it to 8bit for minimal accuracy loss (Esser et al., 2020). Unfortunately, since FLOPs cannot reflect INT8 quantized latency on real-world devices, this traditional routine may fail to get a good quantized model. We observe that existing compact models with low FLOPs have marginal latency speedup by INT8 quantization, and the quantized latency can be unexpectedly longer than a large model as in Fig. 2(c). Applying Neural Architecture Search (NAS) (Shen et al., 2021; Wang et al., 2020) seems to be an alternative, that searches quantized models for target INT8 latency from a SOTA search space. However, we argue that *prior art search spaces cannot be well applied to quantization on the diverse edge devices*, as the current design can unexpectedly *hurt the INT8 latency*. For instance, Squeeze-and-Excitation (SE) (Hu et al., 2018) and Hardswish are widely-used operators in CNN search space as it improves accuracy with little latency introduced, but their INT8 inference is slower than full-precision inference on VNNI CPU (Table 1). As a result, these prior art search spaces undesirably limit NAS to find better quantized models.

We perform an in-depth study to understand the factors that determine on-device quantization efficiency and how they affect search space design. Our study shows: (1) both operator type and configurations (e.g., channel width) greatly impact the INT8 latency; Improper selections can slow down the INT8 latency. (2) The quantization efficiency varies across different hardware, and the preferred settings can be contradictory. Based on these observations, we argue that designing a *quantization-friendly* search space must consider hardware preferred operators and configurations to guarantee efficiency while maintaining the same level of accuracy as normal search spaces. Moreover, each type of device requires a specialized search space to avoid contradictory preferred designs.



(a) SpaceEvo: the space search framework (b) Build accuracy lookup table with block-wise knowledge distillation (BKD)

Figure 1: (a) The search concept of SpaceEvo is analogous to NAS: we have three major components: hyperspace (search spaces pool), search algorithm and space quality estimator; (b) we adopt block-wise knowledge distillation to greatly reduce model accuracy evaluation cost in a search space.

However, we face the challenge of designing such quantization-friendly search space for the large variety of edge devices, as it requires domain knowledge from both AI and hardware experts to optimize the accuracy and INT8 latency. Recently, (Ci et al., 2021; Chen et al., 2021) can automatically shrink a large space to better search spaces without any human efforts. Inspired by this, the questions naturally arise: *1) Can we design a search space pool and automatically search a good one that is consisting of quantization-friendly operators and configurations for the target device? 2) How to handle with the prohibitive cost caused by quality evaluation of a candidate search space?*

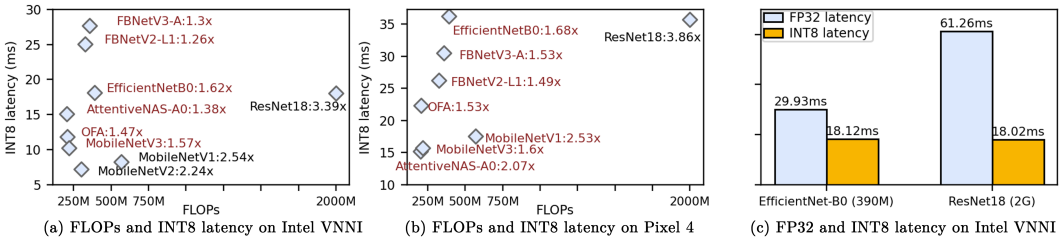
To this end, we propose SpaceEvo, the first to search a quantization-friendly search space for a given device, so that its top-tier sub-networks can achieve optimal INT8 quantization latency and accuracy. We first formulate the space search into neural architecture search (NAS) process as shown in Fig. 1(a). Specifically, we factorize and encode a search space into a sequence of elastic stages, which have flexible operator types and configurations. Then, we design a large hyperspace to include many candidate search spaces and leverage aging evolution (Real et al., 2019) to perform random elastic stage mutations for search space evolution. The evolution process is guided by *Q-T* score, which is proposed to evaluate the INT8 accuracy-latency quality of top-tier sub-networks in a search space. However, Q-T score requires expensive accuracy evaluation for numerous sub-networks. Therefore, instead of naively training a candidate search space from scratch, we further propose a *block-wise quantization scheme* to build a quantized accuracy lookup table in Fig. 1(b). This significantly reduces the training and evaluation cost, while providing effective accuracy rankings among search spaces. Finally, we train a quantized-for-all supernet over the searched space containing a variety of well-quantized models with adaptive sizes and latency. An evolutionary search (Cai et al., 2020) with an INT8 latency predictor is performed to get Pareto-frontier INT8 models. The specialized models can be directly deployed on target hardware without additional training or quantization cost.

Extensive experiments on ImageNet and two popular edge devices (Pixel 4 CPU and Intel VNNI CPU) demonstrate that our searched spaces consistently produce superior INT8 quantized models than the existing manually-designed search spaces with much higher accuracy, lower latency and better speedups. Code and models will be released. In summary, we make the following contributions:

- We systematically study the INT8 quantization efficiency on real-world edge devices and find that the choices of operator types and configurations in a quantized model can significantly impact the INT8 inference latency, leaving a huge room for quantized model design optimization.
- Motivated by our study, we propose SpaceEvo, the first space search algorithm that evolves a quantization-friendly search space producing superior quantized models under desired INT8 latency constraints. To reduce the extremely expensive search cost, we further propose a novel block-wise quantization scheme to build accuracy lookup table.
- We demonstrate the effectiveness and efficiency of SpaceEvo on two edge devices and ImageNet dataset. Our discovered model, SeqNet@vnni-A4 achieves 80.0% accuracy on ImageNet, which is 3.3ms faster with 1.8% higher accuracy than FBNetV3-A. SeqNet@pixel4-A1 runs 2.1× faster than EfficientNetB0 with 0.9% higher accuracy. Moreover, SpaceEvo produces superior tiny models, achieving up to 10.1% accuracy improvement over the tiny ShuffleNetV2x0.5 (41M FLOPs, 4.3ms).

2 On-device Quantization Efficiency Analysis

Most existing works optimize the quantized models through reducing FLOPs. While such metric does not reflect the real on-device latency for quantized models. To understand what factors and design choices slow down the INT8 latency, we conduct a comprehensive study on two widely-used edge devices: an Intel CPU device supported with VNNI instructions and onnxruntime (abbr Intel VNNI) and a Pixel 4 phone CPU with TFLite 2.7 (abbr Pixel 4). We reveal key observations as follows:



(a) FLOPs and INT8 latency on Intel VNNI (b) FLOPs and INT8 latency on Pixel 4 (c) FP32 and INT8 latency on Intel VNNI

Figure 2: INT8 latency and speedups (annotated) for SOTA models. FLOPs and FP32 latency are not good indicators of INT8 latency; Compact models have very limited quantization speedup ($\sim$1.5$\times$).

Observation 1: *FLOPs and FP32 latency are bad indicators of INT8 latency. Existing low-FLOPs models achieve marginal latency speedup.* To deploy a model on edge devices, a common belief is that a compact model with low FLOPs or FP32 latency is preferred than a larger model. However, our comparison in Fig. 2 between models with low and high FLOPs, shows that neither FLOPs nor FP32 latency can be a good indicator of INT8 latency. As shown in Fig. 2(c), a very large model (ResNet18) can be even faster than a compact model (EfficientNetB0) after quantization. Moreover, the recent SOTA compact models searched by FBNets (Wan et al., 2020; Dai et al., 2021), OFA (Cai et al., 2020) and AttentiveNAS (Wang et al., 2021b) all have only $\sim$1.5$\times$ speedup by INT8 inference, suggesting that only optimizing FLOPs and FP32 latency can not lead to lower INT8 latency.

Observation 2: *The choice of operators' type and configurations greatly impacts the INT8 latency.*

Table 1: Avg. INT8 latency speedup (refer to FP32) of major operators in MobileNetV3.

Operator	Intel VNNI	Pixel4
Conv	2.6$\times$	2.5$\times$
DWConv	1.2$\times$	2.0$\times$
SE	0.7$\times$	1.4$\times$
Hardswish	0.7$\times$	0.5$\times$
Swish	0.8$\times$	2.1$\times$

While edge regime models require a small FLOPs ($\leq$600M FLOPs), the prior art search spaces in recent NAS works are MobileNetV2 or MobileNetV3 based chain-structure, where a search space is comprised with a sequence of blocks (stages). The block type is *fixed* to the MBConv and is allowed to search from a *handcraft range of configurations* including kernel size, expansion ratio, channel width and depth. In particular, these handcraft configurations are designed with human wisdom. For instance, many works (Cai et al., 2019; Wang et al., 2021b) observe that mobile CNNs prefer deeper depths and narrower channels, and manually set small channel numbers but large depths in the search space.

However, we find that many block type and configuration choices in current search space unexpectedly slow down the INT8 latency. We first study the operator type impact in Table 1. SE and Hardswish are lightweight operators in edge regime search spaces, but their INT8 inference becomes slower on Intel VNNI. Compared to Conv, DWConv can greatly reduce the FLOPs, but it benefits less from INT8 quantization. Besides the operator type, the configuration choices also determine the quantization efficiency. Fig. 3 shows the speedups of Conv1$\times$1 under various channel numbers. Results suggest that small channel widths in OFA and AttentiveNAS spaces cannot benefit well from quantization. In contrast, SpaceEvo can automatically search a search space with larger channel widths to better utilize hardware capability.

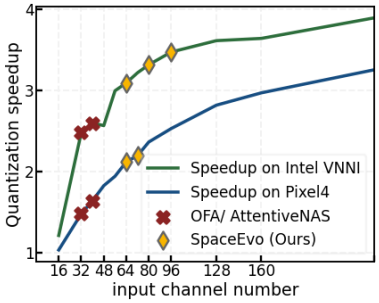


Figure 3: Conv1x1 speedups under various channel numbers. Config: HW=28, C_{out}=4xC_{in}(expand=4).

