MonoNAS: Efficiently Scaling Up Cross-Device Neural Architecture Search for Mobile Inference

Bingqian Lu *UC Riverside*
Jianyi Yang *UC Riverside*
Weiwen Jiang *Notre Dame*
Yiyu Shi *Notre Dame*
Shaolei Ren *UC Riverside*

Abstract

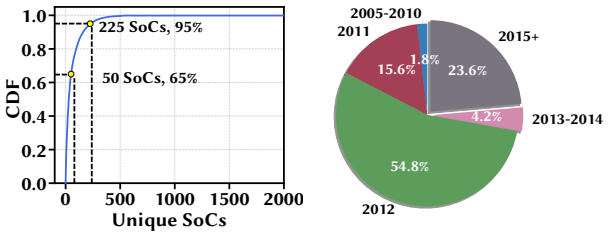
To run convolutional neural network (CNN) inference directly on mobile devices (a.k.a. mobile inference) with an optimal performance, device-aware neural architecture search (NAS) is crucial. Despite recent advances, however, NAS is still a time-consuming process for even a single device, let alone the extremely diverse mobile market. Thus, efficiently scaling up NAS across many devices without losing optimality arises as a key challenge. In this work, we address the scalability challenge by exploiting latency monotonicity *— the ranking orders of different models' latencies on many devices are highly correlated. Our proposed approach, MonoNAS, provably optimizes neural architectures for many mobile devices, yet at a much lower total search cost than that of state-of-the-art per-device NAS. We conduct experiments with different mobile devices on ImageNet. Our results highlight that, even when latency monotonicity is approximately satisfied for target devices in practice, MonoNAS can find almost the same Pareto-optimal architectures as those obtained by using device-specific NAS.*

1. Introduction

The growing integration of convolutional neural networks (CNNs) into mobile apps for on-device inference (a.k.a. mobile inference) has been quickly emerging in recent years [35]. Compared to cloud-based inference, mobile inference offers several major advantages, including being free from the network connection requirement, saving bandwidths and better protecting user privacy as a result of local data processing.

The neural architecture of a CNN can largely affect the resulting model performance such as accuracy and latency. As such, optimizing the neural architecture through *device-aware* neural architecture search (NAS) is crucial. A common goal of NAS for mobile inference is to maximize the accuracy subject to an average inference latency constraint [5, 6, 12, 29, 30, 34].

In the presence of extremely diverse mobile devices (Fig. 1), however, device-aware NAS for mobile inference is very challenging. For example, Facebook's neural network for style transfer runs on billions of mobile devices, more than half of which still use CPUs designed in 2012 or before [35]. Importantly, there are over one hundred major system-on-chips (SoCs) available, without a single one dominating the market: only top 30 SoCs can each have over 1% of the share, top 50 SoCs only account for 65% of the market [35].



(a) SoC Market Share (b) Mobile CPU Year

Figure 1: Device statistics for Facebook users as of 2018 [35]. (a) Fragmented mobile SoC market. (b) Most mobile CPUs were designed more than 6 years ago.

Worse yet, because of the exponentially large search space, NAS is time-consuming for even a single mobile device [13, 29, 30, 32, 34, 36]. Multiplied by hundreds or even more, the total cost of NAS for all the mobile devices can quickly become prohibitive. While effective search techniques (e.g., reinforcement learning) can greatly reduce the number of searched architectures and one-shot NAS avoids training every candidate CNN by using a super net, evaluating hundreds of thousands of candidate architectures in terms of their accuracy and latency (on the target mobile device) still slows down NAS [29, 32, 34]. More recently, performance predictors have been utilized to speed up NAS without actually deploying or running each candidate CNN on the target device [5, 6, 11–13, 32, 36].

Nonetheless, building a latency predictor for a target device requires significant engineering efforts and can be very slow. For example, [11] measures average inference laten-

cies for 5k sample DNNs on a mobile device and uses the results to build a latency lookup table for that specific device. Assuming 20 seconds for each measurement and non-stop measurement, it can take 27+ hours to build the latency predictor for one single device [11]. Therefore, the existing latency-aware NAS [5, 13, 32] suffers from a total cost of $\mathcal{O}(n)$ for n different devices. Crucially, the latency predictor is device-specific [5, 12, 13, 32], and naively applying an architecture searched for one mobile device to another can result in dramatically different latencies, violating the desired latency constraint and causing a poor user experience [35].

Consequently, how to efficiently scale up NAS in the presence of diverse mobile devices has arisen as a significant challenge. A straightforward approach is to heuristically adapt the architecture optimized on one device to another (by, e.g., changing the number of layers and/or channels), but it may be far from optimum and has no performance guarantees [29, 30, 36]. On the other hand, given the cost of NAS, optimizing the architecture for each individual device does not scale well [6].

Contributions. In this paper, we address the scalability challenge of efficient cross-device NAS for mobile inference.[1] Concretely, we exploit *latency monotonicity* and propose MonoNAS (Monotonicity for NAS), which provably optimizes neural architectures for multiple devices at a much lower search cost comparable to that of state-of-the-art NAS on only one single device.

Informally, latency monotonicity means that the ranking orders of different models' latencies are highly correlated on two or more devices. We empirically demonstrate cross-device latency monotonicity *within* each platform (including mobile, desktop GPUs/CPUs, and FPGAs), as further supported by roofline analysis [17]. Then, for all devices satisfying latency monotonicity, we prove that they share the same set of Pareto-optimal neural architectures (in terms of accuracy-latency tradeoff).

We validate theory-supported MonoNAS by running experiments with four mobile devices on ImageNet. Our results show that, provided that latency monotonicity is approximately satisfied (measured in terms of a ranking correlation coefficient in Section 4.1), by using MonoNAS to transfer architectures optimized on one device to another, there is almost no performance loss in terms of top-1 accuracy under the same latency constraint (compared to CNN models specifically optimized on the new target device). Importantly, given 300 devices, the total cost of MonoNAS is estimated to be around $\mathcal{O}(10)$ instead of $\mathcal{O}(300)$.

[1] In this paper, "cross-device" means different devices of the same type (e.g., Samsung S5e vs. Google Pixel1), whereas "cross-platform" means devices of differen types (e.g., mobile vs. FPGA).

2. Related Work

The huge search space for architectures presents significant challenges. Thus, automated approaches, such as reinforcement learning, have been proposed to speed up NAS (see [15, 21, 22, 29, 30, 34, 37] and references therein). Further, to minimize the cost of training numerous architectures, one-shot NAS uses a super net that includes all the weights for candidate architectures [4–6, 10, 16].

Besides accuracy, NAS for mobile inference also needs to account for inference latency. Measuring the actual latencies of numerous candidate architectures on a target device can be extremely time-consuming. For example, MnasNet measures actual latencies and takes several days to search for optimal architectures on a single device [29]. More recently, without actually evaluating each candidate architecture, performance predictors or lookup tables have been commonly utilized to assist with NAS [7, 22–25, 27, 28, 31, 32]. Nonetheless, building a latency predictor itself can be time-consuming (e.g., 5k sample DNNs in [11] and 350k records in [13]). In fact, the time for building a latency predictor is already comparable to or even longer than some state-of-the-art NAS processes [5, 6]. Importantly, unlike an accuracy predictor, the latency predictor is device-specific (e.g., as shown in Fig. 4 in [13], the same operator has very different latencies on two devices).

Consequently, the existing latency-constrained NAS cannot scale well to many heterogeneous mobile devices. Likewise, generative techniques [20] require numerous training samples, while still requiring a large number of device-specific latency evaluations.

Heuristic scaling approaches to adapting architectures optimized on one device to a new device have also been used, e.g., by changing the number of layers and/or channels [29, 30]. But, they have coarse scaling granularity and no performance guarantees. Considering a latency metric aggregated over a few devices, simultaneous multi-device NAS [12] may not meet the latency constraint or achieve Pareto optimality for any involved device. By contrast, MonoNAS exploits cross-device latency monotonicity to address the poor scalability of existing NAS for mobile inference at a very low total cost, yet (almost) without losing optimality.

3. Problem Formulation

The general problem of device-aware NAS for mobile inference can be formulated as follows:

$$\max_{\mathbf{x}\in\mathcal{X}} \max_{\omega_{\mathbf{x}}} \; accuracy(\mathbf{x}, \omega_{\mathbf{x}}) \tag{1}$$

$$s.t., \;\; latency(\mathbf{x}; \mathbf{d}) \leq \overline{L}_{\mathbf{d}} \tag{2}$$

where $\mathbf{x}$ represents the architecture, $\mathcal{X}$ is the search space under consideration, $\omega_{\mathbf{x}}$ is the network weight given architecture $\mathbf{x}$, $\overline{L}_{\mathbf{d}}$ is the average inference latency constraint,

and $\mathbf{d} \in \mathcal{D}$ denotes a mobile device with $\mathcal{D}$ being the device set. Note that $accuracy(\mathbf{x}, \omega_{\mathbf{x}})$ is measured on a dataset independent of the device $\mathbf{d}$, and can also be replaced with a certain loss function (e.g., cross entropy). As in [5, 12, 29, 34, 36], we mainly focus on the inference latency constraint, as it is an important metric directly affecting user experience and also correlated with other metrics such as energy consumption [35].

For a device $\mathbf{d} \in \mathcal{D}$, we denote the optimal architecture as $\mathbf{x}^*(\overline{L}_{\mathbf{d}}; \mathbf{d})$, highlighting its dependency on the target device $\mathbf{d}$. By varying $\overline{L}_{\mathbf{d}}$ between its feasible range $[\overline{L}_{\mathbf{d},\min}, \overline{L}_{\mathbf{d},\max}]$, we can obtain a set of *Pareto*-optimal architectures, denoted by $\mathcal{P}_{\mathbf{d}} = \{\mathbf{x}^*(\overline{L}_{\mathbf{d}}; \mathbf{d}), \text{for } \overline{L}_{\mathbf{d}} \in [\overline{L}_{\mathbf{d},\min}, \overline{L}_{\mathbf{d},\max}]\}$.[2]

Clearly, measuring device-specific $latency(\mathbf{x}; \mathbf{d})$ for numerous candidate architectures is time-consuming [13, 29]. Meanwhile, building such a latency lookup table with a high accuracy is also slow and requires significant engineering efforts for each individual device [11, 13, 32].

4. Latency Monotonicity

We demonstrate a critical property — *latency monotonicity* — that exists among many mobile devices and enables MonoNAS to efficiently scale up cross-device NAS.

4.1. Definition and Metric

We now formally define latency monotonicity.

