# SCAN-EDGE: FINDING MOBILENET-SPEED HYBRID NETWORKS FOR COMMODITY EDGE DEVICES

**Hung-Yueh Chiang & Diana Marculescu**
Chandra Family Department of Electrical and Computer Engineering
The University of Texas at Austin
2501 Speedway, Austin, TX 78712, USA
{hungyueh.chiang, dianam}@utexas.edu

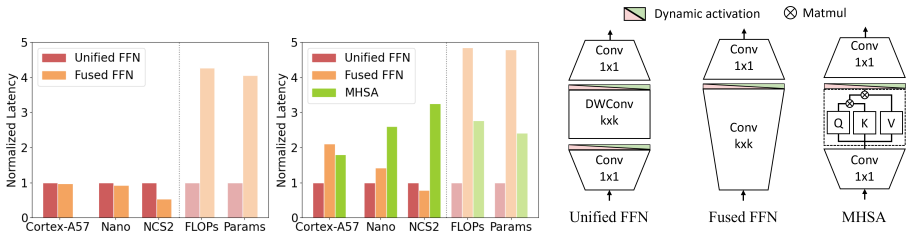
## ABSTRACT

Designing low-latency and high-efficiency hybrid networks for diverse low-cost commodity edge devices is costly and tedious, thereby leading to the use of neural architecture search (NAS) for finding optimal architectures. However, the challenges of unifying NAS for a wide range of edge devices lay in the sheer number of hardware designs, supported operations, and compilation optimizations. Existing methods fix the search space of architecture choices (*e.g.,* activation, convolution, or self-attention) for network stages and estimate the latency with hardware-agnostic proxies (*e.g.,* FLOPs), which fail to achieve proclaimed latency on a wide variety of edge devices. We address the issue and propose a unified NAS framework, termed **SCAN-Edge**, which jointly searches **S**elf-attention, **C**onvolution, and **A**ctivatio**N** to best accommodate the diversity of **Edge** devices, such as CPU-, GPU-, and hardware accelerator-based. During the search, **SCAN-Edge** accurately estimates the end-to-end latency with pre-built calibrated latency lookup tables and addresses the resulting large search space with a hardware-aware evolutionary algorithm, which accelerates the sampling process. Experiments on large-scale datasets show that, compared with prior art, our hybrid networks match *actual MobileNetV2 latency* for $224 \times 224$ input resolution on various commodity edge devices.

## 1 INTRODUCTION

Automatically designing deep learning architectures for specific hardware has been an appealing research topic and has achieved huge success in convolutional neural networks (CNNs) Yu et al. (2020); Cai et al. (2020); Stamoulis et al. (2019). Recent approaches Gong & Wang (2022); Tang et al. (2023) extend previous methods that train a one-shot supernet where they search subnets for different devices to hybrid networks (convolution and transformer). However, the aforementioned methods fix the search space of architecture choices (*e.g.,* GELU activation, depth-wise convolution, or self-attention layers) for network stages, which can be sub-optimal for a certain edge device, since the optimal search space largely depends on the hardware implementation and compiler optimization. For example, a memory-bound operator like depthwise convolution is not necessarily efficient nor optimal for a highly parallelized device Lu et al. (2021); Zhang et al. (2020). Moreover, prior-art Gong & Wang (2022) approximates model complexity with zero-cost proxies White et al. (2023) such as floating-point operations (FLOPs) and number of parameters, which do not reflect the *actual latency* on target edge devices and fail to generalize to a wide range of edge devices. To illustrate this argument, in Figure 1 (and in Appendix A), we show various edge devices favor different operations when actual hardware metrics are considered. As shown in Figure 1, the actual latency greatly depends on the input feature map size and on the hardware platform. Although unified feedforward networks (unified FFNs) learn spatial relationships in the feature maps by a depthwise convolution with fewer parameters and fewer theoretical FLOPs, they are highly memory-bound with low arithmetic intensity. Despite having more FLOPs and parameters, fused feedforward networks (fused FFNs) that replace the expansion and depthwise convolutions with a vanilla convolution have lower latency than MHSA layers on Nano and NCS2.