Observation 3: *Quantization-friendly settings are diverse and contradictory across devices.*

The real-world edge devices are equipped with *(i)*: diverse processors, such as ARM CPU, Intel CPU, and GPU, that have fundamentally different hardware designs; *(ii)* diverse inference engines, such as TFLite (Google, 2022) and Onnxruntime (Microsoft, 2022), that have different optimizations. To deal with the huge diversity, many NAS works (Cai et al., 2019; 2020) use one handcraft search space for all hardware and conduct latency-aware search. However, this does not apply to INT8 inference on diverse devices, because each device has its own set of quantization-friendly operators as shown in Table 1. Constructing a huge search space to cover all devices' preferred settings can be an intuitive solution. However, the very large search space has been demonstrated with prohibitive search cost and convergence failure issues (Ci et al., 2021; Zhang et al., 2020). Moreover, our two devices show some contradictory behaviours. For instance, Swish achieves a 2$\times$ speedup on Pixel4, but it's a

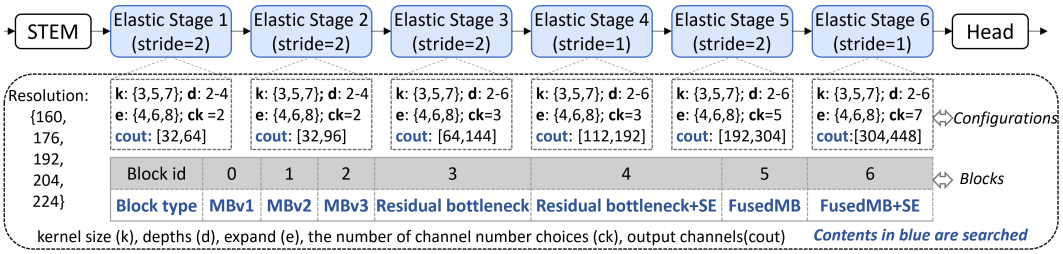


Figure 4: The illustration of our hyperspace. A sampled search space is encoded by a sequential elastic stages. An elastic stage can select its block type and channel number list in evolution search.

quantization-unfriendly operator on VNNI with a 0.8× slowdown. Thus, we argue that each device requires a specialized search space to achieve the optimal quantized latency-accuracy trade-off.

3 METHODOLOGY

Motivated by the above observations, we propose SpaceEvo to search a quantization-friendly space with optimal operator type and configurations for a given device. Our approach is based on two main ideas: *(i)* we formulate the space search into the neural architecture search process; *(ii)* we introduce the block-wise quantization scheme to tackle the challenge of huge search space evaluation cost.

3.1 PROBLEM FORMULATION

Search space factorization into elastic stages. In our work, we consider the chain-structured search space (Cai et al., 2020; Yu et al., 2020; Wang et al., 2021b). It can be factorized as a sequence of STEM, head and N searchable stages, where each stage is allowed to search from a range of configurations c (e.g., kernel size, channel number, depth) for a block type b. This design can effectively reduce the search complexity. Without loss of generality, we define a stage structure as elastic stage $E_{b,c}$. Given a search space $\mathcal{A}$ with N stages, it can be modularized as $STEM \circ E^1_{b,c}, ... \circ E^N_{b,c} \circ head$.

Problem definition. Operator type b and configuration c are two crucial objectives when searching quantization-friendly search space. Through the definition of elastic stage, the task of space search then is simplified to find a search space with the optimal elastic stages. We formulate our problem as:

$$\mathcal{A}(E^1_{b,c} \circ E^2_{b,c} \circ ... \circ E^N_{b,c})^* = \underset{E^i_{b,c} \in \mathcal{H}^i}{\arg\max}\, \mathcal{Q}(\mathcal{A}(E^1_{b,c} \circ E^2_{b,c} \circ ... \circ E^N_{b,c}), T); \tag{1}$$

where $\mathcal{A}(\cdot)$ denotes the search space, and $E^i_{b,c}$ is the i^{th} elastic stage of $\mathcal{A}(\cdot)$. $\mathcal{H}$ denotes the hyperspace and $\mathcal{Q}$ is a measurement of search space quality. Given the constraints T (i.e., a set of targeted quantized latency), SpaceEvo aims to find the optimal elastic stages $(E^1_{b,c} \circ E^2_{b,c} \circ ... \circ E^N_{b,c})^*$ from the 1^{st} to N^{th} stage for $\mathcal{A}^*$ that has the maximum quality score $\mathcal{Q}$: the top-tier quantized models can achieve best accuracy under the constraint T. Fig. 1(a) illustrates the overall process. In this work, we focus on the latency of INT8 quantized models. Our approach can be generalized to lower bits once they are supported on commercial devices.

3.2 SEARCHING THE SEARCH SPACE

Hyperspace $\mathcal{H}$. Analogous to NAS, hyperspace $\mathcal{H}$ defines which search space a search algorithm might discover in principle. We factorize a search space into elastic stages and then search the optimal architecture (i.e., operator type and configurations) for each elastic stage. As shown in Fig. 4, a search space can be encoded by N=6 sequential elastic stages along with STEM and head layers. In our work, we search the following two dimensions for an elastic stage:

- Block type b: MBv1 (Howard et al., 2017), MBv2 (Sandler et al., 2018), MBv3 (Howard et al., 2019), residual bottleneck (He et al., 2016), residual bottleneck with SE, FusedMB (Tan & Le, 2021) and FusedMB with SE. Among them, residual bottleneck and FusedMB are consisting of Conv and thus are quantization-friendly blocks on our evaluated devices; the efficiency of MB blocks relies on the device. For instance, DWConv and SE are less quantization-efficient on Intel VNNI.
- Output channel width list $cout$. In Section 2, we observe that quantized models can better utilize edge devices under a larger channel number setting. However, directly increasing the channel numbers will also lead to longer latency. Therefore, instead of manual configurations, we search the optimal stage-wise channel width list $cout$: $\{w^*_{min}, ..., w^*_{max}\}$, which provides different channel width choices for final neural architecture search. Specifically, as described in Fig. 4, we predefined

a wide range of $[w_{min}, w_{max}]$ by enlarging the channel widths in existing search spaces, and allow each elastic stage to choose $cout$ from $[w_{min}, w_{max}]$. To better utilize on-device INT8 quantization, the channel widths in a model is constrained to be divisible by 8 on Pixel4 and 16 on Intel VNNI (refer to Appendix B). For example, given a predefined range of [32, 64] and ck=2, the possible channel number lists $cout$ can be {32,48} or {48, 64} for Intel VNNI.

Besides channel widths, other configuration dimensions (e.g., kernel size) also impact a model's quantized latency. However, searching all dimensions leads to a large amount of choices in a stage, which exponentially enlarges the hyperspace size. Fortunately, we observe that other dimensions usually have a small space (e.g., kernel size selects from {3,5,7}). It's easy to find the optimal value for a model in the final NAS process. Therefore, we follow existing practices to configure the choices of kernel size, depth, and expand ratios. Appendix C.1 provides more details.

Our hyperspace has a distinct advantage of balancing the trade-off between accuracy and quantization efficiency. Suppose that a stage has m choices of channel widths, there are 7 (blocks)$\times$m selections for each elastic stage. In total, for a typical search space with N=6 stages, the hyperspace has $\sim 10^9$ candidate search spaces, which is extremely large and poses challenges for efficient search.

Evolutionary space search. We leverage aging evolution (Real et al., 2019) to efficiently search the large hyperspace. We first randomly initialize a population of P search spaces, where each sampled space is encoded as $(E^1_{b,c} \circ E^2_{b,c} \circ ... \circ E^N_{b,c})$. Each individual is rapidly evaluated with a Q-T quality score (see next section). After this, evolution improves the initial population in mutation iterations. At each iteration, we sample S random candidates from the population and select the one with highest score as the *parent*. Then we alternately mutate the *parent* for block type and widths to generate two *children* search spaces. For instance, suppose the i^{th} stage $E^i_{b,c}$ is selected for mutation, we first randomly modify its block type and produce $E^i_{b^*,c}$ for child 1, then we mutate the widths and produce E^i_{b,c^*} for child 2. Once the children are constructed, we evaluate their Q-T scores by equation 2. We add the children to current population and remove the oldest two for next iteration. After all iterations finish, we collect all the sampled space and select the one with best score as the final search space.

Quality estimator Q-T score. Q-T score serves as the metric to measure the quality of a search space, and guide the evolution search process. Since our ultimate goal is to search best quantized models from the searched space, we treat a space with good quality if its **top-tier** sub-networks achieve optimal quantized accuracy under constraints T. We use **multiple INT8 latency constraints** to measure a space's quality, as real-world applications usually have different deployment requirements. For a sampled space $\mathcal{A}$ and a set of quantized latency constraints $T_{1,...,n}$, Q-T score is the sum of each constraint: $\mathcal{Q}(\mathcal{A}, T_{1,...,n})$= $\mathcal{Q}(\mathcal{A}, T_1)$+$\mathcal{Q}(\mathcal{A}, T_2)$+..., $\mathcal{Q}(\mathcal{A}, T_n)$, where $\mathcal{Q}(\mathcal{A}, T_i)$ is defined as:

$$\mathcal{Q}(\mathcal{A}, T_i) = \mathbb{E}_{\alpha \in \mathcal{A}, LAT(\alpha) \leq T_i}[Acc_{int8}(\alpha)] \tag{2}$$

where α denotes a top-tier (best searched) sub-network in $\mathcal{A}$ and $Acc_{int8}(\alpha)$ is its top-1 quantized accuracy evaluated on ImageNet validation set, $LAT(\alpha)$ predicts the quantized latency (in next section). In our experiments, we randomly sample 5k sub-networks and select top 20 that under the latency constraints as the top-tier models to approximate the expectation term.

To measure the Q-T score, the most accurate evaluations are to get accuracy by training a supernet (search space) from scratch (Ci et al., 2021; Chen et al., 2021) and get latency by on-device measurement. However, it's impractical to conduct large-scale search due to the prohibitive cost. For example, it costs more than 10 days to train a supernet on 8 V100 GPUs (Yu et al., 2020). To address this issue, we introduce block-wise knowledge distillation scheme and latency predictor in next section.

3.3 Efficient Q-T score evaluation

Block-wise quantization with knowledge distillation. Block-wise knowledge distillation (BKD) is firstly proposed in DNA (Li et al., 2020a) and then further improved in DONNA (Moons et al., 2021). It uses block-wise representation of existing models (teacher) to supervise a student (a stage block of the search space). Since a stage size is much smaller than search space size, the training time is greatly reduced. Inspired by this, we adopt BKD to train all the elastic stages in our hyperspace.

Fig. 1(b) illustrates the BKD process. We use EfficientNet-B5 as the teacher, and separately train each elastic stage to mimic the behavior of corresponding teacher block by minimizing the NSR loss (Moons et al., 2021) between their output feature maps. Specifically, the i^{th} stage receives the output of $(i-1)^{th}$ teacher block as the input and is optimized to predict the output of i^{th} teacher block with NSR loss. At each training step, we adopt sandwich rule (Yu et al., 2020) to sample four

Table 2: ImageNet results compared with SOTA quantized models on the Intel VNNI CPU.