Definition (Latency Monotonicity): Given two different devices $\mathbf{d}_1 \in \mathcal{D}$ and $\mathbf{d}_1 \in \mathcal{D}$, if $latency(\mathbf{x}_1; \mathbf{d}_1) \geq latency(\mathbf{x}_2; \mathbf{d}_1)$ and $latency(\mathbf{x}_1; \mathbf{d}_2) \geq latency(\mathbf{x}_2; \mathbf{d}_2)$ hold simultaneously for any two neural architectures $\mathbf{x}_1 \in \mathcal{X}$ and $\mathbf{x}_2 \in \mathcal{X}$, then the two devices $\mathbf{d}_1$ and $\mathbf{d}_2$ are said to satisfy latency monotonicity. ■

While latency monotonicity might seem intuitive, it has not been demonstrated in the context of device-aware NAS for mobile inference. In practice, it is difficult to perfectly satisfy latency monotonicity by two different devices, i.e., the latency rankings of all CNN models on two devices may not be entirely identical.

To quantify the degree of latency monotonicity, we use the metric of Spearman's Rank Correlation Coefficient (**SRCC**), which lies between -1 and 1 and assesses statistical dependence between the rankings of two variables using a monotonic function. The greater the SRCC of CNN latencies on two devices, the better the latency monotonicity. SRCC of 0.8 to 1.0 is usually viewed as *strongly* dependent in terms of monotonicity [3].

[2]Due to non-convexity, the obtained neural architectures by solving Eqns. (1)(2) may not be globally Pareto-optimal in a strict sense. Also, in practice, the Pareto-optimal set has a finite number of architectures.

4.2. Latency Monotonicity on Mobile Devices

While inference speedups through co-processors like GPUs and purpose-built accelerators are possible on today's mobile devices, CPUs still remain as the mainstream for mobile inference, due in part to the practical challenge of programmability (e.g., inference in Facebook app) [35]. Next, we empirically show latency monotonicity by running inference on CPUs of our own devices as well as leveraging third-party results (including a wide range of devices with both GPUs and CPUs). We also show roofline analysis in the appendix to further explain latency monotonicity.

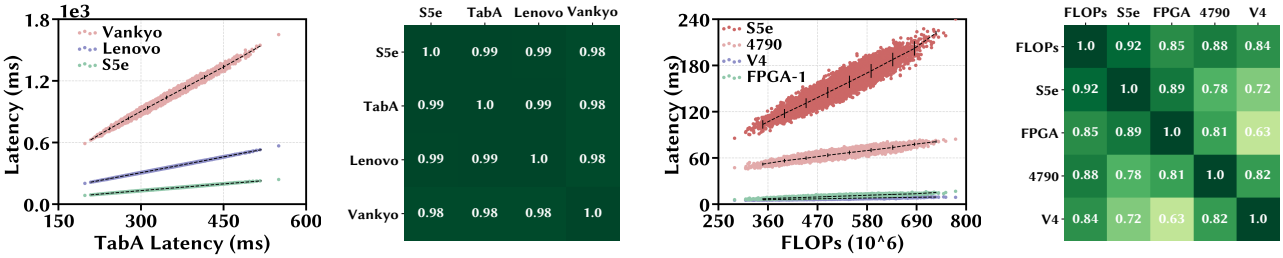
4.2.1 Empirical Measurement

We first empirically measure the actual latencies of CNN models on four mobile devices (detailed specification in the appendix): Samsung Galaxy **S5e**, **TabA**, **Lenovo** Moto Tab, and **Vankyo** MatrixPad Z1 (a low-end device). We randomly sample 10000 models from the MobileNet-V2 [26] backbone with variable depths of "2, 3, 4" in each stage, variable filter size of "3, 5, 7" in each convolutional layer, and variable expansion ratio of "3, 4, 6" in each block (details in Section 6). Then, we deploy these models on the four devices and calculate their average inference latencies. We show the actual latencies on S5e, Lenovo, Vankyo versus TabA in Fig. 2(a), where each dot represents one CNN model.

We see that when the sampled CNN models run faster on TabA, they also become faster on the other devices. In Fig. 2(a), the maximum standard deviation (denoted by the vertical line within each bin) is 1.3% for Vankyo, while it is negligibly 0.6% and 0.84% for Lenovo and S5e. Thus, latency monotonicity is well preserved on these devices. We further show the SRCC values of these 10000 sampled model latencies on our four mobile device in Fig. 2(b) with heatmap. We see that SRCC between any pair of our mobile devices is larger than 0.98, implying an extremely strong monotonic relationship.

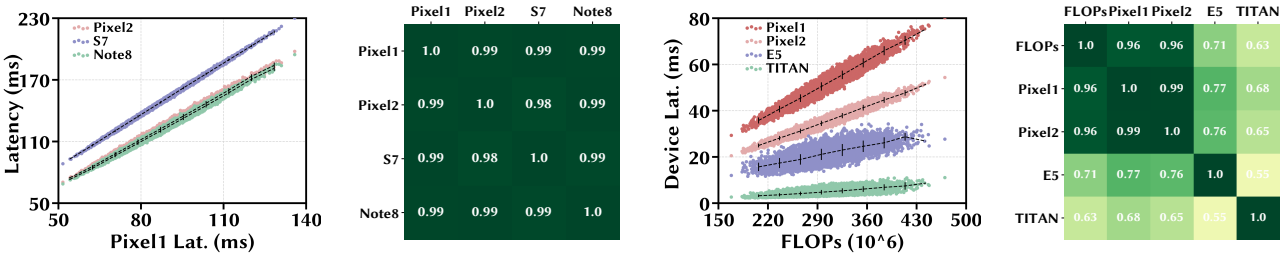
While the focus of MonoNAS is on mobile devices, it is worth exploring whether cross-platform latency monotonicity exists (e.g., mobile vs. FPGA). Specifically, we choose one FPGA (Xilinx ZCU 102), one desktop CPU (Intel Core i7-**4790**), and one desktop GPU (Tesla **T4**) as the cross-platform devices, whose details are in the appendix.

In addition, FLOP is also often used as an indicator of CNN model latency [30], while it is a common belief that FLOPs do not necessarily reflect its actual latency ranking [13, 34]. Thus, we include FLOP-latency monotonicity and SRCCs, by considering a *virtual* device whose inference latency is proportional to model FLOPs. We show the latency monotonicity results and SRCC values for the same set of 10000 models in Figs. 2(c) and 2(d), respectively. It can be seen that while latency rankings are still moder-



(a) Cross-device Monotonicity (b) Cross-device SRCC (c) Cross-platform Monotonicity (d) Cross-platform SRCC

Figure 2: Empirical measurement. (a)(c) Black vertical lines denote the standard deviation of latency data points within each bin, with the center denoting the average. (b)(d) SRCC of 10000 sampled model latencies on different pairs of devices.



(a) Latency Monotonicity (b) SRCC (c) Latency Monotonicity (d) SRCC

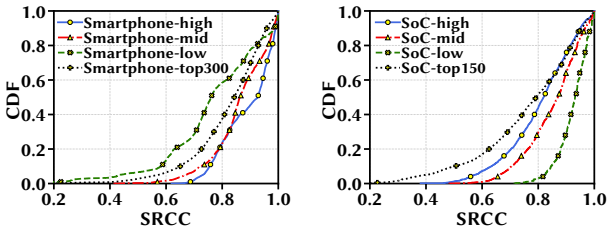
Figure 3: Latency monotonicity on third-party latency predictors [8,9]. (a)(b) and (c)(d) use different search spaces and DNN acceleration libraries.

ately correlated for cross-platform devices, the SRCC values are lower than in the case of mobile device pairs (shown in Fig. 2(b)), since mobile devices differ significantly from desktops/FPGAs. Interestingly, the rankings of FLOPs and latencies on our mobile device are quite highly correlated (SRCC $= 0.92$), partly because the MobileNet-V2 backbone is specifically designed for mobile inference. Thus, FLOPs may also replace the latency predictor for device-aware NAS on mobiles (more results in Section 6.3).

4.2.2 Third-party latency predictor

To corroborate our own measurement, we examine latency monotonicity by leveraging third-party latency predictors built for other devices. Specifically, we obtain latency lookup tables for four mobile devices [8]: Google **Pixel1**, **Pixel2**, Samsung Galaxy **S7** edge, **Note8** in the MobileNet-V2 space with stage widths "32, 16, 24, 48, 80, 104, 192, 320, 1280". In addition, we obtain from [9] latency lookup tables for four cross-platform devices (used in [11]): Google **Pixel1**, **Pixel2**, **TITAN** Xp, **E5**-2640 v4 in the MobileNet-V2 space with different stage widths "32, 16, 24, 40, 80, 96, 192, 320, 1280" and measured with the MKL-DNN library. Note that latency predictors are very accurate (e.g., with an root mean squared error of less than 1% of the average) [5, 11, 13, 36], which we also confirm through our own experiments in the appendix.

We randomly sample 10000 models in each search space



(a) Top-300 Mobile Phones (b) Top-150 Mobile Phones

Figure 4: CDF of SRCC values of DNN models on mobile phones and SoCs. The annotation "high/mid/low" represents the highest/middle/lowest 33.3% of the devices.

with variable depths of "2, 3, 4" in each stage, variable filter sizes of in "3, 5, 7" in each convolutional layer, and variable expansion ratios of "3, 4, 6" in each block. We show the results in Fig. 3, which are in line with our experiments: latency monotonicity among mobile devices is strong (> 0.95), while FLOP-latency ranking correlation for mobile devices is also quite strong but cross-platform latency monotonicity degrades.

4.2.3 AI-Benchmark Data

To examine latency monotonicity at scale, we resort to the AI-Benchmark dataset showing DNN inference latency

measurements on diverse hardware [2]. Considering top-300 smartphones (ranging from Huawei Mate 40 Pro to Sony Xperia Z3) and top-150 mobile SoCs (ranging from HiSilicon Kirin 9000 to MediaTek Helio P10) ranked by the metric "AI-Score" [18], we show in Fig. 4 the SRCC values of latency rankings based on the 22 DNN models including both floating-point and quantized models (e.g., MobileNet-V2-INT8 and MobileNet-V2-FP16) listed in the dataset.

We see that latency monotonicity is well preserved at scale. For example, among the top-100 mobile phones, SRCC values among 50+% of *any* device pairs are higher than 0.9 (a very strong ranking correlation). While the AI-Benchmark dataset is built for orthogonal purposes, the SRCC values, combined with our own experiments and third-party latency lookup tables, show good latency monotonicity for mobile devices at scale.

4.3. Latency Monotonicity on Other Platforms

Going beyond the mobile platform, we also perform experiments to show latency monotonicity on other platforms: FPGA, desktop CPUs and GPUs. We further show the latency monotonicity for all the 195 desktops listed in the AI-Benchmark dataset [1]. Our results are provided in the appendix and show that latency monotonicity exists within each platform (e.g., SRCC > 0.9), but degrades for cross-platform devices. This implies that MonoNAS can also be broadly applicable for other platforms (e.g., FPGA and desktops).