To address this issue, we propose a unified NAS framework **SCAN-Edge** that relies on a weight-sharing supernet for searching hybrid networks targeted for running on edge devices. Our search




**(a)** The first stage **(b)** The last stage **(c)** Block candidates

**Figure 1:** We profile the latency and zero-cost proxies of EfficientFormerV2 S0 on different devices. (a) shows the latency of the first stage (FFNs only) with input size (h, w, c)=(56, 56, 28). While unified FFN has fewer FLOPs, it is bounded by memory and has a similar latency to fused FFN. (b) shows the latency of the last stage (FFNs and MHSAs) with input size (7, 7, 176). The stage latency is device-dependent and highly different from the proxies. (c) Our supernet consists of three different blocks with dynamic activation layers. Each MHSA block is followed by either a unified FFN or a fused FFN. The components, *e.g.,* residual connections, are simplified to avoid cluttering the figure.

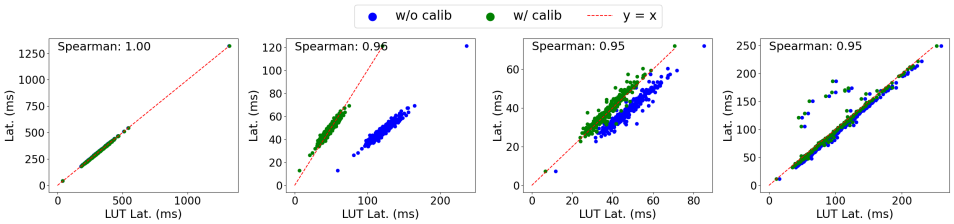
space includes unified feedforward networks (unified FFNs, memory-bound), fused feedforward networks (fused FFNs, compute-bound), and multi-head self-attention layers (MHSAs), as well as two activation functions, GELU and ReLU, to best support a wide variety of the commodity devices. During the search, we incorporate calibrated latency lookup tables (LUTs) profiled on target devices and a learning-based accuracy predictor. To search for the optimal subnet from this huge search space, our search algorithm optimizes the search space based on the hardware and the search space quality. Our experiments show that hybrid networks found by our framework match the *actual MobileNetV2 latency* while *providing better accuracy* on a wide variety of edge devices when compared with prior approaches.

## 2 One-shot Supernet

**Dual feedforward network.** We design our supernet with dual FFN to best accommodate various edge devices. As shown in Figure 1, in each FFN block, we provide two choices for searching: unified FFN and fused FFN with different kernel sizes (*e.g.,* 3, 5) and expansion ratios (*e.g.,* 2, 3, 4). We use dual feedforward networks that are composed of two sets of separate weight matrices. Only one set of weight matrices will be activated (Unified or Fused) in a block during training (*cf.* in Appendix B). During training, we sample $N_{\text{ffn}}$ blocks for every stage $S_i$ and randomly switch between unified FFN and fused FFN. For each FFN block $j$ in stage $i$, we sample the kernel size $K_i^j$, and expansion ratio $E_i^j$. In the search stage, we search the number of FFNs $N_{\text{ffn}}$ for every stage $S_i$, and the kernel size $K_i^j$, as well as the expansion ratios $E_i^j$. The full search space is shown in Table 5. We list the detailed training parameters in the Appendix G.

**Searching for multi-head self-attention.** We follow the settings in Chen et al. (2021a) to design our MHSA with weight entanglement. We allow the $V$ matrices in each MHSA to have larger dimensions similar to Graham et al. (2021), Tang et al. (2023). Specifically, we search the expansion ratios $E_i^j$ for the value matrix in every MHSA block, such that $\dim(V_i^j) = N_{\text{head}} \times \dim(\text{Q-K-V}) \times E_i^j$, where $N_{\text{head}}$ and $\dim(\text{Q-K-V})$ are fixed. The dimension of the $Q$ and $K$ matrices are fixed to $N_{\text{head}} \times \dim(\text{Q-K-V})$ so that the attention matrices $A = QK^\intercal$ are shared with all subnets. During subnet search, the algorithm finds the optimal subnet by searching *the number* and *the position* of MHSA $N_{\text{mhsa}}$ and deciding their expansion ratios $E_i^j$ for the $V$ matrices in the last two stages (*i.e.,* $i \in \{3, 4\}$).