Model	Acc% **INT8**	VNNI Latency **INT8**	**speedup**	Acc% FP32	FLOPs
MobileNetV3Small	66.3	4.4 ms	1.1×	67.4	56M
SeqNet@vnni-A0	**74.7**	**4.4 ms**	**2.0**×	74.8	163M
MobileNetV2	71.4	7.3 ms	2.2×	72.0	300M
ProxylessNAS-R	74.6	8.8 ms	1.8×	74.6	320M
OQAT-8bit	74.8	9.8 ms	1.8×	75.2	214M
MobileNetV3Large	74.5	10.3 ms	1.5×	75.2	219M
OFA (#25)	75.6	11.2 ms	1.5×	76.4	230M
SeqNet@vnni-A1	**77.4**	**8.8 ms**	**2.4**×	77.5	358M
APQ-8bit	73.6	15.0 ms	1.5×	73.6	297M
AttentiveNAS-A0	76.1	15.1 ms	1.4×	77.3	203M
OQAT-8bit	76.3	14.9 ms	1.7×	76.7	316M
EfficientNet-B0	76.7	18.1 ms	1.6×	77.3	390M
SeqNet@vnni-A2	**78.5**	**14.1 ms**	**2.4**×	78.8	638M
APQ-8bit	74.9	20.0 ms	1.5×	75.0	393M
OQAT-8bit	76.9	19.5 ms	1.6×	77.3	405M
AttentiveNAS-A1	77.2	22.4 ms	1.4×	78.4	279M
AttentiveNAS-A2	77.5	22.5 ms	1.3×	78.8	317M
SeqNet@vnni-A3	**79.5**	**18.9 ms**	**2.6**×	79.6	981M
FBNetV2-L1	75.8	25.0 ms	1.2×	77.2	325M
FBNetV3-A	78.2	27.7 ms	1.3×	79.1	357M
SeqNet@vnni-A4	**80.0**	**24.4 ms**	**2.4**×	80.1	1267M

Table 3: ImageNet results compared with SOTA quantized models on the Pixel 4 smartphone.

Model	Acc% **INT8**	Pixel4 Latency **INT8**	**speedup**	Acc% FP32	FLOPs
MobileNetV3Small	66.3	6.4 ms	1.3×	67.4	56M
SeqNet@pixel4-A0	**73.6**	**5.9 ms**	**2.1**×	73.7	107M
MobileNetV2	71.4	16.5 ms	1.9×	72.0	300M
ProxylessNAS-R	74.6	18.4 ms	1.8×	74.6	320M
MobileNetV3Large	74.5	15.7 ms	1.5×	75.2	219M
APQ-8bit	74.6	14.9 ms	2.0×	74.4	340M
OFA (#25)	75.6	14.8 ms	1.7×	76.4	230M
OQAT-8bit	75.8	15.2 ms	1.9×	76.2	287M
AttentiveNAS-A0	76.1	15.2 ms	2.0×	77.3	203M
SeqNet@pixel4-A1	**77.6**	**14.7 ms**	**2.2**×	77.7	274M
APQ-8bit	75.1	20.0 ms	1.9×	75.1	398M
OQAT-8bit	76.5	20.4 ms	1.8×	76.8	347M
AttentiveNAS-A1	77.2	21.1 ms	2.0×	78.4	279M
AttentiveNAS-A2	77.5	22.7 ms	2.0×	78.8	317M
SeqNet@pixel4-A2	**78.3**	**19.4 ms**	**2.3**×	78.4	402M
FBNetV2-L1	75.8	26.7 ms	1.5×	77.2	325M
OQAT-8bit	77.0	29.9 ms	1.7×	77.2	443M
FBNetV3-A	78.2	30.5 ms	1.5×	79.1	357M
SeqNet@pixel4-A3	**79.5**	**30.8 ms**	**2.1**×	79.5	591M
EfficientNet-B0	76.7	36.4 ms	1.7×	77.3	390M
SeqNet@pixel4-A4	**79.9**	**35.5 ms**	**2.2**×	80.0	738M

sub-paths from the target elastic stage to improve the training efficiency. Each elastic stage is firstly trained for 5 epochs and then performed 1 epoch LSQ+ (Bhalgat et al., 2020) for INT8 quantization.

The key insight behind BKD is that block-level quality (i.e., per-stage NSR loss on validation set) can be used to rate model accuracy. Thus, we follow DNA to estimate the quantized loss of a sampled model by summing up the NSR loss of all stages. To reduce the evaluation cost, we build a quantized loss lookup table (see Fig. 1 (b)). Specifically, we randomly sample 2k architectures from each elastic stage and evaluate their stage-wise NSR loss. These stage-architectures and losses are stored in the lookup table. When evaluating the Q-T score, we measure a sampled model's quantized loss by rapidly looking up its all stage-architectures from the table. We inverse the measured loss to approximate the quantized accuracy. In our work, the BKD and lookup table construction can be sped up in a parallel way and finished in 1 day, which amounts a one-time cost before the space search.

nn-Meter for INT8 latency prediction. Pevious works build a latency lookup table to measure all the stage-architectures' latencies in a search space. In our work, it requires 0.2 millions of measurements on each device, which is extremely expensive. To reduce the cost, we use nn-Meter (Zhang et al., 2021) to build kernel-level (i.e., a fusion of multiple fused operators) latency regressors, which can accurately predict latency for a kernel with **arbitrary** configurations. We predict the total INT8 latency of a model by the latency sum of all kernels.

3.4 INT8 MODEL DEPLOYMENT WITH QUANTIZATION-FRIENDLY SPACE

Once SpaceEvo discovers a quantization-friendly search space for the target device, we perform two-stage neural architecture search to derive the Pareto-frontier quantized models. The first stage is to train a *quantized-for-all* supernet. We start by pretraining a full-precision supernet without quantizers on ImageNet. We adopt the sandwich sampling rule and inplace distillation introduced in (Yu et al., 2020). Then, we perform quantization-aware training (QAT) on the trained supernet. This step follows the same training protocol (i.e., sandwich rule and inplace distillation). To better quantize MB-based blocks, we use LSQ+ as the QAT algorithm. In the second stage, we search Pareto-frontier models from the quantized-for-all supernet with nn-Meter INT8 latency predictor. Note all the quantized models can be directly deployed without retraining or finetuning.

4 EVALUATION

4.1 EXPERIMENT SETUP

We conduct space search for two popular edge devices. The INT8 latency constraints are {8, 10, 15, 20, 25} *ms* for Intel VNNI, and {15, 20, 25, 30, 35} *ms* for Pixel4. For each device, we search 5k search spaces in total. The population size P is 500 and sample size S is 125. After the search finishes, we train a quantized-for-all supernet over the searched space. The full-precision supernet follows a similar training receipt and hyperparameter setting in BigNAS (Yu et al., 2020) and AlphaNet (Wang et al., 2021a). For the supernet QAT, we use a 10× smaller initial learning rate and set 50 epochs for

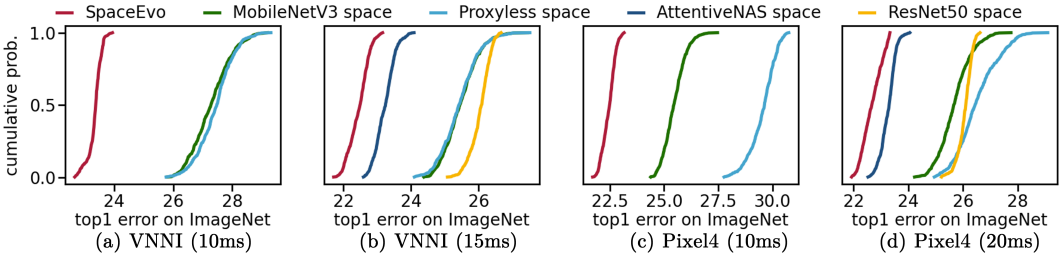


Figure 5: Model error distribution comparison of different search spaces. For a target latency constraint on each device, we randomly sample 512 models to characterize the error distribution.

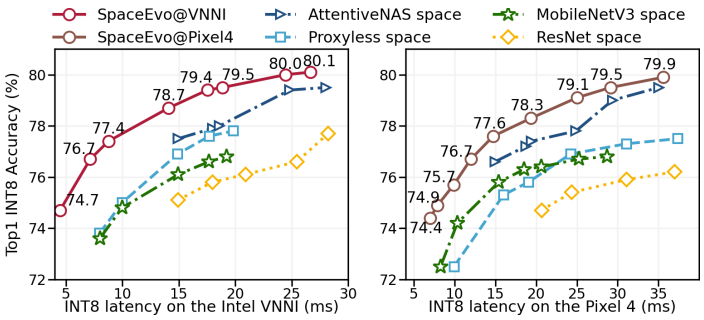


Figure 6: Comparison of best searched INT8 models from different spaces. Our searched spaces deliver a much wider latency range of better models than existing manually-designed search spaces.

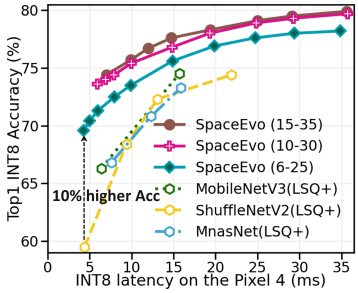


Figure 7: Space search under diverse INT8 latency constraints. SpaceEvo(6-25) delivers superior tiny INT8 models.

training. For the final INT8 quantized model search, we use the evolutionary search in OFA (Cai et al., 2020) to search 5k models for a given INT8 latency. We list out detailed training settings in Appendix D. In the following, we refer to the two searched spaces as *SpaceEvo@VNNI* and *SpaceEvo@Pixel4*, the searched model families are *SeqNet@vnni* and *SeqNet@pixel4*.

4.2 Main Results on ImageNet

Comparison with SOTA quantized models. We compare SeqNet with two baselines: (1) *prior art manually-designed and NAS-searched models on ImageNet*; and (2) *quantization-aware NAS*. For baseline (1), we collect official pre-trained FP32 checkpoints and conduct LSQ+ QAT to get the quantized accuracy. The hyperparameter settings follow the original LSQ+ paper, except that we set a larger epoch of 10 to achieve better accuracy. The latency numbers are measured on our devices. For (2), we compare with strong baselines including APQ (Wang et al., 2020) and OQAT (Shen et al., 2021). Specifically, we limit APQ to search for the fixed 8bit (INT8) models. Since OQAT has no 8bit supernet checkpoint, we follow the official source code and conduct supernet QAT for 50 epochs. During the INT8 model search, we use nn-Meter as the INT8 latency predictors for fair comparison.