5. MonoNAS: Scalable Cross-Device NAS

We now present MonoNAS to efficiently scale up cross-device NAS based on latency monotonicity.

5.1. Theoretical Support

We first offer theoretical justification for MonoNAS.

Theorem 5.1 *If two mobile devices* $\mathbf{d}_1 \in \mathcal{D}$ *and* $\mathbf{d}_2 \in \mathcal{D}$ *strictly satisfy latency monotonicity, then they have the same set of Pareto optimal architectures, i.e.,* $\mathcal{P}_{\mathbf{d}_1} = \mathcal{P}_{\mathbf{d}_2}$*, where* $\mathcal{P}_{\mathbf{d}_i} = \{\mathbf{x}^*(\overline{L}_{\mathbf{d}_i}; \mathbf{d}_i)$*, for* $\overline{L}_{\mathbf{d}_i} \in [\overline{L}_{\mathbf{d}_i,\min}, \overline{L}_{\mathbf{d}_i,\max}]\}$ *for* $i = 1, 2$.

Proof: Define $\mathcal{X}_{\overline{L}_{\mathbf{d}_1},\mathbf{d}_1}$ as the set of architectures satisfying $latency(\mathbf{x}; \mathbf{d}_1) \leq \overline{L}_{\mathbf{d}_1}$. By latency monotonicity, we can find another constraint $\overline{L}_{\mathbf{d}_2}$ such that $\mathcal{X}_{\overline{L}_{\mathbf{d}_1},\mathbf{d}_1} = \mathcal{X}_{\overline{L}_{\mathbf{d}_2},\mathbf{d}_2}$. In other words, the latency constraint $latency(\mathbf{x}; \mathbf{d}_1) \leq \overline{L}_{\mathbf{d}_1}$ is equivalent to $latency(\mathbf{x}; \mathbf{d}_2) \leq \overline{L}_{\mathbf{d}_2}$. Therefore, device-aware NAS formulated in Eqns. (1)(2) for devices $\mathbf{d}_1$ and $\mathbf{d}_2$ are equivalent, sharing the same set of Pareto-optimal architectures. ■

Theorem 5.1 lays the foundation for MonoNAS: for any two mobile devices satisfying latency monotonicity, we only need to run device-aware NAS on *one* device, reducing the total search cost yet without losing optimality.

Algorithm 1: MonoNAS for Cross-device NAS

Input: Set of proxy devices $\mathcal{D}_0$ and their Pareto-optimal architectures $\mathcal{P}_{\mathbf{d}_0}$ for $\mathbf{d}_0 \in \mathcal{D}_0$, sample architecture set $\mathcal{A}$, SRCC threshold S_{th}, target latency constraint $\overline{L}_{\mathbf{d}}$

Output: Optimal architecture $\mathbf{x}^*(\overline{L}_{\mathbf{d}}; \mathbf{d})$

Initialize: Set $\mathbf{d}_0 = null$ and SRCC $S_{\mathbf{d}_0,d} = -1$;

//select a proxy device $\mathbf{d}_0$

for $\mathbf{d}' \in \mathcal{D}_0$ **do**
  Estimate SRCC $S_{\mathbf{d}',d}$ for sample architectures in $\mathcal{A}$;
  if $S_{\mathbf{d}',d} > S_{\mathbf{d}_0,d}$ *and* $S_{\mathbf{d}',d} \geq S_{th}$ **then**
    $\mathbf{d}_0 = \mathbf{d}'$;
end
if $\mathbf{d}_0 \neq null$ **then**
  Set $L_{\min} = \overline{L}_{\mathbf{d}_0,\min}$ and $L_{\max} = \overline{L}_{\mathbf{d}_0,\max}$;
  for i= *1 to* $Max\_Iterate$ **do**
    $L = \frac{L_{\min}+L_{\max}}{2}$;
    Measure $latency(\mathbf{x}^*(L; \mathbf{d}_0); \mathbf{d})$;
    if $latency(\mathbf{x}^*(L; \mathbf{d}_0); \mathbf{d}) \geq \overline{L}_{\mathbf{d}} + \delta$ **then**
      $L_{\max} = L$;
    else if $latency(\mathbf{x}^*(t^*); \mathbf{d}) \leq \overline{L}_{\mathbf{d}} - \delta$ **then**
      $L_{\min} = L$;
    else
      Break;
  end
  $\mathbf{x}^*(\overline{L}_{\mathbf{d}}; \mathbf{d}) \leftarrow \mathbf{x}^*(L; \mathbf{d}_0)$
else
  NAS on target device $\mathbf{d}$ and $\mathcal{D}_0 \leftarrow \mathcal{D}_0 \cup \{\mathbf{d}\}$;
  Save $\mathcal{P}_{\mathbf{d}} = \{\mathbf{x}^*(l; \mathbf{d}),$ for $l \in [\overline{L}_{\mathbf{d},\min}, \overline{L}_{\mathbf{d},\max}]\}$;
return $\mathbf{x}^*(\overline{L}_{\mathbf{d}}; \mathbf{d})$;

5.2. Algorithm of MonoNAS

Based on Theorem 5.1, given the set $\mathcal{P}_{\mathbf{d}_0}$ of architectures that are Pareto-optimal for one device (referred to as *proxy* device $\mathbf{d}_0$), we only need to search over this Pareto-optimal set for a new device $\mathbf{d}$, if the two devices satisfy latency monotonicity. Compared to the full space $\mathcal{X}$, the new search space $\mathcal{P}_{\mathbf{d}_0}$ is smaller by orders of magnitude.

MonoNAS is described in Algorithm 1 and illustrated in Fig. 5. Concretely, MonoNAS includes a pool of proxy devices and, given a new target device, selects a proxy device and quickly returns an optimal architecture.

Proxy device pool. We pre-build a small pool of proxy devices, which can be expanded subsequently. A proxy device should preferably have good latency monotonicity with many target devices. In Section 6.3.2, we show that as few as 10 proxy mobile devices can be enough to cover 300 mobile devices.

For each proxy device, we perform NAS to obtain its Pareto-optimal architectures under various latency con-

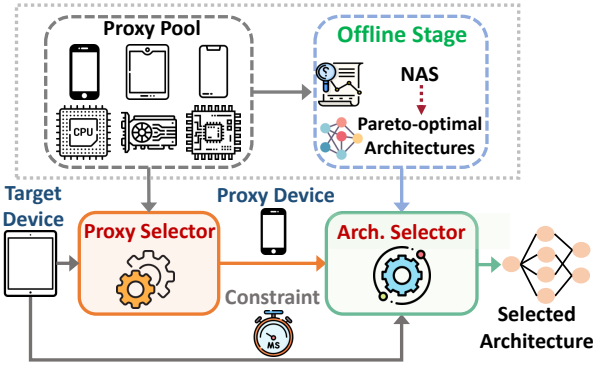


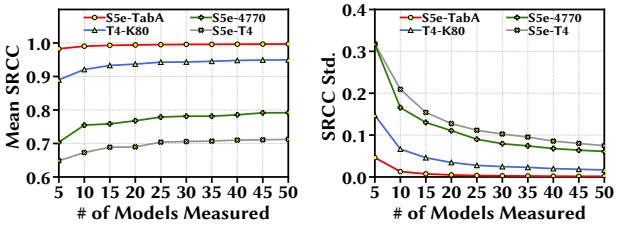
Figure 5: Block diagram of MonoNAS.

straints. Like in the existing architecture scaling techniques [12, 30], MonoNAS uses a state-of-the-art device-aware NAS algorithm as the base. For example, we can build an accuracy predictor and a latency lookup table, and then apply evolutionary search or reinforcement learning [5, 6, 13, 32, 36]. Alternatively, following [29], we can also use real accuracy and latency measurement for NAS on each proxy device, yielding a set of optimal architectures (e.g., Fig. 6 in [29]).

Proxy device selection. For each target device, we select a proxy device that has the best latency monotonicity exceeding a threshold (measured in SRCC). To this end, we estimate the SRCC based on a small set of sample architectures. In Fig. 6, we can see that latency measurement based on a few sample architectures is enough to reliably estimate the SRCC value: if we set 0.9 as the SRCC threshold, then with about 10 sample architectures, we can decide whether or not strong latency monotonicity exists between the target device and a proxy device. Compared to the cost of building a latency lookup table (e.g., 5k sample architectures in [11]), the cost of estimating SRCC for proxy device selection is negligible. If no proxy device with a satisfactory latency monotonicity is found, then we add the target device into the pool as a new proxy device and perform NAS to obtain its optimal architectures.

Architecture selection. After proxy device selection, using bisection search, the corresponding latency constraint L on the *proxy* device $\mathbf{d}_0$ can be found very quickly such that its Pareto-optimal architecture $\mathbf{x}^*(L; \mathbf{d}_0)$ also satisfies the latency constraint on the target device $\mathbf{d}$. In practice, a few iterations are enough in most cases (e.g., with 10 iterations, the granularity of L is already less than 1/1000 of $\overline{L}_{\mathbf{d}_0,\max} - \overline{L}_{\mathbf{d}_0,\min}$).

In summary, with latency monotonicity, MonoNAS can find the optimal architecture for a new target device at a negligible cost. Even in the unlikely event that no proxy device meets the SRCC threshold, MonoNAS rolls back to the existing per-device NAS without increasing the search cost for a new target device.



(a) Average Estimated SRCC (b) Standard Deviation

Figure 6: SRCC estimation for proxy device selection. X-axis denotes the number of sample architecture we randomly select per run. We use 1000 runs to calculate the mean and standard deviation. "x-y" means the device pair is (x, y).

6. Experiment

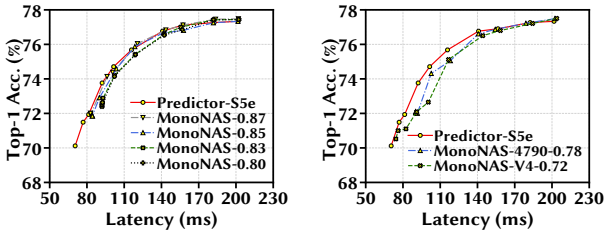
In this section, we present our experiment setup, baseline approaches, and results. Our results highlight that MonoNAS can effectively find out (almost) the same Pareto-optimal architectures on target devices as those obtained using device-specific NAS.

6.1. Setups

Search Space. As in [11], the backbone of our CNN architecture is MobileNet-V2 with multiplier 1.3, with the channel number in each block fixed. The search space consists of depth of each stage, kernel size of convolutional layers, and expansion ratio of each block. The depth can be chosen from "2, 3, 4", kernel size can be "3, 5, 7", and candidate expansion ratios are "3, 4, 6". There are five stages whose configurations can be searched. An example searched architecture is shown in the appendix.