**Dynamic activation layers.** The advanced activation layers such as GELU Hendrycks & Gimpel (2016) are not well-supported by edge devices and their compilers, thereby incurring a latency overhead during inference. We designed our supernet to support ReLU Agarap (2018), one of the basic activation layers that are friendly to hardware implementations and compiler optimizations. During training, we randomly switch between two activation layers for FFN and MHSA. Our search algorithm searches for the best activation combinations to optimize latency and accuracy.

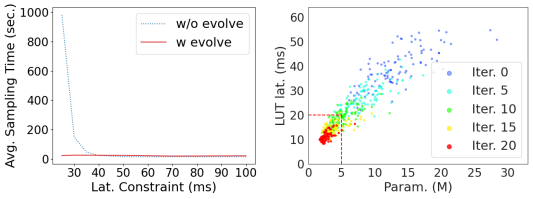



(a) Cortex-A57 (b) Intel NCS2 (c) Nano TensorRT (d) Nano CUDA

**Figure 2:** Naive latency estimations from block-wised latency lookup tables (LUTs) tend to be overestimated (blue). We additionally profile 10 end-to-end subnet latencies to calibrate the LUTs by linear regression. We show the high quality of the calibrated latency estimations that fit $y = x$ closely (green).

## 3 Searching Subnets for Edge Devices

**Search Objective.** Given a supernet architecture $\mathcal{A} = \{\alpha_1, ..., \alpha_n\}$ and its trained weight $\mathcal{W} = \{w_1, ..., w_n\}$, we denote the supernet as $f(\mathcal{A}, \mathcal{W})$ and a sampled subnet as $f(\alpha_i, w_i)$, where the architecture $\alpha_i \in \mathcal{A}$ and the weights $w_i \in \mathcal{W}$ are sampled from the supernet. Our search objective is to find an optimal architecture $\alpha^*$ and $w^*$ such that $\alpha^*, w^* = \arg\max_{\alpha \in \mathcal{A}, w \in \mathcal{W}} \text{Acc}(f(\alpha, w))$ that maximizes the accuracy while satisfying a set of constraints $\zeta_i(\alpha, w) < c_i$, $i = 1, 2, ..., n$ on a given device. $\zeta_i$ is the predictor function for the latency or the memory footprint of the subnet. Since evaluating the accuracy and the latency of thousands of subnets in the search process is not practical, we train a neural network $\chi$ to predict the accuracy and use lookup tables to estimate the real latency during the search.




**(a)** Sampling time **(b)** Search space evolution

**Figure 3:** (a) The subnet sampling time increases exponentially as the latency constraint is reduced. (b) The search space evolves from blue to red dots, where it meets the constraints: 5 M parameters and 20 ms latency.

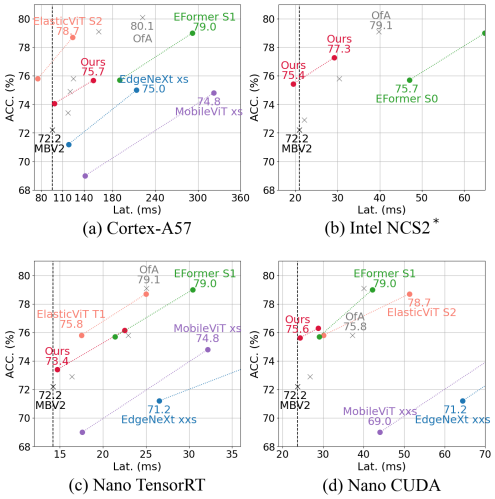
**Latency Lookup Table with Calibration.** We build latency lookup tables (LUTs) for every device and profile all possible blocks in every stage on the device. We additionally profile the end-to-end latency of 10 subnets to calibrate the over-estimated latency with linear regression, as shown in Figure 2. During the search process, we estimate the subnet latency by summing the latency of every block in the lookup table, such that $c_{\tilde{\text{lat.}}} = \zeta_{\text{lat.}}(\alpha_i, w_i) = \kappa \Sigma_{j=1}^{n} \text{LUT}(\alpha_i^j) + \epsilon$. $\kappa$ and $\epsilon$ are the parameters obtained from the linear regression algorithm.