Table 2 and Table 3 summarize the results on two edge devices. Remarkably, our searched model family, SeqNet significantly outperform SOTA efficient models and quantization-aware NAS searched models, with higher INT8 quantized accuracy, lower INT8 latency and better speedups. Without finetuning, our tiny models - SeqNet@vnni-A0 and SeqNet@pixel4-A0 achieve 74.7% and 73.6% top1 accuracy on ImageNet, which is 8.4% and 7.3% higher than MobileNetV3-Small (56M FLOPs) while maintaining the same level quantized latency. For larger models, SeqNet@vnni-A4 (80.0%) outperforms FBNetV3-A with 1.8% higher accuracy while runs 3.3ms faster. In particular, to achieve the same level accuracy (i.e., around 77.2%), AttentiveNAS-A1 has 22.4ms latency while SeqNet@vnni-A1 (77.4%) only needs 8.8 ms (2.6 $\times$ faster). More importantly, our searched models can better utilize the INT8 hardware optimizations: the latency speedups compared to full-precision inference are all $\geq 2\times$, and this leaves room to search larger models with higher accuracy.

Comparison with SOTA search spaces. The performance gains of SeqNet in tables 2 and 3 come from our quantization-friendly search spaces. To further demonstrate it, we compare both model distributions and top-tier models' accuracy with prior art search spaces including: (1) MobileNetV3, ProxylessNAS and AttentiveNAS search spaces that are manually designed for mobile-regime models; and (2) ResNet50 search space proposed by OFA that is a quantization-friendly space on our two devices. For fair comparison, we use one supernet training and QAT receipt for all search spaces.

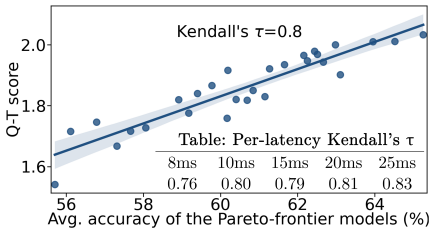


Figure 8: Q-T score effectiveness (Kendall's τ) on ranking search spaces.

Method	Op	Width	Best quantized models 10ms	15ms	20ms	30ms	36ms
Manual Design†	-	-	-	76.6	77.2	79.0	79.5
SpaceEvo-op	search	fix*	75.0	76.6	77.8	78.6	78.8
SpaceEvo-width	fix*	search	75.4	77.4	78.0	79.1	79.5
SpaceEvo	search	search	**75.7**	**77.6**	**78.3**	**79.5**	**79.9**

Table 4: Different space search methods and their best resulting quantized models on Pixel4. †: we compare with the SOTA mobile-friendly AttentiveNAS space. *: the search dimension use the same settings in AttentiveNAS.

We use nn-Meter as the INT8 latency predictor for model sampling and search. For all experiments, search space is the only difference.

Model distributions. We first sample a set of quantized models from each search space to characterize the model error distributions (Radosavovic et al., 2019). Specifically, we focus on comparing low latency models and set two tight quantized latency constraints in Fig. 5. For each constraint, we randomly sample 512 models from the quantized-for-all supernet and evaluate their quantized accuracy on ImageNet. As shown in Fig. 5, compared to SOTA search spaces, we have significantly better model distributions in terms of quantized accuracy while under a same latency.

Pareto-frontier models. We conduct evolutionary search to compare the resulting top-tier models of each search space. Specifically, for each space, we measure its min and max latency and search the Pareto-frontier models within that range. As shown in Fig. 6, SpaceEvo@VNNI and SpaceEvo@Pixel4 consistently deliver superior quantized models than state-of-the-art search spaces. Under the same-level latency, the best quantized models from SpaceEvo@VNNI outperform the existing state-of-the-art search spaces with +0.7% to +3.8% (+0.4% to +3.2% on Pixel4) higher accuracy. Moreover, our search space is the only that is able to deliver superior quantized models under both extremely low (only ∼5ms) and large latency constraints.

SpaceEvo under diverse latency constraints. We extensively study the effectiveness of SpaceEvo under different latency constraints. Specifically, we perform space search under two tight constraints of {10, 15, 20, 25, 30}ms and {6, 10, 15, 20, 25}ms on Pixel4. The results are shown in Fig. 7. Our proposed method can handle the diverse latency requirements and produce high-quality spaces. As expected, the searched spaces under 10-30ms and 6-25ms have much more low-latency quantized models. To further verify the effectiveness of these low-latency models, we compare with existing SOTA tiny models. Significantly, even under the extremely low latency constraints of 6-25 ms, our searched space delivers very competitive tiny quantized models. Compared to the smallest model ShuffleNetV2x0.5, we can achieve +10.1% higher accuracy under the same latency of 4.3 ms.

Search space design implications. We now summarize our learned practice and implications for designing quantization-friendly search space. We notice that the searched spaces show different preferences when targeting different devices: (i) All stages should use much wider channel widths compared to existing manually-designed spaces on the VNNI, while only early stages prefer wider channels on Pixel 4. (ii) VNNI space prefers MBv2, MBv3 and FusedMB blocks. Pixel4 prefers MBv2 and MBv3 under large latency constraints. When given lower latency constraints, Pixel4 space prefers to use more MBv1 and Residual blocks. The searched space details are in Appendix C.2.

4.3 Ablation Study

Q-T score effectiveness. Q-T score is crucial as it guides the space evolution process. To evaluate its effectiveness, we randomly sample 30 search spaces, and measure the rank correlation (Kendall's τ) between their Q-T score and their actual Pareto-frontier models' accuracies. Specifically, we use VNNI as the test device and set a same latency constraints of {8, 10, 15, 20, 25}ms. For each sampled space, we train it from scratch for 50 epochs, and conduct evolutionary search to get the Pareto-frontier models' accuracies. As shown in Fig. 8, the Kendall's τ between the Q-T score and the actual Pareto-frontier models' accuracies is 0.8, which indicates a very high rank correlation.

Space search effectiveness. In Section 2, we conclude that operator type and configuration are two key factors impacting INT8 latency, which is our motivation of SpaceEvo. To verify the effectiveness, we create two strong baselines based on the SOTA mobile-regime AttentiveNAS search space: (1) SpaceEvo-op: we fix each elastic stage's width to AttentiveNAS space, then allow each elastic stage to search for the optimal operator type; and (2) SpaceEvo-width: we fix all elastic stages'

Table 5: Comparison with SOTA space search methods. We measure the time cost on 8 Tesla V100 GPUs. N denotes the number of deployment scenarios.

Method	Model	Operators	Configurations	Hardware-aware	Search iterations	Total Cost GPU hours
NSE (Ci et al., 2021)	FP32 CNN	searchable	fixed	✗	6	$560N$
S3 (Chen et al., 2021)	FP32 Transformer	fixed	searchable	✗	3	$960N$
SpaceEvo (Ours)	Quantized CNN	searchable	searchable	✓	**5000**	**24+1*N***

block types to AttentiveNAS space, then search for the optimal width. Table 4 reports the space comparison between different search methods on the Pixel4. By searching both operator type and width, SpaceEvo finds the optimal search space where its best searched quantized models achieve the highest accuracy under all latency constraints. Moreover, even searching for one dimension, SpaceEvo-op and SpaceEvo-width outperform AttentiveNAS space under small latency constraints.

Search cost. Finally, we measure the search cost of SpaceEvo and compare with state-of-the-art space shrinking methods. The cost is evaluated as the time to obtain search spaces for N deployment scenarios (one scenario is denoted by a given range of latency constraints). As shown in Table 5, SpaceEvo can search a space under a given constraint (N=1) in 1 day, which saves cost by 22$\times$ and 38$\times$ compared to NSE (Ci et al., 2021) and S3 (Chen et al., 2021), respectively. NSE and S3 are time-consuming, as they need to train each search space from scratch for quality evaluation. In contrast, SpaceEvo only requires a preprocess of accuracy lookup table construction (24 hours), which amounts one time cost. Thus, SpaceEvo is lightweight and feasible for real-world usage.

5 Related Works

Quantization has been widely used for efficiency in deployment. Extensive efforts can be classified into post-training quantization (PTQ) (Nagel et al., 2019; Banner et al., 2019) and quantization-aware training (QAT) (Jacob et al., 2018; Louizos et al., 2019; Esser et al., 2020; Bhalgat et al., 2020). QAT generally outperforms PTQ in quantizing compact DNNs to typical 8bit and very low-bit (2, 3, 4bit) by finetuning the quantized weights. Despite their success, traditional quantization methods focus on minimizing accuracy loss for a given pre-trained model, but ignore the real-world inference efficiency. In our work, we consider both the quantized accuracy and latency on diverse edge devices.

Neural Architecture Search. Early NAS works focus on automating neural network design for SOTA accuracy. Recent hardware-aware NAS methods (Cai et al., 2019; 2020; Dai et al., 2021) consider both accuracy and hardware efficiency by introducing latency predictors. However, these works consider the latency of FP32 models, leading to a big gap and performance degradation for INT8 quantized models. (Wang et al., 2019; Wu et al., 2019; Guo et al., 2020; Wang et al., 2020; Cai & Vasconcelos, 2020) formulates mixed-precision problem into NAS to search layer bit-width with a given architecture. Recently, (Shen et al., 2021; Bai et al., 2021) train a quantized-for-all supernet to search both architecture and bit-width. However, little attention is paid on optimizing quantized model latency on real-world devices. Through searching quantization-friendly search space, our discovered quantized models can achieve both high accuracy and low latency.

Search Space Design. Starting from (Cai et al., 2019), the manually-designed MBConv-based space becomes the dominant in most NAS works (Cai et al., 2019; 2020; Yu et al., 2020; Wang et al., 2021b). RegNet (Radosavovic et al., 2020) is the first to present standard guidelines to optimize a search space by each dimension. Recently, (Hu et al.; Noy et al., 2020; Li et al., 2020b; Xia et al., 2022; Ci et al., 2021; Chen et al., 2021) propose to shrink to a better compact search space by either pruning unimportant operators or configurations. However, these works focus on optimizing the accuracy and cannot be applied to search a quantization-friendly space for two reasons. Firstly, none of them search the space for both operator type and configurations, which are the crucial factors impacting INT8 latency; Secondly, the current search is constrained within 10 iterations due to the huge training cost. Our work is the first lightweight solution for hardware-friendly search space automation.