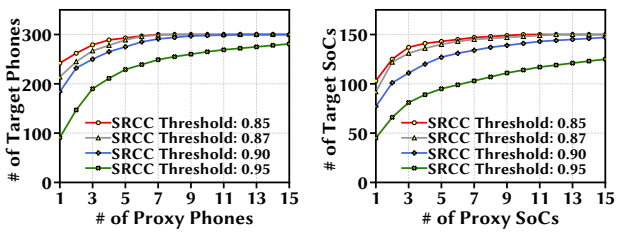
Accuracy/Latency Predictor. To find Pareto-optimal architectures on each proxy device, we adopt predictor-based NAS based on evolutionary search [6]. Specifically, we build a latency lookup table with latencies of various layers and operators for each proxy device. For each sample model, we profile the average latency of 1000 runs. We use a single thread for running the TensorFlow Lite interpreter by default. As corroborated by prior studies [13, 32, 36], our measurement (in the appendix) shows that the predicted average latency is almost identical to the actual value. Our accuracy predictor is a neural network with four fully-connected layers and updated with 176 samples on top of the predictor used in [10]. The accuracy predictor takes a 128-dimensional feature vector (which is converted from a 21-dimensional architecture configuration within the search space) as input.

Evolutionary Search. Our parameter settings are: population size is 1000, parent ratio is 0.25, mutation probability is 0.1, mutation ratio is 0.25, and we search for 50 gen-



(a) Synthetic Devices (b) Cross-platform Devices

Figure 7: Results on the target device S5e, using four synthetic devices, Tesla V4 and Intel Core i7-4790 as proxy devices, respectively. "Predictor-S5e" is the baseline #1, and the legend "...-x" means the respective SRCC is x.



(a) Mobile phone (b) Mobile SoC

Figure 8: Estimated total search cost.

erations given each latency constraint. Evolutionary search takes less than 30 seconds for each run.

Model Evaluation. For a searched architecture, the actual model performance is measured. We consider the *Once-For-All* network [10] as a super net that has the same search space as ours. We evaluate models on the ImageNet validation dataset [14], which consists of 50000 images in 1000 classes.

6.2. Baselines

To highlight the benefit of MonoNAS, we consider the following baseline approaches.

#1: Device-specific NAS. For each target device, we use the same predictor-based NAS algorithm described in Section 6.1. Due to the high cost of building a latency predictor for each device, this approach has a total search cost of $\mathcal{O}(n)$ for n devices [5, 13].

#2: Heuristic Model Scaling. There are different ways to scale a CNN to meet different latency constraints: e.g., adjusting the network depth and/or width [29, 30]. Since the number of channels in our backbone network is fixed, we heuristically scale the depth of a Pareto-optimal architecture on a proxy device and transfer it to new target devices. More specifically, we increase (for higher accuracy) or reduce (for smaller latency) the depth by up to two blocks. This approach has $\mathcal{O}(1)$ complexity.

6.3. Results

6.3.1 SRCC Threshold

In MonoNAS, a proxy device is selected only when the SRCC of latency rankings on this proxy device and a target device exceeds a threshold. Intuitively, the higher threshold, the better latency monotonicity and architecture selection, but the less likely to have a qualified proxy device.

We use S5e as the target device and synthesize proxy devices by adding random noises to Vankyo's latency lookup tables. The resulting SRCCs between S5e and the synthetic proxy devices are 0.80, 0.83, 0.85, and 0.87, respectively. Then, we show the results of using these four proxy devices in Fig. 7(a). Compared to device-specific NAS (i.e., "Predictor-S5e"), the performance of searched architectures using the synthetic devices are nearly identical when SRCC is 0.85 or 0.87, and only slightly worse when SRCC is 0.83 or 0.80.

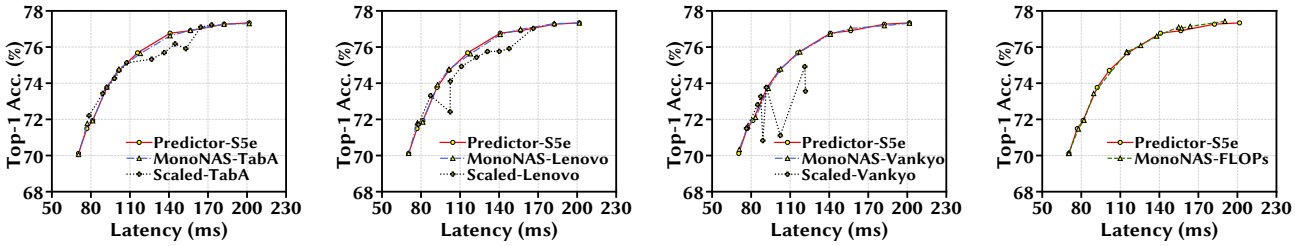
Next, we turn to using non-mobile proxy devices — Tesla V4 and Intel Core i7-4790 on desktops — which have SRCC values of 0.72 and 0.78 with the target device S5e, respectively. We see from Fig. 7(b) that, due to the low SRCC and bad latency monotonicity, MonoNAS cannot transfer optimal architectures from proxy devices to a target device.

While latency monotonicity is not strictly satisfied, our empirical results show that 0.85 or 0.87 (still viewed as strong monotonicity in general [3]) is a reasonable SRCC threshold for MonoNAS without noticeably affecting the performance. This can be explained by noting the imperfection in accuracy predictor, randomness in evolutionary search and non-convexity of NAS, which altogether may compensate for the imperfect satisfaction of latency monotonicity (i.e., SRCC < 1). For example, we show in appendix the actual and predicted accuracies, which have a SRCC of 0.903 and root mean squared error of 1.11%.[3] Thus, although latency monotonicity is not perfect, it is masked by the imperfection in other parts of the NAS process.

6.3.2 Total Search Cost at Scale

We now examine the total search cost of using MonoNAS at scale and use the estimated SRCC values for the top-300 mobile phones and top-150 mobile SoCs based on AI-Benchmark [18]. Specifically, we greedily find the proxy devices that have the best latency monotonicity with other devices, and show the number of target devices that can meet the SRCC threshold. We see from Fig. 8 that with

[3] Accuracy predictors used in other NAS studies have similar performance, e.g., with a mean square error between 1.69-13.92% [13].



(a) TabA as proxy device (b) Lenovo as proxy device (c) Vankyo as proxy device (d) FLOPs as proxy

Figure 9: (a)(b)(c) Results on the target device S5e, using TabA, Lenovo, and Vankyo as proxy devices, respectively. "Preditor-S5e" is the baseline #1, "MonoNAS-x" means using MonoNAS with x as the proxy device, and "Scaled-x" means heuristic scaling applied to x's Pareto-optimal architecture. (d) Searched architectures on S5e using device-specific NAS and FLOPs-based NAS.

about 10 proxy devices, the vast majority of target devices can be covered (even with a SRCC threshold of 0.9). Concretely, to find Pareto-optimal architectures for 300 mobile phones, the existing device-aware NAS has a total search cost of $\mathcal{O}(300)$, while MonoNAS has a total search cost of $\mathcal{O}(10)$ yet without losing model performance.

6.3.3 Performance of Searched Architectures

We compare the measured top-1 accuracy on ImageNet versus average inference latency of searched architectures on each target device under different baselines.

Cross-device MonoNAS. Fig. 9(a) shows the result for the target device S5e, using TabA as the proxy device with SRCC of 0.99. We see that MonoNAS can result in almost the same $(accuracy, latency)$ tradeoff as device-specific NAS (i.e., "Predictor-S5e") but the additional cost of MonoNAS for each a target device is negligible. Further, we see that despite its $\mathcal{O}(1)$ complexity, heuristic scaling (baseline #2) can result in really bad architectures without performance guarantees. The results on S5e with Lenovo and Vankyo as proxy devices are similar and presented in Fig. 9(b) and Fig. 9(c), respectively.

FLOPs as proxy. FLOP-based NAS can be viewed as special device-aware NAS on a *virtual* proxy device, whose latency is proportional to the model FLOPs. Given various FLOP constraints, we use the same evolutionary search as in previous experiments to find Pareto-optimal architectures, by only replacing the inference latency with model FLOPs. Given the high SRCC between FLOP and latency rankings on S5e (0.92 in Fig. 2(d)), we see from Fig. 9(d) that FLOP-based Pareto-optimal architectures are also performing very well on S5e.

When TabA, Lenovo, and Vankyo are target devices, the corresponding results are similar and shown in the appendix. Our results empirically demonstrate the efficiency of MonoNAS for scaling up NAS on diverse mobile devices (almost) without losing optimality.

7. Conclusion

In this paper, we efficiently scale up NAS for *diverse* mobile devices. Concretely, we demonstrate the widely-existing latency monotonicity among different devices and propose MonoNAS, which optimizes neural architectures for mobile devices at a much lower total cost comparable to that of the existing per-device NAS. We conduct experiments with different mobile devices on ImageNet. Our experimental results highlight that, even with approximate latency monotonicity, MonoNAS can find Pareto-optimal architectures while incurring a very low total search cost.

References

[1] AI-Benchmark. Ai benchmark for Windows, Linux and macOS. https://ai-benchmark.com/ranking_deeplearning_detailed.html. 5, 11, 12

[2] AI-Benchmark. Performance of mobile phones. http://ai-benchmark.com/ranking_detailed.html. 5

[3] Haldun Akoglu. User's guide to correlation coefficients. *Turkish Journal of Emergency Medicine*, 18(3):91 – 93, 2018. 3, 7

[4] Gabriel Bender, Pieter-Jan Kindermans, Barret Zoph, Vijay Vasudevan, and Quoc Le. Understanding and simplifying one-shot architecture search. In *ICML*, 2018. 2

[5] Gabriel Bender, Hanxiao Liu, Bo Chen, Grace Chu, Shuyang Cheng, Pieter-Jan Kindermans, and Quoc V. Le. Can weight sharing outperform random architecture search? an investigation with tunas. In *CVPR*, 2020. 1, 2, 3, 4, 6, 7

[6] Hadjer Benmeziane, Kaoutar El Maghraoui, Hamza Ouarnoughi, Smail Niar, Martin Wistuba, and

Naigang Wang. A comprehensive survey on hardware-aware neural architecture search, 2021. 1, 2, 6

[7] Ermao Cai, Da-Cheng Juan, Dimitrios Stamoulis, and Diana Marculescu. *NeuralPower*: Predict and deploy energy-efficient convolutional neural networks. In *ACML*, 2017. 2

[8] Han Cai. Latency lookup tables of mobile devices. https://file.lzhu.me/hancai/. 4

[9] Han Cai. Latency lookup tables of mobile devices and gpus. https://file.lzhu.me/LatencyTools/tvm_lut/. 4

[10] Han Cai, Chuang Gan, and Song Han. Once for all: Train one network and specialize it for efficient deployment. In *ICLR*, 2019. 2, 6, 7, 12, 13