**Search space quality estimation.** The sampling time increases exponentially when searching subnets with strict constraints, as shown by the blue line in Figure 3 (a). To address the issue, inspired by Chen et al. (2021b), we update the search space during the search. The search space evolves if the quality of the search space $\mathcal{Q}(\mathcal{A}_t^*)$ defined by the current top $k$ subnets $\mathcal{A}_t^* = \{\alpha_1, ..., \alpha_k\}$ is better than the previous one, such that $\mathcal{Q}(\mathcal{A}_t^*) - \mathcal{Q}(\mathcal{A}_{t-1}^*) > \delta$. The quality of the search space is evaluated by the average predicted accuracy of the top $k$ subnets, *i.e.* $\mathcal{Q}(\mathcal{A}_t^*) = E_{\alpha \in \mathcal{A}_t^*}[\chi(\alpha)]$.

**Hardware-aware Search space evolution.** We update the sampling space by the moving average of the probability: $p_{t+1}(X_i^j = x) = \lambda p_t(X_i^j = x) + (1 - \lambda) p^*(X_i^j = x)$ where $\lambda \in [0, 1]$ is a scalar, $X_i^j$ is a random variable for a set of architecture choices of block $j$ in stage $i$, $x$ is a specific architecture, and $p_t(X_i^j = x)$ is the current probability of selecting architecture $x$ for the block. The $p^*$ is a probability defined as $p^*(X_i^j = x) = \Sigma_{s=1}^{k} \sigma_s(X_i^j = x)/k$ where $\sigma_s$ is an indicator function for subnet $s$ in top $k$ population. As shown in Figure 3 (b), by changing the sampling probabilities, the search space evolves from blue to red dots to where it meets the constraints, *i.e.,* red dots are mostly inside the red dashed rectangle.

## 4 MAIN RESULTS

We profile model latency on three different commodity hardware with their compilers using official benchmark tools: **(1) Edge CPU.** We get the model latency on ARM Quad-core Cortex-A57 (Cortex). Models are converted to ONNX onnx.ai format and run with the default compiler and execution provider in full precision (FP32). **(2) Edge GPU.** We obtain the latency on Jetson Jetson Nano 4G (Nano). Models are converted to ONNX format and compiled by the Cuda / TensorRT (TRT) in full precision (FP32). **(3) USB accelerator.** We get the latency on Intel Neural Compute Stick 2 (NCS2). Models are converted to OpenVINO IR (Intermediate Representation) and run with OpenVINO Intel in half precision (FP16). *All* models are compiled and profiled with batch size 1.




(a) Cortex-A57 (b) Intel NCS2* (c) Nano TensorRT (d) Nano CUDA

**Figure 4:** Our models achieve MobileNet speed among all hybrid counterparts across platforms while outperforming MobileNetV2 in accuracy. We also pivot Once-for-all (Conv only) in grey crosses for reference.

**Table 1:** Joint optimization latency and model size for Cortex-A57 with default compiler, Nano with TensorRT compiler, and NCS2 with OpenVINO compiler. The naming follows @{*lat*}ms@{*size*}M.

| Method | Acc. (%) | Size (M) | cortex / nano / ncs2 (ms) |
|---|---|---|---|
| MBV2 | 72.2 | 3.5 | 95.5 / 14.2 / 20.9 |
| EFormerV2 S0 | 75.7 | 3.6 | 190.3 / 21.4 / 47.0 |
| Cortex-A57 | | | |
| @150ms | 75.7 | 9.7 | 153.0 / - / - |
| @150ms@5M | 75.5 | 5.0 | **152.4** / - / - |
| Nano TensorRT | | | |
| @20ms | 76.1 | 13.5 | - / 22.5 / - |
| @20ms@5M | **75.9** | 5.0 | - / 22.2 / - |
| NCS2 OpenVINO | | | |
| @45ms | 78.6 | 47.9 | - / - / 47.5 |
| @45ms@5M | **75.7** | 5.0 | - / - / **43.0** |

### 4.1 IMAGE CLASSIFICATION WITH MOBILENET SPEED

In this experiment, we optimize the latency with $224 \times 224$ input resolution that is widely used not only in image classification but also in object detection and segmentation. We perform the search for each platform and compiler with our trained supernet, accuracy predictor, and pre-built latency tables. The details of searched architectures are listed in Appendix I. The weights of the searched subnets are inherited from the supernet and then fine-tuned on the ImageNet with 150 additional epochs. We test the subnet and report accuracy on the validation set. To get the latency, models are compiled with the compilers (*e.g.,* TensorRT and OpenVINO) and profiled with a batch size of 1 with $224 \times 224$ resolution input [†].