6 Conclusion

In this paper, we introduced SpaceEvo, the first to search a quantization-friendly space for a given device, which delivers superior INT8 quantized models with SOTA efficiency on real-world edge devices. Through the use of accuracy look-up-table, built through block-wise quantization, SpaceEvo efficiently evolves search space for the optimal quantization-friendly operator type and configurations guided by the Q-T score. Extensive experiments on two popular edge devices demonstrate its effectiveness compared to existing prior art manual-designed search spaces. In future work, we plan to apply SpaceEvo for other hardware efficiency optimization (e.g., energy-efficient search space).

REFERENCES

Haoping Bai, Meng Cao, Ping Huang, and Jiulong Shan. Batchquant: Quantized-for-all architecture search with robust quantizer. In *NeurIPS*, 2021.

Ron Banner, Yury Nahshan, and Daniel Soudry. Post training 4-bit quantization of convolutional networks for rapid-deployment. In *Advances in Neural Information Processing Systems*, 2019.

Yash Bhalgat, Jinwon Lee, Markus Nagel, Tijmen Blankevoort, and Nojun Kwak. Lsq+: Improving low-bit quantization through learnable offsets and better initialization. In *CVPRW*, 2020.

Softtek blog. Edge ai: The futuer of artificial intelligence, 2020. URL https://softtek.eu/en/tech-magazine-en/artificial-intelligence-en/edge-ai-el-futuro-de-la-inteligencia-artificial/.

Han Cai, Ligeng Zhu, and Song Han. ProxylessNAS: Direct neural architecture search on target task and hardware. In *International Conference on Learning Representations*, 2019.

Han Cai, Chuang Gan, Tianzhe Wang, Zhekai Zhang, and Song Han. Once-for-all: Train one network and specialize it for efficient deployment. In *ICLR*, 2020.

Zhaowei Cai and Nuno Vasconcelos. Rethinking differentiable search for mixed-precision neural networks. In *2020 IEEE/CVF Conference on Computer Vision and Pattern Recognition*, pp. 2346–2355. Computer Vision Foundation / IEEE, 2020.

Minghao Chen, Kan Wu, Bolin Ni, Houwen Peng, Bei Liu, Jianlong Fu, Hongyang Chao, and Haibin Ling. Searching the search space of vision transformers. In *NeurIPS*, 2021.

Yuanzheng Ci, Chen Lin, Ming Sun, Boyu Chen, Hongwen Zhang, and Wanli Ouyang. Evolving search space for neural architecture search. In *2021 IEEE/CVF International Conference on Computer Vision (ICCV)*, pp. 6639–6649, 2021.

Ekin D. Cubuk, Barret Zoph, Dandelion Mane, Vijay Vasudevan, and Quoc V. Le. Autoaugment: Learning augmentation strategies from data. In *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition (CVPR)*, June 2019.

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 *2021 IEEE/CVF Conference on Computer Vision and Pattern Recognition (CVPR)*, 2021.

Steven K. Esser, Jeffrey L. McKinstry, Deepika Bablani, Rathinakumar Appuswamy, and Dharmendra S. Modha. Learned step size quantization. In *ICLR*, 2020.

Google. Tensorflow lite, 2022. URL https://www.tensorflow.org/lite/guide.

Zichao Guo, Xiangyu Zhang, Haoyuan Mu, Wen Heng, Zechun Liu, Yichen Wei, and Jian Sun. Single path one-shot neural architecture search with uniform sampling. In *ECCV*, 2020.

Kaiming He, Xiangyu Zhang, Shaoqing Ren, and Jian Sun. Deep residual learning for image recognition. In *2016 IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, 2016.

Andrew Howard, Mark Sandler, Grace Chu, Liang-Chieh Chen, Bo Chen, Mingxing Tan, Weijun Wang, Yukun Zhu, Ruoming Pang, Vijay Vasudevan, Quoc V. Le, and Hartwig Adam. Searching for mobilenetv3. In *ICCV*, 2019.

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. *CoRR*, abs/1704.04861, 2017.

Jie Hu, Li Shen, and Gang Sun. Squeeze-and-excitation networks. In *2018 IEEE/CVF Conference on Computer Vision and Pattern Recognition*, 2018.

Yiming Hu, Yuding Liang, Zichao Guo, Ruosi Wan, Xiangyu Zhang, Yichen Wei, Qingyi Gu, and Jian Sun. Angle-based search space shrinking for neural architecture search. In *ECCV*.

Benoit Jacob, Skirmantas Kligys, Bo Chen, Menglong Zhu, Matthew Tang, Andrew Howard, Hartwig Adam, and Dmitry Kalenichenko. Quantization and training of neural networks for efficient integer-arithmetic-only inference. In *2018 IEEE/CVF Conference on Computer Vision and Pattern Recognition*, 2018.

Diederik P. Kingma and Jimmy Ba. Adam: A method for stochastic optimization. In Yoshua Bengio and Yann LeCun (eds.), *3rd International Conference on Learning Representations, ICLR 2015, San Diego, CA, USA, May 7-9, 2015, Conference Track Proceedings*, 2015.

Changlin Li, Jiefeng Peng, Liuchun Yuan, Guangrun Wang, Xiaodan Liang, Liang Lin, and Xiaojun Chang. Block-wisely supervised neural architecture search with knowledge distillation. In *2020 IEEE/CVF Conference on Computer Vision and Pattern Recognition, CVPR*, 2020a.

Xiang Li, Chen Lin, Chuming Li, Ming Sun, Wei Wu, Junjie Yan, and Wanli Ouyang. Improving one-shot nas by suppressing the posterior fading. In *2020 IEEE/CVF Conference on Computer Vision and Pattern Recognition (CVPR)*, 2020b.

Ilya Loshchilov and Frank Hutter. SGDR: Stochastic gradient descent with warm restarts. In *International Conference on Learning Representations*, 2017.

Christos Louizos, Mathias Reisser, Tijmen Blankevoort, Efstratios Gavves, and Max Welling. Relaxed quantization for discretized neural networks. In *ICLR*, 2019.

Microsoft. onnxruntime, 2022. URL https://onnxruntime.ai/.

Bert Moons, Parham Noorzad, Andrii Skliar, Giovanni Mariani, Dushyant Mehta, Chris Lott, and Tijmen Blankevoort. Distilling optimal neural networks: Rapid search in diverse spaces. In *ICCV*, 2021.

Markus Nagel, Mart van Baalen, Tijmen Blankevoort, and Max Welling. Data-free quantization through weight equalization and bias correction. In *ICCV*, 2019.

Asaf Noy, Niv Nayman, Tal Ridnik, Nadav Zamir, Sivan Doveh, Itamar Friedman, Raja Giryes, and Lihi Zelnik. Asap: Architecture search, anneal and prune. In *Proceedings of the Twenty Third International Conference on Artificial Intelligence and Statistics*, pp. 493–503, 2020.

Ilija Radosavovic, Justin Johnson, Saining Xie, Wan-Yen Lo, and Piotr Dollár. On network design spaces for visual recognition. In *ICCV*, 2019.

Ilija Radosavovic, Raj Prateek Kosaraju, Ross Girshick, Kaiming He, and Piotr Dollár. Designing network design spaces. In *CVPR*, 2020.

Esteban Real, Alok Aggarwal, Yanping Huang, and Quoc V Le. Regularized evolution for image classifier architecture search. In *AAAI*, 2019.

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 (CVPR)*, 2018.

Mingzhu Shen, Feng Liang, Ruihao Gong, Yuhang Li, Chuming Li, Chen Lin, Fengwei Yu, Junjie Yan, and Wanlin Ouyang. Once quantization-aware training: High performance extremely low-bit architecture search. In *ICCV*, 2021.

Mingxing Tan and Quoc Le. Efficientnetv2: Smaller models and faster training. In *Proceedings of the 38th International Conference on Machine Learning*, pp. 10096–10106. PMLR, 2021.

Alvin Wan, Xiaoliang Dai, Peizhao Zhang, Zijian He, Yuandong Tian, Saining Xie, Bichen Wu, Matthew Yu, Tao Xu, Kan Chen, Peter Vajda, and Joseph E. Gonzalez. Fbnetv2: Differentiable neural architecture search for spatial and channel dimensions. In *2020 IEEE/CVF Conference on Computer Vision and Pattern Recognition (CVPR)*, 2020.

Dilin Wang, Chengyue Gong, Meng Li, Qiang Liu, and Vikas Chandra. Alphanet: Improved training of supernet with alpha-divergence. In *ICML*, 2021a.

Dilin Wang, Meng Li, Chengyue Gong, and Vikas Chandra. Attentivenas: Improving neural architecture search via attentive sampling. In *Conference on Computer Vision and Pattern Recognition*, 2021b.

Kuan Wang, Zhijian Liu, Yujun Lin, Ji Lin, and Song Han. Haq: Hardware-aware automated quantization with mixed precision. In *CVPR*, 2019.

Tianzhe Wang, Kuan Wang, Han Cai, Ji Lin, Zhijian Liu, and Song Han. Apq: Joint search for network architecture, pruning and quantization policy. In *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition*, 2020.

Bichen Wu, Yanghan Wang, Peizhao Zhang, Yuandong Tian, Peter Vajda, and Kurt Keutzer. Mixed precision quantization of convnets via differentiable neural architecture search. 2019.

Xin Xia, Xuefeng Xiao, Xing Wang, and Min Zheng. Progressive automatic design of search space for one-shot neural architecture search. In *WACV*, 2022.

Jin Xu, Xu Tan, Renqian Luo, Kaitao Song, Jian Li, Tao Qin, and Tie-Yan Liu. Nas-bert: Task-agnostic and adaptive-size bert compression with neural architecture search. In *SIGKDD*, 2021.

Jiahui Yu, Pengchong Jin, Hanxiao Liu, Gabriel Bender, Pieter-Jan Kindermans, Mingxing Tan, Thomas Huang, Xiaodan Song, Ruoming Pang, and Quoc Le. Bignas: Scaling up neural architecture search with big single-stage models. In *ECCV*, 2020.

Li Lyna Zhang, Yuqing Yang, Yuhang Jiang, Wenwu Zhu, and Yunxin Liu. Fast hardware-aware neural architecture search. In *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition (CVPR) Workshops*, June 2020.

Li Lyna Zhang, Shihao Han, Jianyu Wei, Ningxin Zheng, Ting Cao, Yuqing Yang, and Yunxin Liu. nn-meter: Towards accurate latency prediction of deep-learning model inference on diverse edge devices. In *Proceedings of the 19th Annual International Conference on Mobile Systems, Applications, and Services*, pp. 81–93, New York, NY, USA, 2021. ACM.