[11] Han Cai, Ligeng Zhu, and Song Han. ProxylessNas: Direct neural architecture search on target task and hardware. In *ICLR*, 2019. 1, 2, 3, 4, 6

[12] Grace Chu, Okan Arikan, Gabriel Bender, Weijun Wang, Achille Brighton, Pieter-Jan Kindermans, Hanxiao Liu, Berkin Akin, Suyog Gupta, and Andrew Howard. Discovering multi-hardware mobile models via architecture search, 2020. 1, 2, 3, 6, 15

[13] Xiaoliang Dai, Peizhao Zhang, Bichen Wu, Hongxu Yin, Fei Sun, Yanghan Wang, Marat Dukhan, Yunqing Hu, Yiming Wu, Yangqing Jia, et al. ChamNet: Towards efficient network design through platform-aware model adaptation. In *CVPR*, 2019. 1, 2, 3, 4, 6, 7, 12, 13

[14] Jia Deng, Wei Dong, Richard Socher, Li-Jia Li, Kai Li, and Li Fei-Fei. Imagenet: A large-scale hierarchical image database. In *CVPR*, 2009. 7, 14

[15] Thomas Elsken, Jan Hendrik Metzen, and Frank Hutter. Neural architecture search: A survey. *Journal of Machine Learning Research*, 20(55):1–21, 2019. 2

[16] 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. 2

[17] Mark Hill and Vijay Janapa Reddi. Gables: A roofline model for mobile SoCs. In *HPCA*, 2019. 2, 16, 17

[18] Andrey Ignatov, Radu Timofte, Andrei Kulik, Seungsoo Yang, Ke Wang, Felix Baum, Max Wu, Lirong Xu, and Luc Van Gool. Ai benchmark: All about deep learning on smartphones in 2019. In *ICCVW*, 2019. 5, 7

[19] Weiwen Jiang, Lei Yang, Sakyasingha Dasgupta, Jingtong Hu, and Yiyu Shi. Standing on the shoulders of giants: Hardware and neural architecture co-search with hot start. *TCAD*, 2020. 11

[20] Sheng-Chun Kao, Arun Ramamurthy, and Tushar Krishna. Generative design of hardware-aware dnns. 2020. 2

[21] Chenxi Liu, Barret Zoph, Maxim Neumann, Jonathon Shlens, Wei Hua, Li-Jia Li, Li Fei-Fei, Alan Yuille, Jonathan Huang, and Kevin Murphy. Progressive neural architecture search. In *ECCV*, 2018. 2

[22] Qing Lu, Weiwen Jiang, Xiaowei Xu, Yiyu Shi, and Jingtong Hu. On neural architecture search for resource-constrained hardware platforms. In *ICCAD*, 2019. 2

[23] Renqian Luo, Fei Tian, Tao Qin, Enhong Chen, and Tie-Yan Liu. Neural architecture optimization. In *NIPS*, 2018. 2

[24] Xuefei Ning, Wenshuo Li, Zixuan Zhou, Tianchen Zhao, Yin Zheng, Shuang Liang, Huazhong Yang, and Yu Wang. A surgery of the neural architecture evaluators. *arXiv preprint arXiv:2008.03064*, 2020. 2

[25] Binxin Ru, Xingchen Wan, Xiaowen Dong, and Michael Osborne. Neural architecture search using bayesian optimisation with weisfeiler-lehman kernel. *arXiv preprint arXiv:2006.07556*, 2020. 2

[26] Mark Sandler, Andrew Howard, Menglong Zhu, Andrey Zhmoginov, and Liang-Chieh Chen. Mobilenetv2: Inverted residuals and linear bottlenecks. In *CVPR*, 2018. 3, 16

[27] Han Shi, Renjie Pi, Hang Xu, Zhenguo Li, James T. Kwok, and Tong Zhang. Multi-objective neural srchitecture search via predictive network performance optimization. *arXiv preprint arXiv:1911.09336*, 2019. 2

[28] D. Stamoulis, E. Cai, D. Juan, and D. Marculescu. HyperPower: Power- and memory-constrained hyperparameter optimization for neural networks. In *DATE*, 2018. 2

[29] Mingxing Tan, Bo Chen, Ruoming Pang, Vijay Vasudevan, Mark Sandler, Andrew Howard, and Quoc V. Le. MnasNet: Platform-aware neural architecture search for mobile. In *CVPR*, 2019. 1, 2, 3, 6, 7, 15

[30] Mingxing Tan and Quoc Le. EfficientNet: Rethinking model scaling for convolutional neural networks. In *ICML*, 2019. 1, 2, 3, 6, 7, 15

[31] Linnan Wang, Yiyang Zhao, Yuu Jinnai, Yuandong Tian, and Rodrigo Fonseca. AlphaX: exploring neural architectures with deep neural networks and monte carlo tree search. *arXiv preprint arXiv:1903.11059*, 2019. 2

[32] Tianzhe Wang, Kuan Wang, Han Cai, Ji Lin, Zhijian Liu, Hanrui Wang, Yujun Lin, and Song Han. APQ: Joint search for network architecture, pruning and quantization policy. In *CVPR*, 2020. 1, 2, 3, 6, 12

[33] Samuel Williams, Andrew Waterman, and David Patterson. Roofline: an insightful visual performance model for multicore architectures. *Communications of the ACM*, 2009. 16

[34] Bichen Wu, Xiaoliang Dai, Peizhao Zhang, Yanghan Wang, Fei Sun, Yiming Wu, Yuandong Tian, Peter Vajda, Yangqing Jia, and Kurt Keutzer. FBNet: Hardware-aware efficient ConvNet design via differentiable neural architecture search. In *CVPR*, 2019. 1, 2, 3

[35] Carole-Jean Wu, David Brooks, Kevin Chen, Douglas Chen, Sy Choudhury, Marat Dukhan, Kim Hazelwood, Eldad Isaac, Yangqing Jia, Bill Jia, Tommer Leyvand, Hao Lu, Yang Lu, Lin Qiao, Brandon Reagen, Joe Spisak, Fei Sun, Andrew Tulloch, Peter Vajda, Xiaodong Wang, Yanghan Wang, Bram Wasti, Yiming Wu, Ran Xian, Sungjoo Yoo, and Peizhao Zhang. Machine learning at Facebook: Understanding inference at the edge. In *HPCA*, 2019. 1, 2, 3

[36] Tien-Ju Yang, Andrew Howard, Bo Chen, Xiao Zhang, Alec Go, Mark Sandler, Vivienne Sze, and Hartwig Adam. Netadapt: Platform-aware neural network adaptation for mobile applications. In *ECCV*, 2018. 1, 2, 3, 4, 6, 12, 13

[37] Barret Zoph and Quoc V Le. Neural architecture search with reinforcement learning. In *ICLR*, 2017. 2

Supplementary Material

In the supplementary material, we provide details of our mobile devices and FPGAs, latency monotonicity results on non-mobile platforms, and additional experiment results (Sections A–D). Then, we investigate latency monotonicity on our mobile devices for a different search space based on ResNet-50 (Section E). Finally, we conclude by offering roofline analysis (Section F).

A. Device Specification

Here, we list the detailed specification for mobile devices and FPGAs.

Mobile Devices. Table 1 summarizes the detailed specifications of the four mobile devices (Samsung Galaxy **S5e**, **TabA**, **Lenovo** Moto Tab, and **Vankyo** MatrixPad Z1 (a low-end device)) we experiment on in this work. These devices have major differences in terms of several specifications, such as operating systems, chipset, CPU and GPU, which can affect the latencies. As a result, the absolute latency values on these devices are very different.

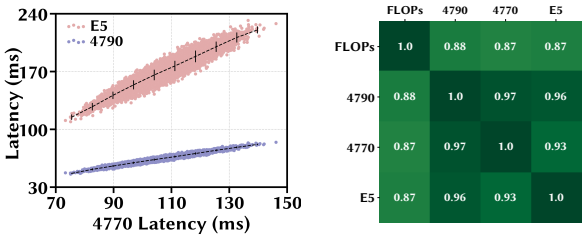
FPGAs. We configure nine subsystems for an Xilinx ZCU 102 FPGA board to create nine different FPGAs following the hardware design space in [19]. The detailed configuration is shown in Table 2. "Computation Design" is the computation subsystem design, T_m, T_n are loop tiling parameters for input and output feature maps, and $T_m(d)$ denotes the parameter for depth-wise separable convolution. "Communication Design" represents the communication subsystem design, where I_p, O_p, and W_p are communication ports allocated for input feature maps, output feature maps and weights, respectively.

B. Latency Monotonicity on Other Platforms

While the focus of MonoNAS is mobile devices, we show the results of latency monotonicity on other hardware platforms including CPUs, GPUs, and FPGAs.

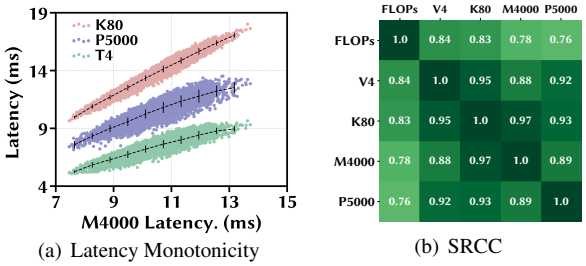
CPU, GPU, and FPGA. We build latency lookup tables for three desktop CPUs: Intel Core i7-**4790**, Intel Core i7-**4770** HQ, and **E5**-2673 v3. In addition, we consider four NVIDIA GPUs: Tesla **T4**, Tesla **K80**, Quadro **M4000**, and Quadro **P5000**. For FPGAs, we measure CNN model latency on nine Xilinx ZCU 102 boards shown in Table 2, using the performance model in [19].

For the three different platforms (i.e., CPU, GPU, and FPGA), we consider latencies for the same 10000 models as in Fig. 2, and plot the results in Figs. 10, 11, and 12, respectively. We see that *within* each platform, latency monotonicity is generally very well preserved, with most SRCC values of 0.9+. As intuitively expected, the cross-platform latency monotonicity is not well satisfied in general, with an example SRCC heatmap shown in Fig. 2(d) in Section 4.



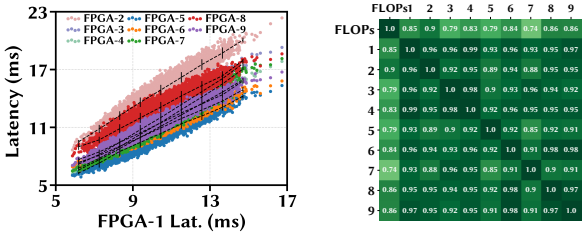
(a) Latency Monotonicity (b) SRCC

Figure 10: Latency monotonicity on desktop CPUs. (a) Black vertical lines denote the standard deviation of latency data points within each bin, with the center denoting the average. (b) SRCC of 10000 sampled model latencies on different pairs of devices.



(a) Latency Monotonicity (b) SRCC

Figure 11: Latency monotonicity on desktop GPUs. (a) Black vertical lines denote the standard deviation of latency data points within each bin, with the center denoting the average. (b) SRCC of 10000 sampled model latencies on different pairs of devices.



(a) Latency Monotonicity (b) SRCC

Figure 12: Latency monotonicity on FPGAs. (a) Black vertical lines denote the standard deviation of latency data points within each bin, with the center denoting the average. (b)SRCC of 10000 sampled model latencies on different pairs of FGPAs.

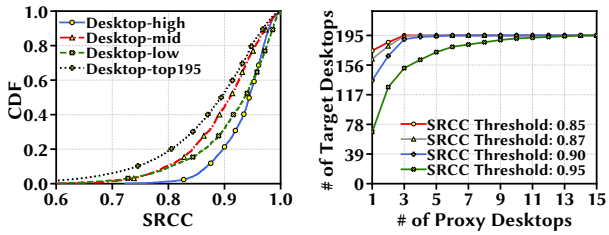
AI-Benchmark Dataset for Desktops. Next, in addition to latency monotonicity at scale for mobile devices and SoCs (shown in Fig. 4), we resort to the AI-Benchmark dataset showing DNN inference latency measurements on 195 different desktops (ranging from Tesla V100 SXM2 32Gb to Intel Core i3-3110M) [1]. We show in Fig. 13 the

Device	Abbrev.	Chipset	CPU (GHz)	Cores	RAM (GB)	RAM Freq. (MHz)	Peak Perf. (GFLOPs/sec)	Mem. Bandwidth (GB/sec)
Samsung Galaxy Tab S5e	**S5e**	Snapdragon 670	2	8	4	1866	40.6	14.93
Samsung Galaxy Tab A	**TabA**	Snapdragon 429	2	4	2	933	15.3	7.46
Lenovo Moto Tab	**Lenovo**	Snapdragon 625	2	8	2	933	26.5	7.5
Vankyo MatrixPad Z1	**Vankyo**	N/A	1.5	4	1	933	N/A	N/A

Table 1: Device specifications. Chipset, peak performance, and memory bandwidth are not available for Vankyo.

Index	Computation Design			Communication Design		
	T_m	T_n	$T_m(d)$	I_p	O_p	W_p
1	160	12	576	11	8	13
2	160	12	576	5	5	22
3	160	12	576	12	13	17
4	160	12	576	10	10	10
5	130	12	832	10	10	10
6	100	16	832	10	10	10
7	220	8	704	10	10	10
8	100	16	832	6	14	10
9	100	18	704	10	10	10

Table 2: Nine FPGA specifications on Xilinx ZCU 102 board.



(a) CDF of SRCC (b) Estimated total search cost

Figure 13: (a) CDF of SRCC values of DNN models on 195 desktops [1]. The annotation "high/mid/low" represents the highest/middle/lowest 33.3% of the devices. (b) Estimated total search cost.

SRCC values of latency rankings based on the DNN models listed in the dataset. We see that latency monotonicity is very well preserved at scale. For example, among the high-end desktops, SRCC values among 50+% of *any* desktop pairs are higher than 0.95 (a strong ranking correlation). While the AI-Benchmark dataset is built for orthogonal purposes, the SRCC values show good latency monotonicity for desktops at scale.

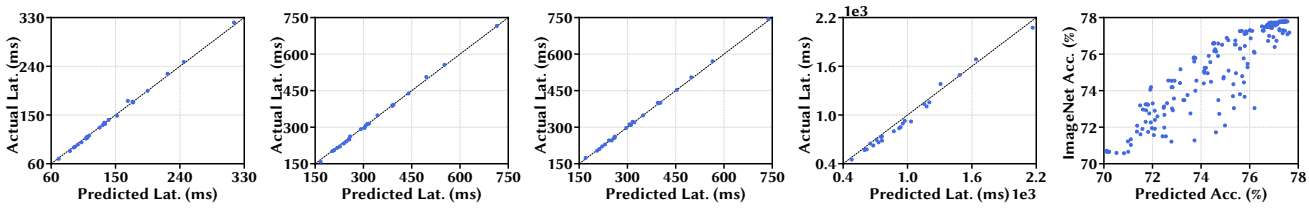
Finally, we examine the total search cost of using MonoNAS for desktops at scale and use the estimated SRCC values for the 195 desktops based on AI-Benchmark [1]. Specifically, we greedily find the proxy devices that have the best latency monotonicity with other devices, and show the number of target devices that can meet the SRCC threshold. We see from Fig. 13(b) that with about 5 proxy devices, the vast majority of target devices can be covered (even with a SRCC threshold of 0.9). In other words, to find Pareto-optimal architectures for 195 desktops, the existing device-aware NAS has a total search cost of $\mathcal{O}(195)$, while MonoNAS has a total search cost of $\mathcal{O}(5)$, yet almost without losing model performance.

C. Latency and Accuracy Predictors

We now show how well the predicted latency and accuracy perform for the MobileNet-based search space.

Latency Predictor. We build a latency lookup table with latencies of various layers and operators for each proxy device. For each sample model, we profile the average latency of 1000 runs. We use a single thread for running the TensorFlow Lite interpreter by default. To show the accuracy of our latency predictors (i.e., latency lookup tables), we sample a few CNN models and measure their actual latency on our four devices listed in Table 1. The comparison between actual and predicted latency is shown in Fig. 14(a), with a root mean squared error of 2.88ms on S5e, 4.69ms on TabA, 3.72ms on Lenovo, and 59.18ms on Vankyo. The absolute prediction error is the largest on Vankyo, because it has the largest absolute latencies due to its low-end hardware specification. As corroborated by prior studies [13, 32, 36], our measurement shows that the predicted average latency is almost identical to the actual value.

Accuracy Predictor. Our accuracy predictor is a neural network with four fully-connected layers and updated with 176 samples on top of the predictor used in [10]. The accuracy predictor takes a 128-dimensional feature vector (which is converted from a 21-dimensional architecture configuration within the search space) as input. Fig. 14(e) compares the actual and predicted accuracies, which have a SRCC of 0.903 and root mean squared error of 1.11%. The performance of our accuracy predictor is in line with the existing NAS literature for MobileNet-based models: e.g., Fig. 15 in [10] shows the root mean squared error of accuracy prediction error can be 15+% and the SRCC of predicted and actual accuracies is $\sim$ 0.9. Note that the SRCC of the predicted vs. actual accuracy is more important than the root mean squared error (e.g., an accuracy predictor whose estimate is always 10% higher than the true accuracy has no impact on the NAS process).



(a) S5e (b) TabA (c) Lenovo (d) Vankyo (e) Accuracy Predictor

Figure 14: (a)(b)(c)(d) Measured average inference latency versus predicted latency based on latency lookup tables. The root mean squared errors for S5e, TabA, Lenovo, and Vankyo are 2.88ms, 4.69ms, 3.72ms, and 59.18ms respectively. (e) Actual vs. predicted accuracy. The root mean squared error is 1.11%, and SRCC is 0.903.

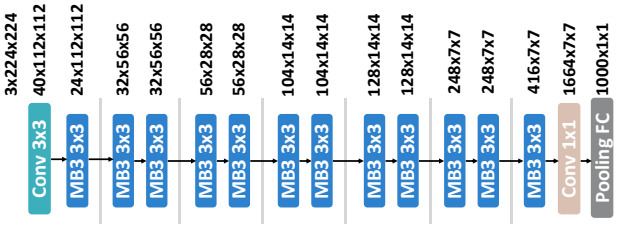


Figure 15: An example architecture in our search space based on MobileNet-V2, which achieves 71ms average inference latency on S5e.

D. Additional Experiment Results

We consider the same setup as considered in Section 6 and show additional experiment results. Specifically, when TabA, Lenovo, and Vankyo are target devices, the corresponding experiment results are shown in Fig. 16, Fig. 17, and Fig. 18, respectively. An example of searched architecture is shown in Fig. 15. Our results demonstrate the effectiveness and efficiency of MonoNAS, which exploits latency monotonicity for efficiently finding Pareto-optimal architectures on new devices without going through the lengthy device-aware NAS.

Finally, we summarize the scaling complexity, latency, and accuracy of different methods in Table 3. We also deploy our backbone network MobileNet-V2 with multiplier 1.3 here, because its latency on our devices are closest to the listed latency constraints with a relatively high accuracy.

E. Results on ResNet-50

Up to this point, all our results have been based on the MobileNet-V2 backbone network, where different layers are executed sequentially. We now turn to a different type of architecture — ResNet-50 — where the execution of layers is more complex.

Search Space. We define the search space as follows. The depth of each stage can be chosen from "0, 1, 2" (added to base depths "2, 2, 4, 2"), and the candidate expansion ratios of each block are "0.2, 0.25, 0.35". With a multiplier 0.65, the stage widths are "168, 336, 664, 1328" in our search space.

Method	Device d @ Lat. Const. $\bar{L}_d$	FLOPs (M)	Actual Lat. (ms)	Top-1 Acc. (%)	Top-5 Acc. (%)	Scaling Complexity
Predictor-S5e	S5e @115ms	446.29	115.88	75.68	92.70	$\mathcal{O}(n)$
MonoNAS-TabA		452.67	118.04	75.64	92.69	
MonoNAS-Lenovo		452.67	115.88	75.68	92.69	$\mathcal{O}(1)$
MonoNAS-Vankyo		451.30	117.46	75.73	92.71	
Scaled-TabA		506.72	126.64	75.33	92.43	
Scaled-Lenovo		473.00	122.70	75.43	92.49	$\mathcal{O}(1)$
Scaled-Vankyo		419.18	121.80	73.56	91.48	
MonoNAS-FLOPs		425.88	114.90	75.73	92.71	$\mathcal{O}(1)$
MobileNet-V2 (1.3x)		509	121	74.4	92.1	$\mathcal{O}(1)$
Predictor-TabA	TabA @260ms	436.16	260.21	75.53	92.60	$\mathcal{O}(n)$
MonoNAS-S5e		439.03	260.57	75.53	92.66	
MonoNAS-Lenovo		436.16	260.57	75.47	92.62	$\mathcal{O}(1)$
MonoNAS-Vankyo		434.21	258.79	75.44	92.64	
Scaled-S5e		481.84	297.90	74.71	92.08	
Scaled-Lenovo		473.0	277.50	75.43	92.49	$\mathcal{O}(1)$
Scaled-Vankyo		472.68	279.20	74.92	92.24	
MonoNAS-FLOPs		425.88	261.85	75.73	92.71	$\mathcal{O}(1)$
MobileNet-V2 (1.3x)		509	271	74.4	92.1	$\mathcal{O}(1)$
Predictor-Lenovo	Lenovo @267ms	441.66	268.08	75.61	92.64	$\mathcal{O}(n)$
MonoNAS-S5e		440.97	267.22	75.47	92.64	
MonoNAS-TabA		441.66	268.08	75.54	92.63	$\mathcal{O}(1)$
MonoNAS-Vankyo		436.17	266.78	75.58	92.65	
Scaled-S5e		481.84	302.7	74.71	92.08	
Scaled-TabA		506.72	292.60	75.33	92.43	$\mathcal{O}(1)$
Scaled-Vankyo		472.68	283.70	74.92	92.24	
MonoNAS-FLOPs		425.88	266.68	75.73	92.71	$\mathcal{O}(1)$
MobileNet-V2 (1.3x)		509	279	74.4	92.1	$\mathcal{O}(1)$
Predictor-Vankyo	Vankyo @718ms	376.66	714.3	74.81	92.23	$\mathcal{O}(n)$
MonoNAS-S5e		374.61	735.58	74.77	92.14	
MonoNAS-TabA		374.61	735.58	74.73	92.15	$\mathcal{O}(1)$
MonoNAS-Lenovo		374.61	735.58	74.72	92.17	
Scaled-S5e		403.72	722.60	74.81	92.03	
Scaled-TabA		417.41	754.80	75.13	92.22	$\mathcal{O}(1)$
Scaled-Lenovo		433.16	771.20	74.93	92.08	
MonoNAS-FLOPs		306.17	630.46	73.43	91.32	$\mathcal{O}(1)$
MobileNet-V2 (1.3x)		509	768	74.4	92.1	$\mathcal{O}(1)$