A DEVICE AND LATENCY PREDICTION

Name	Framework	Processor	Measured Hz*	Precision
Intel VNNI	Onnxruntime v1.10	Intel (R) Xeon(R) Silver 4210 CPU	2GHz	FP32/INT8
Pixel 4	TFLite v2.7	CortexA76 CPU	2.42GHz	FP32/INT8

Table 6: Our measured edge devices. *: For fair comparison, we measure the model latency on a single CPU core with a fixed frequency.

Device	RMSE	±5% accuracy	±10% accuracy
Intel VNNI	1.5ms	79.8%	92.5%
Pixel4	1.3ms	89.6%	100%

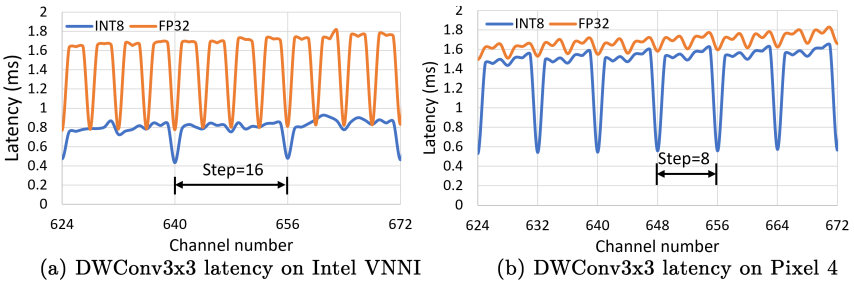
Table 7: INT8 quantized latency prediction performance of nn-Meter. We report the ±5% and ±10% accuracy, that are the percentage of models with predicted latency within the corresponding error bound relative to the measured latency

Device details and latency measurement. Table 6 lists out the detailed inference engine and hardware for our two measured devices. Specifically, we select TFLite and onnxruntime as the inference engines for Pixel 4 mobile CPU and Intel CPU, respectively, because they are well-known high-performance engines for edge AI inference. For latency measurement, we always measure the inference latency of a given model (either FP32 or INT8 quantized model) on a single CPU core with fixed frequency. The inference batch size is 1. The reported latency in the paper is the arithmetic mean of 50 runs after 10 warmup runs. The 95% confidence interval is within 2%.

Latency prediction of quantized models. We now illustrate the details of building quantized latency predictors. Following the original nn-Meter paper, the latency of a given model is the sum of all kernels' predicted latency. Then the procedure contains two main steps: (i) detect kernels in a model, and (ii) build latency predictors for these kernels. For step (i), we perform the fusion rule detection in nn-Meter and detects 17 kernels (e.g., Conv-bn-relu and DWConv-bn-relu6). For step (ii), we run the adaptive data sampler to collect training data for each kernel and train a randomforest regressor as the kernel-level latency predictor. Each training data sample is a pair of (configurations of a kernel, the inference latency). Taking Conv-bn-relu kernel as an example, we sample different kernel sizes, strides, input/output channel numbers and input resolution for Conv-bn-relu kernel, then we measure their INT8 quantized latency on the target device.

The kernel-level latency predictors training and construction are conduct offline. During the evolutionary search, we use them to predict the INT8 quantized latency for arbitrary model. Table 7 lists out the prediction performance. Specifically, we randomly sample 2k models and predict their INT8 quantized latency for evaluation. Remarkably, nn-Meter achieves 92.5% and 100% prediction accuracy on the Intel VNNI and Pixel 4, respectively.

B ON-DEVICE QUANTIZATION EFFICIENCY ANALYSIS



(a) DWConv3x3 latency on Intel VNNI (b) DWConv3x3 latency on Pixel 4

Figure 9: The choice of channel number greatly impact the INT8 latency of DWConv. INT8 latency of DWConv shows a step pattern. Configuration: HW=56.

In Section. 2, we studied how the operator type and channel widths in a quantized model impact final inference efficiency. Next, we will illustrate the impact of other configuration dimensions.

Kernel size. Table 8 shows the latency speedup of Conv and DWConv with different kernel sizes on two devices. Results suggest that the choice of kernel size can result in different speedups after

Kernel size	Intel VNNI		Pixel4	
	Conv	DWConv	Conv	DWConv
1	3.5×	2.7×	2.6×	0.9×
3	3.6×	1.4×	4.0×	2.8×
5	3.8×	1.1×	3.9×	1.1×
7	4.0×	0.8×	3.8×	1.1×

Table 8: INT8 quantization speedup on two devices. Kernel size impacts the speedup for DWConv. Configuration: H=W=56, C_{in}=C_{out}=96.

INT8 quantization. Specifically, we notice that kernel size is more crucial to DWConv quantization efficiency than Conv. Unlike Conv can consistently achieve speedups under various kernel sizes, improper kernel size of DWConv can lead to a significant slowdown. Moreover, the most efficient kernel size for DWConv is highly relying on the target devices. DWConv with smaller kernel sizes can achieve larger speedup on Intel VNNI, while DWConv with $K = 3$ achieves the maximum speedup of 2.8× on Pixel 4.

Consequently, the INT8 quantized models should consider the choices of kernel sizes to achieve both high accuracy and low inference latency. Instead of searching the optimal kernel sizes for a search space, we directly allow all stages to choose from {3, 5, 7}. In our work, we perform latency-aware evolutionary search to derive the optimal quantized models from the resulting search space. As shown in Table 18 and Table 19, the optimal quantization-friendly kernel sizes are chosen for our discovered model family SeqNet. For example, kernel size of 3×3 is the dominate choice in DWConv in SeqNet.

Channel number of DWConv. Fig. 9 show the latency of DWConv 3×3 with different channel number on two devices. Surprisingly, we observe a step pattern: the latency of DWConv 3×3 achieves minimal at special channel numbers. Specifically, in terms of INT8 quantized latency, when C is divisible to 8 on Pixel 4 (16 on Intel VNNI), the latency achieves a minimal and can be accelerated by 2.9× (1.7× on Intel VNNI). Therefore, we constrain the channel widths in our search space to be divisible by 8 on Pixel 4 and 16 on Intel VNNI. Moreover, we notice that the latency patterns of FP32 and INT8 inference are different on two devices, which further motivates SpaceEvo.

C Search Space Details

C.1 Hyper space

Stage	Depths	Kernel Size	Stride	Channel Widths 16-divisible	ck
Conv (STEM)	1	3	2	16 - 32	-
Residual (STEM)	1-2	3	1	16 - 32	-
Stage1	2-4	3, 5, 7	2	32 - 64	2
Stage2	2-4	3, 5, 7	2	32 - 96	2
Stage3	2-6	3, 5, 7	2	64 - 144	3
Stage4	2-6	3, 5, 7	1	112 - 192	3
Stage5	2-6	3, 5, 7	2	192 - 304	5
Stage6	1-2	3, 5, 7	1	304 - 448	7
Classifier (head)	-	-	-	-	-
input resolution	160, 176, 192, 208, 224				

Table 9: VNNI hyperspace. We search the optimal block type and channel widths for Stage1-6.

Stage	Depths	Kernel Size	Stride	Channel Widths 8-divisible	ck
Conv (STEM)	1	3	2	16-32	-
MBv2 (STEM)	1-2	3	1	16-32	-
Stage1	2-4	3, 5, 7	2	24 - 32	2
Stage2	2-6	3, 5, 7	2	40 - 56	2
Stage3	2-6	3, 5, 7	2	80 - 104	3
Stage4	2-8	3, 5, 7	1	96 - 128	3
Stage5	2-8	3, 5, 7	2	192 - 256	5
Stage6	1-2	3, 5, 7	1	320 - 416	7
Classifier (head)	-	-	-	-	-
input resolution	160, 176, 192, 208, 224				

Table 10: Pixel4 hyperspace. We search the optimal block type and channel widths for Stage1-6.

Block type	Search id	Intel VNNI		Pixel4	
		Expand Ratios	Activation	Expand Ratios	Activation
MBv1	0	-	relu	-	relu
MBv2	1	4, 6, 8	relu6	3, 6, 8	relu6
MBv3	2	4, 6, 8	hswish	3, 6, 8	swish
Residual bottleneck	3	0.5, 1.0, 1.5	relu	0.5, 1.0, 1.5	relu
Residual bottleneck+SE	4	0.5, 1.0, 1.5	relu	0.5, 1.0, 1.5	relu
FusedMB	5	1, 2, 3, 4	swish	1, 2, 3, 4	swish
FusedMB+SE	6	1, 2, 3, 4	swish	1, 2, 3, 4	swish

Table 11: Block choices for an elastic stage. We set larger expand ratios and use swish as the activation function

Table 9 and Table 10 summarize the hyperspace structures when targeting Intel VNNI CPU and Pixel 4 mobile CPU, respectively. As introduced in Section 3.2, we search the 6 stages (Stage1-Stage6) for a quantization-friendly search space. For each elastic stage, it can search: (i) an optimal block type from a pool as shown in Table 11; and (ii) the optimal channel widths for that block type as shown in Table 9 and Table 10. Other search space dimensions including kernel sizes, expand ratios and input resolutions are unsearchable and using manual settings.

In total, the hyperspace contains $\sim 10^9$ possible candidate search spaces, which is extremely large and we leverage the aging evolution for efficient search. For better mutations, we encode a candidate search space as the format of *PerStageBlock-PerStageWidth*. As shown in Table 11, we assign an unique search id for each candidate block types. For example, 1 indicates selecting the MBv2 block. For the channel widths, we use the minimal channel index as the encoding. For example, 0 indicates selecting channel widths of {32, 48} (32 is the 0^{th} item in {32, 48, 64}) for Stage1 on the Intel VNNI; 1 indicates selecting {48, 64}. Taking 111111-000000 as an example, it means that all stages choose the MBv2 block; and the chosen channel widths for Stage1 to Stage6 are {32, 48}, {32, 48}, {64, 80, 96}, {112, 128, 144}, {192, 208, 224, 240, 256}, {304, 320, 336, 352, 368, 384, 400}, respectively.

C.2 Searching the Quantization-friendly Search Space

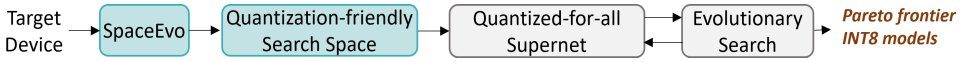


Figure 10: Our overall pipeline has three steps: search a quantization-friendly search space; train a quantized-for-all supernet; search an optimal quantized model under diverse latency constraints.