Table 3: Summary of various scaling methods.

Latency Monotonicity. Unlike for the MobileNet-V2-based search space, it involves more engineering efforts to build a device-specific latency predictor for our search space based on ResNet-50. As such, the latency lookup tables or predictors in [10, 13, 36] are only built for MobileNet-based search space. Here, we empirically measure the average latencies of 30 randomly selected models and calculate SRCC values. Note that, in Fig. 6 in Section 5, we see that 30 models are enough to estimate SRCC values with a low variance. For the ResNet-based search space, the latency monotonicity results are shown in Fig. 19(a). We see that latency monotonicity is still very well preserved,

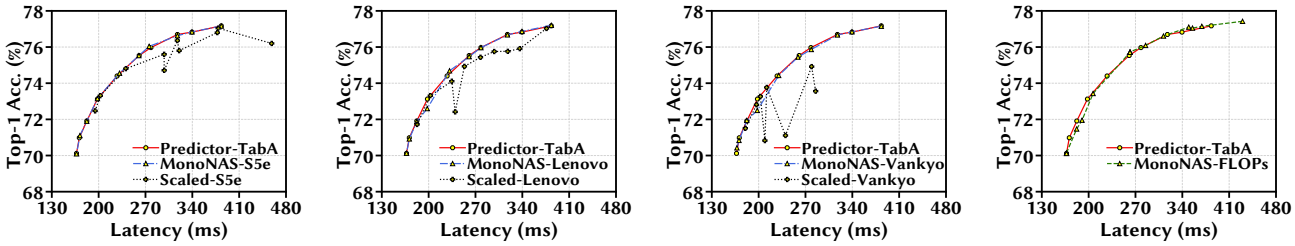


Figure 16: Results on the target device TabA, using S5e, Lenovo, and Vankyo as proxy devices, respectively. "Predictor-TabA" is the baseline #1, "MonoNAS-x" means using MonoNAS with x as the proxy device, and "Scaled-x" means heuristic scaling applied to x's Pareto-optimal architecture.

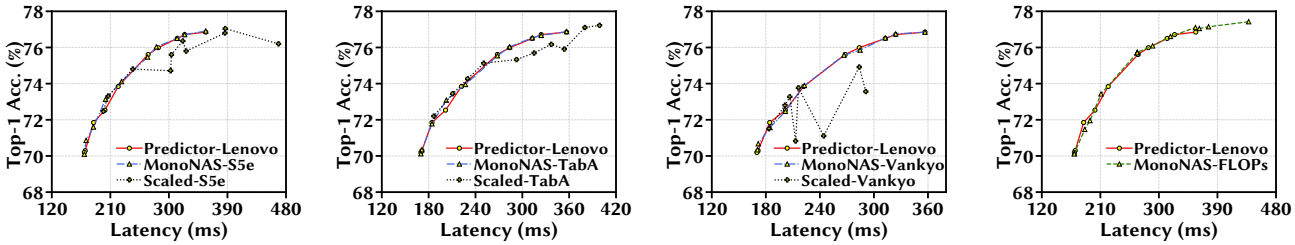


Figure 17: Results on the target device Lenovo, using S5e, TabA, and Vankyo as proxy devices, respectively. "Predictor-Lenovo" is the baseline #1, "MonoNAS-x" means using MonoNAS with x as the proxy device, and "Scaled-x" means heuristic scaling applied to x's Pareto-optimal architecture.

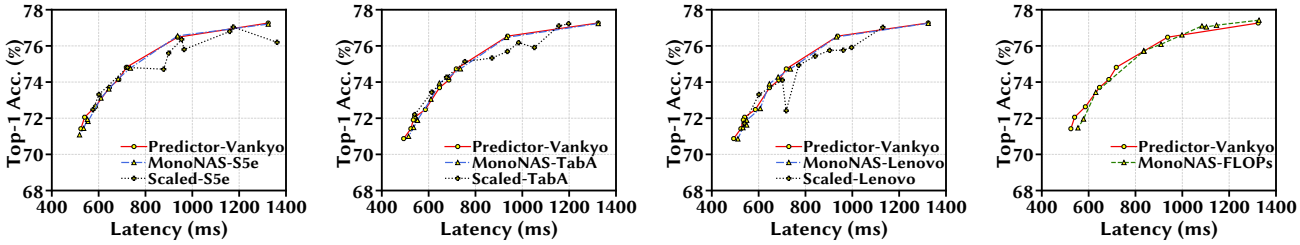
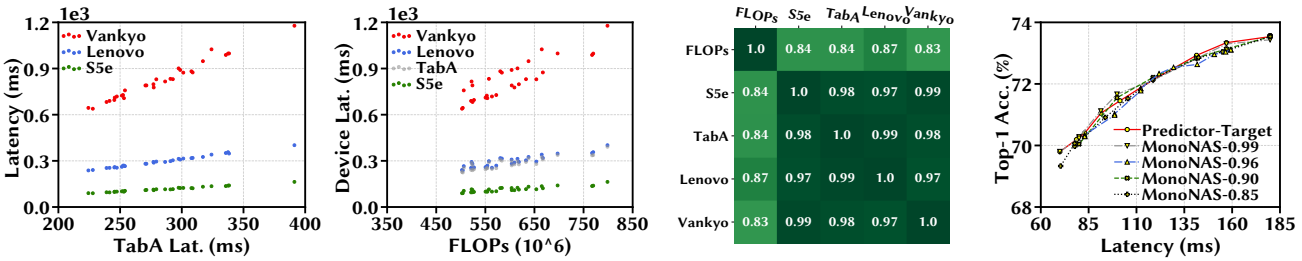


Figure 18: Results on the target device Vankyo, using S5e, TabA, and Lenovo as proxy devices, respectively. "Predictor-Vankyo" is the baseline #1, "MonoNAS-x" means using MonoNAS with x as the proxy device, and "Scaled-x" means heuristic scaling applied to x's Pareto-optimal architecture.

with SRCC close to 1.0. We also show the latency results vs. FLOPs in Fig. 19(b), and see that the monotonicity between the actual latency and FLOPs is not good, which is also reflected in the SRCC values shown in Fig. 19(c). Specifically, unlike in the MobileNet-based search space where SRCC of actual latency and model FLOPs is about 0.9, the corresponding SRCC in the ResNet-based search space is only around 0.8+.

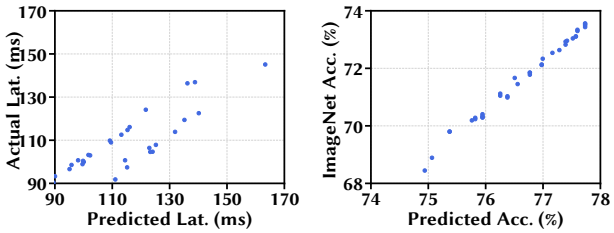
To summarize, in the ResNet-based search space, latency monotonicity still exists for mobile devices, and hence MonoNAS still applies. Nonetheless, FLOPs may not serve as a good proxy latency indicator (i.e., the FLOPs may not be included in the proxy device pool in Fig. 5).

Experiments on Synthetic Devices. We now perform experiments for the new search space based on ResNet-50. We use the same methodology as described in Section 6. Specifically, we consider an evolutionary search with the same settings. The ResNet-50 super net and accuracy predictor is obtained from an open-source project on GitHub (`https://github.com/mit-han-lab/once-for-all`). We evaluate models on the ImageNet validation dataset [14], with input image size 128. The accuracy predictor is a neural network with four fully-connected layers, taking a 82-dimensional feature vector (which is



(a) Latency Monotonicity (b) Latency vs. FLOPs (c) SRCC (d) Searched architectures

Figure 19: Results on ResNet-50. (a)(b) Measured latencies of 30 random models on our four mobile devices, and their FLOPs. (c) SRCC on different pairs of devices. (d) Searched architectures for synthetic devices. "Predictor-Target" means the result for the synthetic target device using its own (perfect) latency predictor, and "MonoNAS-x" means the results by using MonoNAS on different synthetic proxy devices with SRCC values of x.



(a) Latency (b) Accuracy

Figure 20: Performance predictors for architectures based on ResNet-50. (a) Measured vs. predicted average inference latency for S5e. The root mean squared error is 11.21ms, and SRCC is 0.81. (b) Actual vs. predicted accuracy. The root mean squared error is 5.03%, and SRCC is 0.99.

converted from a 30-dimensional architecture configuration within the search space) as input. We also build the latency predictor following the same approach used for the MobileNet-based search space.

We show the performance of latency and accuracy predictors in Fig. 20. It can be seen that the accuracy predictor performs extremely well with SRCC of 0.99. Nonetheless, unlike in the MobileNet case (results shown in Figs. 14(a)–14(d)), the latency predictor performs very poorly with low SRCC of only 0.81, due in part to the more complex layer execution in ResNet. This means that the current latency predictor may not be used by the subroutine device-aware NAS algorithm on proxy devices. Note that MonoNAS can use any subroutine to find Pareto-optimal architectures on proxy devices (like the existing heuristic scaling methods [12, 30]), and is also supported by Theorem 1. Thus, to highlight the practical benefits of MonoNAS, it suffices to show MonoNAS can quickly find optimal architectures for a new target device when latency monotonicity is reasonably well satisfied with a good SRCC value. Whether or not the proxy device is a real one makes no difference (e.g., in the MobileNet-based search space, MonoNAS can use FLOPs as the latency indicator/predictor for a virtual proxy device, because FLOPs and measured latencies on real mobile devices have a good ranking correlation). Thus, although our built latency predictor does not perform well as shown in Fig. 20(a), we can simply consider the predicted latency as the real latency for a virtual synthetic target device. That is, the synthetic target device is assumed to have the same latency as our predicted latency. This can still highlight the essence of MonoNAS (i.e., exploiting latency monotonicity), while avoiding the actual lengthy latency measurement for each candidate architecture during NAS as used in [29]. Next, to serve the synthetic target device, we create synthetic proxy devices whose latencies are noisy versions of the predicted latencies used by the synthetic target device, with SRCC values of 0.99, 0.96, 0.90, and 0.85, respectively.

We show the results in Fig. 19(d) and see that when the SRCC value between the synthetic target device and synthetic proxy device is 0.99, MonoNAS works very well, yielding almost the same set of Pareto-optimal models as running device-aware NAS again on the target device. Even when the SRCC decreases to 0.85, the accuracy gap between models found by MonoNAS and those found by running device-aware NAS is still not noticeable in most latency regions, although it is (slightly) greater than the gap when using a proxy device with SRCC of 0.99. This can be explained by noting that the accuracy predictor for the ResNet-based search space is extremely good (SRCC of 0.99). As a result, the imperfection of latency monotonicity also becomes more critical. On the other hand, if the accuracy predictor itself is less perfect, there is more headroom for the imperfection of latency monotonicity between the target device and proxy device.

To summarize, when the accuracy predictor can almost perfectly predict the actual accuracy ranking (or alternatively, the accuracy is measured), MonoNAS can still work

well in the presence of approximate latency monotonicity (e.g., SRCC 0.9+) and yield almost the same set of Pareto-optimal architectures as running device-aware NAS on each target device. In general, when the other aspects of the sub-routine NAS algorithm used by proxy devices become more perfect, the requirement on latency monotonicity between target device and proxy device also becomes more stringent (i.e., setting a higher SRCC threshold in Algorithm 1).

F. Roofline Analysis

We offer roofline analysis to further explain latency monotonicity. Roofline analysis is a methodology for visual representation of hardware platform's *peak* performance as a function of the operational intensity, which identifies the bottleneck of the system [33].

Fig. 21(a) shows the theoretical roofline model of two mobile devices (Samsung Galaxy S5e and TabA) plotted according to their reported hardware specification listed in Table. 1. When operational intensity is low (linear slope region), memory bandwidth is the limiting factor for program speed (i.e., memory-bound); when operational intensity is high (horizontal region), peak FLOPs rate becomes the bottleneck (i.e., compute-bound).

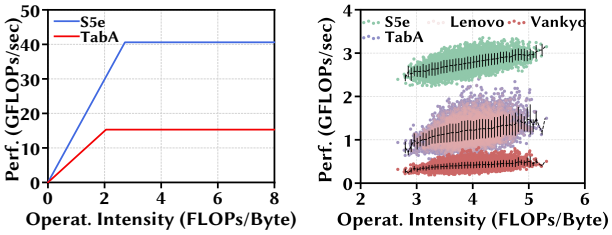
Suppose that we have two devices $\mathbf{d}_1$ and $\mathbf{d}_2$ with memory bandwidths $B_{\mathbf{d}_1}$ and $B_{\mathbf{d}_1}$, respectively, and two CNN models of architectures $\mathbf{x}_1$ and $\mathbf{x}_2$ with operational intensities $OI_{\mathbf{x}_1}$ and $OI_{\mathbf{x}_2}$, respectively. Next, we show that latency monotonicity is guaranteed to hold for two devices if CNN models are either memory-bound or compute-bound on both devices.

Memory-bound. In the memory-bound region, the slope in the roofline model of a device is the bandwidth, and the resulting performance is the bandwidth multiplied by the program's operational intensity. Assuming that $\mathbf{x}_1$ is slower than $\mathbf{x}_2$ on device $\mathbf{d}_1$ without loss of generality, we have $\frac{FLOP_{\mathbf{x}_1}}{OI_{\mathbf{x}_1} \cdot B_{\mathbf{d}_1}} > \frac{FLOP_{\mathbf{x}_2}}{OI_{\mathbf{x}_2} \cdot B_{\mathbf{d}_1}}$. Then, by multiplying both sides by $\frac{B_{\mathbf{d}_1}}{B_{\mathbf{d}_2}}$, we obtain $\frac{FLOP_{\mathbf{x}_1}}{OI_{\mathbf{x}_1} \cdot B_{\mathbf{d}_2}} > \frac{FLOP_{\mathbf{x}_2}}{OI_{\mathbf{x}_2} \cdot B_{\mathbf{d}_2}}$, i.e., $\mathbf{x}_1$ is also slower than $\mathbf{x}_2$ on device $\mathbf{d}_2$. Thus, latency monotonicity holds for the two devices $\mathbf{d}_1$ and $\mathbf{d}_2$.

Compute-bound. Likewise, if CNN models fall into the compute-bound region for two devices, then we can also establish latency monotonicity using a similar logic.

For search spaces with models that span across both memory-bound and compute-bound regions, the roofline analysis cannot guarantee strict latency monotonicity. In addition, the roofline analysis only provides a sufficient condition for latency monotonicity under the assumption that devices run at their peak performances (in terms of FLOPs/sec). Thus, we experimentally show the actual performance of CNN models on our four mobile devices shown in Table 1.

We measure the actual attainable peak performance of

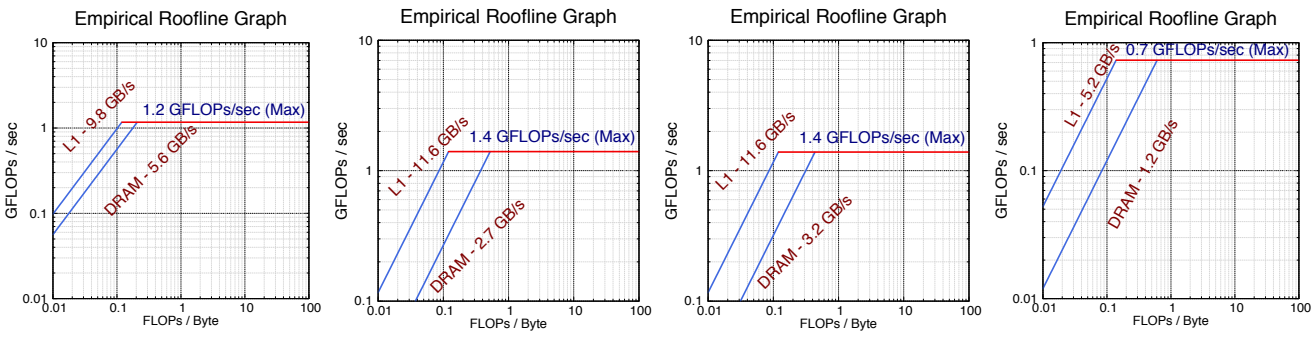


(a) Theoretical roofline model (b) Sampled models

Figure 21: (a) Theoretical roofline model is plotted according to hardware specification of S5e and TabA. (b) Black vertical lines denote the standard deviation of data within each bin, with the center denoting the average.

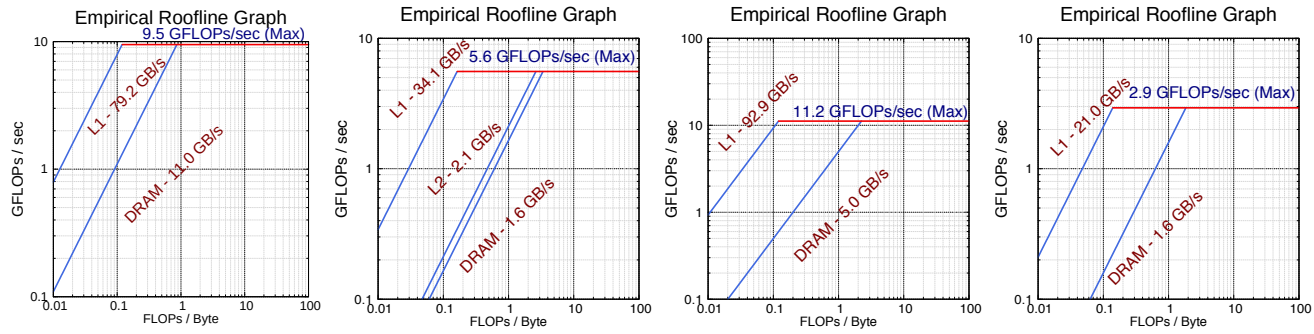
our four devices with the tool in [17], a roofline model specially for mobile SoCs. Our results show that the sampled CNN models (with 2.6 to 5.4 FLOPs/Byte) are all in the *compute*-bound region for the devices. We randomly sample 10000 models from the MobileNet-V2 [26] backbone with variable depths of each stage in "2, 3, 4", variable filter size of each convolutional layer in "3, 5, 7", and variable expansion ratio of each block in "3, 4, 6" (more details in Section 6). The empirical roofline results are shown in Fig. 21(b). The operational intensity of the sampled models ranges from 2.6 to 5.4 FLOPs/Byte, while the devices' actual performances are much lower than their peaks and vary for different models. Additionally, similar empirical models utilizing all the cores (4 or 8 in this work) are measured in Fig. 23, which shows that the ridge operational intensity of S5e, Lenovo, and Vankyo are less than or around 2 FLOPs/Byte, while TabA has a threshold of 3 FLOPs/Byte. Thus, in this case, most of our sampled models reside in the compute-bound region of these devices, except for those with operational intensity less than 3 FLOPs/Byte on TabA.

To summarize, although perfect latency monotonicity (i.e., SRCC=1.0) is difficult in practice, a very good latency monotonicity with SRCC values of higher than 0.95 or even 0.99 can be observed within the mobile platform, as confirmed by our own empirical measurement, third-party latency predictor and AI-Benchmark dataset. In practice, considering imperfection in other aspects of NAS (e.g., accuracy predictor, randomness in the search process, and/or non-convexity of NAS), this high degree of latency monotonicity is enough to for MonoNAS to efficiently scale up the NAS process for diverse mobile devices, yet almost without losing Pareto optimality.



(a) S5e (b) TabA (c) Lenovo (d) Vankyo

Figure 22: Empirical roofline models of devices in Table 1 measured with Gables [17].



(a) S5e (b) TabA (c) Lenovo (d) Vankyo

Figure 23: Empirical roofline models of devices in Table 1 measured with Gables [17]. All the device cores are utilized.