Algorithm 1 Search Space Evolution Algorithm

Input: Hyperspace $\mathcal{H}$, total number of spaces to explore N, population size P, sample size S, latency_auc tradeoff α, target latency constraints $T_{0,1,..n}$

1: $population^{(0)} \leftarrow$ `initialize` ($\mathcal{H}$, P)
2: **for** i=1: (N-P) **do**
3: $parent \leftarrow$ `sample_with_maximum_score` ($population^{(i-1)}$,S)
4: $child_1 \leftarrow$ `mutate-block-type` ($parent$, $T_{0,1,..n}$)
5: $score_1 \leftarrow$ `get_quality_score` ($child_1$, $T_{0,1,..n}$,α) (see Equation 2)
6: $child_2 \leftarrow$ `mutate-block-width` ($parent$, $T_{0,1,..n}$)
7: $score_2 \leftarrow$ `get_quality_score` ($child_2$, $T_{0,1,..n}$, α) (see Equation 2)
8: $population^{(i)} \leftarrow$ add $child_1$ and $child_2$ to right of $population^{(i-1)}$, remove the oldest two spaces from left of $population^{(i-1)}$
9: **end for**
10: return the search space with maximum score under the $T_{0,1,...n}$

Search algorithm. Algorithm 1 illustrates the major procedures of searching a quantization-friendly search space. To generate the initial population, we randomly sample P=500 search spaces from the hyperspace, then we perform random mutations to evolve better search spaces. Specifically, for each evolution iteration, we first sample S=125 spaces from the current population, and select the one with highest Q-T score as the parent. Then we alternately mutate the parent and produce two children (line 4-7). The following illustrates an example. Assume the search space 111111-000000 is selected as the parent, and stage 2 is randomly selected for mutation. For the block type mutation, an example child_1 131111-000000 is produced. For the channel width mutation, we can randomly create 111111-020000 as the child_2. We add the two children into the current population and remove the oldest two for next iteration. In total, we search 5k search spaces for each device.

Fig. 11 compares the search efficiency with random search. For each method, we set a same range of latency constraints, and keep track of the searched spaces for every 500 iterations. We select the top 10 spaces with the best Q-T score over time. Fig. 11 demonstrates that our space evolution algorithms consistently achieve better search spaces (i.e., higher Q-T score) than random search.

Searched space. Table 12 and Table 13 list out the searched space structures for Intel VNNI and Pixel4 mobile CPU, respectively. In experiment section, the main results (Table 2, Table 3, Fig. 5 and Fig. 6) are reported through searching the two search spaces. In addition, we provide the detailed search space structures under other latency constraints in Table 14, Table 15 and Table 16.

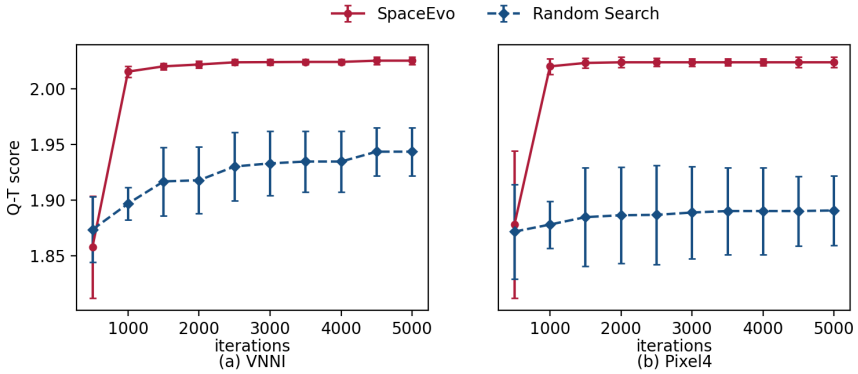


Figure 11: Space search efficiency comparison.

Stage	Depths	Kernel Size	Stride	Channel Widths 16-divisible	Expand Ratio
Conv	1	3	2	16 - 32	-
Residual bottleneck	1-2	3	1	16 - 32	0.5
MBv2	2-4	3, 5, 7	2	32 - 48	4, 6, 8
FusedMB	2-4	3, 5, 7	2	64 - 80	1, 2, 3, 4
MBv2	2-6	3, 5, 7	2	112 - 144	4, 6, 8
MBv2	2-6	3, 5, 7	1	144 - 176	4, 6, 8
MBv3 (hswish)	2-6	3, 5, 7	2	240 -304	4, 6, 8
MBv3 (hswish)	1-2	3, 5, 7	1	320 - 416	4, 6, 8
Classifier	-	-	-	-	-
input resolution	160, 176, 192, 208, 224				

Table 12: SpaceEvo@VNNI: searched space under constraint of {8, 10, 15, 20, 25}ms on VNNI.

Stage	Depths	Kernel Size	Stride	Channel Widths 8-divisible	Expand Ratio
Conv	1	3	2	16-32	-
MBv2	1-2	3	1	16-32	1
MBv2	2-4	3, 5, 7	2	32 - 40	3, 6, 8
MBv3	2-6	3, 5, 7	2	64 - 72	3, 6, 8
MBv3	2-6	3, 5, 7	2	96 - 112	3, 6, 8
MBv3	2-8	3, 5, 7	1	144 - 160	3, 6, 8
MBv3	2-8	3, 5, 7	2	192 - 224	3, 6, 8
MBv3	1-2	3, 5, 7	1	216 - 312	3, 6, 8
Classifier	-	-	-	-	-
input resolution	160, 176, 192, 208, 224				

Table 13: SpaceEvo@Pixel4: searched space under constraint of {15, 20, 25, 30, 35}ms on Pixel4.

Stage	Depths	Kernel Size	Stride	Channel Widths 8-divisible	Expand Ratio
Conv	1	3	2	16 - 32	-
MBv2	1-2	3	1	16 - 32	1
MBv2	2-4	3, 5, 7	2	32 - 40	3, 6, 8
MBv2	2-4	3, 5, 7	2	64 - 72	3, 6, 8
MBv3	2-6	3, 5, 7	2	88 - 104	3, 6, 8
MBv3	2-6	3, 5, 7	1	144 - 160	3, 6, 8
MBv3	2-6	3, 5, 7	2	200 - 240	3, 6, 8
MBv1	1-2	3, 5, 7	1	216 - 312	3, 6, 8
Classifier	-	-	-	-	-
input resolution	160, 176, 192, 208, 224				

Table 14: SpaceEvo@Pixel4-medium: searched space under constraint of {10, 15, 20, 25, 30}ms.

Stage	Depths	Kernel Size	Stride	Channel Widths 8-divisible	Expand Ratio
Conv	1	3	2	16-32	-
MBv2	1-2	3	1	16-32	1
Residual bottleneck	2-4	3, 5, 7	2	32 - 40	3, 6, 8
Residual bottleneck	2-6	3, 5, 7	2	48 - 56	3, 6, 8
MBv3	2-6	3, 5, 7	2	88- 104	3, 6, 8
MBv2	2-8	3, 5, 7	1	128 - 144	3, 6, 8
MBv2	2-8	3, 5, 7	2	192 - 224	3, 6, 8
MBv1	1-2	3, 5, 7	1	216 - 312	3, 6, 8
Classifier	-	-	-	-	-
input resolution	160, 176, 192, 208, 224				

Table 15: SpaceEvo@Pixel4-tiny: searched space under constraint of {6, 10, 15, 20, 25}ms.

Stage	Depths	Kernel Size	Stride	Channel Widths 16-divisible	Expand Ratio
Conv	1	3	2	16 - 32	-
Residual bottleneck	1-2	3	1	16 - 32	0.5
MBv2	2-4	3, 5, 7	2	48 - 64	4, 6, 8
MBv3 (hswish)	2-4	3, 5, 7	2	80 - 96	4, 6, 8
MBv2	2-6	3, 5, 7	2	112 - 144	4, 6, 8
MBv2	2-6	3, 5, 7	1	160 - 192	4, 6, 8
MBv3 (hswish)	2-6	3, 5, 7	2	240 - 304	4, 6, 8
MBv3 (hswish)	1-2	3, 5, 7	1	320 - 416	4, 6, 8
Classifier	-	-	-	-	-
input resolution	160, 176, 192, 208, 224				

Table 16: SpaceEvo@VNNI-large: searched space under constraint of {10, 15, 20, 25, 30}ms.

C.3 BLOCK-WISE QUANTIZATION

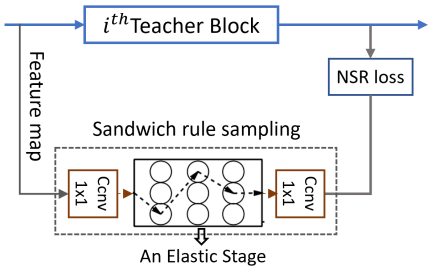


Figure 12: A detailed illustration of block-wise knowledge quantization process. We separately distill all elastic stages from the corresponding teacher block.

Fig. 12 illustrates the detailed process of quantizing a candidate elastic stage through block-wise knowledge distillation (BKD). Firstly, we divide our teacher model EfficientNet-B5 and our search space into 6 stages. Then, we separately train each candidate elastic stage (student) in the search space under the guidance of teacher's feature map in the corresponding block.

At each training step, we apply sandwich rule to sample 4 stage-architectures (min, max, and two random) from an elastic stage. The two random stage-architectures are allowed to select different channel numbers, kernel sizes, expand ratios, depths, and input resolutions. Since the channel numbers of the sampled stage-architecture (student) may be different from that in the teacher block, we cannot directly leverage the input and output of the teacher block as the training data. Inspired by (Xu et al., 2021), we use a learnable linear transformer layer of Conv1x1 at the input and output of the elastic stage to transform the input/output channel numbers to match that of the teacher block, as shown in Fig. 12.

NSR Loss. For a sampled stage-architecture m in the i^{th} stage of a search space, it receives the output of the $(i-1)^{th}$ teacher block as the input and is optimized to predict the output of the i^{th} teacher block with NSR (per-channel noise-to-signal-power ratio) loss:

$$\mathcal{L}(E(m)^i, Y_i) = \frac{1}{C}\sum_{c=0}^{C}\frac{\|Y_{i,c} - f(Y_{i-1}; E(m)^i)_c\|^2}{\sigma_{i,c}^2} \tag{3}$$

Where $E(m)^i$ is a sampled stage-architecture in elastic stage E^i, it take the $(i-1)^{th}$ teacher block's output feature map Y_{i-1} as the input. Y_i is the target output feature map of the i^{th} block of the teacher model, $f(Y_{i-1}; E(m)^i)$ maps the output feature map of the sampled stage-architecture. C is the number of channels in a feature map and $\sigma_{i,c}^2$ is the variance of $Y_{i,c}$.

Training Hyperparameters. All candidate elastic stages are trained in a parallel way. Specifically, each elastic stage is trained for 6 epochs on 4 V100 GPUs. The first 5 epochs are full-precision training without INT8 quantizers. We use 0.005 as the initial learning rate for Stage 1 and Stage 6, and 0.01 for all the other Stages. We apply a cosine learning rate schedule (Loshchilov & Hutter, 2017), a batch size of 256, the Adam (Kingma & Ba, 2015) optimizer. After the full-precision training finishes, we conduct 1 epoch INT8 quantization-aware training with LSQ+. We use a much smaller learning rate of 0.0025 for quantization.

D EXPERIMENT

Supernet Training and QAT. To train our quantized-for-all supernet, we perform two-stage training: (i) supernet pretraining without quantizers; and (ii) QAT on the pretrained supernet.

(i) supernet pretraining without quantizers. We use the sandwich rule and inplace distillation in BigNAS (Yu et al., 2020). Our training hyperparameters setting follows AlphaNet (Wang et al., 2021a). Specifically, we use SGD with a cosine learning rate decay. We train our supernet for 360 epochs on 8 Tesla Nvidia V100 GPUs. The mini-batch size is 128 per GPU. The initial learning rate is set as 0.1 and is linearly scaled up for every 256 training samples. We use momentum of 0.9, weight decay of 1e-5, dropout of 0.2 after the global average pooling layer. We use AutoAugment Cubuk et al. (2019) for data augmentation and set label smoothing coefficient to 0.1.

(ii) QAT on the pretrained supernet. This step starts from a full-precision pretrained supernet. We load checkpoints from step (i) and perform LSQ+ for supernet quantization. The training receipt follows step (i) except we use a smaller learning rate of 0.01 and train the supernet for 50 epochs.

Quantization-aware NAS baselines. In our paper, we focus on INT8 quantization because it has been widely supported on real-world devices. Since existing quantization-aware NAS works focus on searching mixed-precision models, we would like not to directly compare our method with mixed-precision NAS works. We select the two state-of-the-art methods of APQ and OQAT and make some modifications based on their original settings.

APQ originally searches the model architecture and its layer-wise mixed-precision of {4, 8} bits. It manages an accuracy predictor that can predict the accuracy of a mixed-precision quantized model. To compare with APQ, we constrain each layer can only search 8bit. For fair comparison, we add our INT8 quantized latency predictor nn-Meter during the model search process. After the search finishes, we follow APQ to perform finetuning of 30 epochs.

OQAT also trains a quantized-for-all supernet. However, it supports 4bit and 2 bit. To compare with OQAT, we simply modify its source code to quantize 8bit and re-do the supernet QAT process. Same with us, the supernet QAT is trained for 50 epochs.

Search Space	Best searched models						
	smallest	8ms	10ms	15ms	20ms	25ms	28ms
SpaceEvo@VNNI	74.7 (4.4ms)	76.7	77.4	78.7	79.5	80.0	80.1
SpaceEvo@VNNI-large	76.2 (6.7ms)	76.5	77.1	78.9	79.7	80.1	80.3

Table 17: SpaceEvo@VNNI-large: searched space under constraint of {10, 15, 20, 25, 30}ms. We report the best searched quantized model accuracy and latency.

SpaceEvo under larger latency constraints for Intel VNNI. In Section 4.2, we focus on tight INT8 latency constraints. We now provide additional results of search under larger latency constraints of {10, 15, 20, 25, 30}ms. The searched space structure is shown in Table 16. We report the best searched quantized models and compare with SpaceEvo@VNNI in Table 17. Under larger latency constraints, the searched space SpaceEvo@VNNI-large produces better large models ($\geq$15 ms).

E ARCHITECTURE VISUALIZATION OF SEQNET

In the following, we visualize the searched INT8 quantized model architectures in Table 18 and Table 19. 'd' denotes number of layers, 'c' denotes the number of output channels, 'k' denotes kernel size, 'e' denotes expansion ratio. If a stage has multiple layers, we list out the kernel size, output channel numbers, expansion ratios for each layer and use '-'to separate them. For example, c: 32-48 indicates that the channel numbers are 32 for the first layer and 48 for the second layer.

	SeqNet@vnni-A0	SeqNet@vnni-A1	SeqNet@vnni-A2	SeqNet@vnni-A3	SeqNet@vnni-A4
Conv	d: 1 c: 16 k: 3	d: 1 c: 16 k: 3	d: 1 c: 16 k: 3	d: 1 c: 16 k: 3	d: 1 c: 32 k: 3
Residual bottleneck	d: 1 c: 16 k: 3 e: 0.5	d: 1 c: 16 k: 3 e: 0.5	d: 1 c: 16 k: 3 e: 0.5	d: 1 c: 32 k: 3 e: 0.5	d: 1 c: 32 k: 3 e: 0.5
MBv2	d: 2 c: 32-32 k: 3-3 e: 4-4	d: 2 c: 32-32 k: 3-3 e: 4-4	d: 2 c: 32-32 k: 3-3 e: 4-6	d: 2 c: 32-48 k: 3-3 e: 4-8	d: 3 k: 3-3-3 c: 48-48-32 e: 4-6-6
FusedMB	d: 2 c: 64-64 k: 3-3 e: 1-1	d: 2 c: 64-80 k: 3-3 e: 2-1	d: 2 c: 64-64 k: 3-3 e: 3-2	d: 2 c: 80-80 k: 3-3 e: 4-2	d: 3 c: 64-64-64 k: 5-3-3 e: 4-3-1
MBv2	d: 2 c: 112-112 k: 3-3 e: 4-4	d: 3 c: 128-112-112 k: 3-5-3 e: 4-6-4	d: 4 c: 112-112-128 k: 3-3-5-5 e: 4-4-6-8	d: 3 c: 144-128-112 k: 5-5-5 e: 6-6-6	d: 4 c: 112-112-112-112 k: 5-5-3-5 e: 8-8-4-4
MBv2	d: 2 c: 144-144 k: 3-3 e: 4-4	d: 2 c: 144-176 k: 3-3 e: 4-4	d: 4 c: 144-144-160-160 k: 3-5-5-3 e: 8-4-4-4	d: 4 c: 144-176-144-144 k: 3-3-5-3 e: 6-8-4-4	d: 5 c: 176-176-144-144-160 k: 5-5-3-3-3 e: 8-4-4-6-4
MBv3 (hswish)	d: 2 c: 240-240 k: 3-3 e: 4-4	d: 3 c: 304-304-256 k: 3-3-3 e: 4-6-4	d: 4 c: 288-272-288-256 k: 3-3-3-3 e: 4-6-6-4	d: 4 c: 272-288-272-240 k: 3-5-3-3 e: 8-6-4-6	d: 4 c: 304-272-272-288 k: 3-3-5-3 e: 8-8-6-4
MBv3 (hswish)	d: 1 c: 320 k: 3 e: 4	d: 1 c: 368 k: 5 e: 4	d: 1 c: 320 k: 5 e: 4	d: 1 c: 320 k: 3 e: 6	d: 2 c: 352-320 k: 3-3 e: 6-4
Resolution	160	192	208	224	224

Table 18: INT8 quantized models produced by SpaceEvo@VNNI.

	SeqNet@pixel4-A0	SeqNet@pixel4-A1	SeqNet@pixel4-A2	SeqNet@pixel4-A3	SeqNet@pixel4-A4
Conv	d: 1 c: 16 k: 3	d: 1 c: 16 k: 3	d: 1 c: 16 k: 3	d: 1 c: 16 k: 3	d: 1 c: 24 k: 3
MBv2	d: 1 c: 16 k: 3 e: 1	d: 1 c: 16 k: 3 e: 1	d: 1 c: 16 k: 3 e: 1	d: 1 c: 16 k: 3 e: 1	d: 2 c: 16-24 k: 3-3 e: 1-1
MBv2	d: 2 c: 32-32 k: 3-3 e: 3-3	d: 2 c: 32-32 k: 3-3 e: 3-3	d: 2 c: 40-32 k: 3-3 e: 3-3	d: 2 c: 32-32 k: 3-3 e: 8-3	d: 2 k: 3-3 c: 32-32 e: 6-8
MBv3	d: 2 c: 64-64 k: 3-3 e: 3-3	d: 2 c: 64-64 k: 3-3 e: 3-3	d: 2 c: 64-72 k: 3-3 e: 3-3	d: 3 c: 64-72 k: 3-3-5 e: 3-6-3	d: 3 c: 72-64-64 k: 3-3-5 e: 6-3-3
MBv3	d: 2 c: 96-96 k: 3-3 e: 3-3	d: 2 c: 96-96 k: 7-3 e: 3-3	d: 2 c: 96-96 k: 3-3 e: 3-6	d: 3 c: 64-112-96 k: 7-3-3 e: 8-3-6	d: 4 c: 112-96-104-104 k: 3-5-5-3 e: 6-6-3-3
MBv3	d: 2 c: 144-144 k: 3-3 e: 3-3	d: 2 c: 144-144 k: 3-3 e: 3-3	d: 3 c: 152-160-152 k: 5-3-3 e: 3-6-3	d: 4 c: 104-152-144-152 k: 3-3-3-5 e: 6-6-3-3	d: 4 c: 152-144-144-152 k: 3-3-3-5 e: 8-8-6-3
MBv3	d: 2 c: 192-192 k: 3-3 e: 3-3	d: 4 c: 224-192-208-192 k: 3-3-5-3 e: 6-6-6-3	d: 4 c: 200-224-200-200 k: 7-5-7-5 e: 6-3-3-6	d: 4 c: 200-192-208-192 k: 3-3-3-5 e: 6-8-6-6	d: 5 c: 224-216-208-224-208 k: 7-5-3-5-3 e: 8-6-6-3-3
MBv3	d: 1 c: 216 k: 3 e: 3	d: 1 c: 232 k: 3 e: 3	d: 1 c: 248 k: 5 e: 3	d: 1 c: 280 k: 3 e: 6	d: 1 c: 296 k: 3 e: 6
Resolution	160	208	224	224	224

Table 19: INT8 quantized models produced by SpaceEvo@Pixel4