We show the results in Figure 4 and list the details of the comparison in Table 6. Once-for-all (OfA, Conv only) is also shown in Table 6 and Figure 4 for reference. Prior-art Mehta & Rastegari (2021); Pan et al. (2022); Maaz et al. (2022); Li et al. (2023) *fails to reach MobileNetV2 latency with on-par accuracy* on the three hardware platforms[*], although they have fewer FLOPs (EdgeNeXt) or smaller model size (MobileViTs, EdgeNeXt) than MobileNetV2. Although ElasticViT T1[‡] Tang et al. (2023) has the lowest latency on Cortex-A57, it uses a search space specific for edge CPU (*e.g.,* MobilenetV2 and V3 block) and thereby fails to reach MobilenetV2 on other computation platforms. Our hybrid models outperform MobileNetV2 in accuracy (**74%** *vs.* **72.2%** on Cortex-A57) while maintaining the same level of inference latency (**98.6 ms** *vs.* **95.5 ms** on Cortex-A57).

[†] MobileViTs Mehta & Rastegari (2021), EdgeNeXt Maaz et al. (2022) are trained and tested with $256 \times 256$ according to their original implementation

[*] MobileViTs Mehta & Rastegari (2021), EdgeNeXt Maaz et al. (2022), EdgeViT Pan et al. (2022), ElasticViT Tang et al. (2023) fail to compile on Intel NCS2 due to unsupported operators.

[‡] We evaluate the accuracy and latency with $224 \times 224$ inputs. The original implementation is $128 \times 128$.

### 4.2 JOINT OPTIMIZATION OF LATENCY AND MODEL SIZE

We experiment with joint optimization on ARM Cortex-A57 and Nvidia Jetson Nano platforms by constraining the search algorithm with both latency and number of parameters. In the experiment, we use ONNX default compiler for Cortex-A57 and TensorRT for Jetson Nano. As shown in Figure 3 (b), our search space evolves to where it meets both constraints. The searched models follow the naming @{*lat*}ms@{*size*}M, where *lat* and *size* are the search constraints. As shown in Table 1, our framework searches optimal models with the given constraints for Cortex-A57 and Nano with little accuracy loss ($0.2\%$).

## 5 DOWNSTREAM TASKS VIA TRANSFER LEARNING

We perform transfer learning from the ImageNet pre-trained weight to various downstream tasks: CIFAR10, CIFAR100 Krizhevsky et al. (2009), Food Bossard et al. (2014), and Pets Parkhi et al. (2012). All models are trained for 50 epochs with downstream datasets on an A500 GPU and set the batch size to 256 with a $10^{-3}$ base learning rate scaled by the batch size. The results are shown in Table 2. In general, our models outperform in accuracy their counterparts with similar latency in the downstream classification tasks on Cortex-A57. Moreover, our models match the MobileNetV2 latency on Cortex-A57 CPU (*cf.* Table 6).

**Table 2:** Our models have higher accuracy in the downstream classification tasks and better (lower) latency on Cortex-A57.

| Method | CF10 | CF100 | Food | Pets |
|---|---|---|---|---|
| MobileNetV2 | 91.6 | 72.1 | 71.2 | 83.2 |
| MobileViT XXS$^{\dagger}$ | 93.1 | 71.2 | 73.1 | 80.7 |
| EdgeNeXt XXS$^{\dagger}$ | 95.7 | 76.2 | 80.0 | 84.2 |
| cortex@95ms | 97.1 | 80.9 | **82.7** | 86.0 |
| cortex@150ms | **97.3** | **81.1** | 81.9 | **86.9** |

## 6 OBJECT DETECTION

We integrate our searched subnets as the backbone to SSDLite Sandler et al. (2018). We train the SSDLite with different backbones on COCO2017 dataset Lin et al. (2014) by using the MMDetection library OpenMMLab (b). We load the ImageNet pre-trained weights of each backbone, and train the detection models with a resolution of $320 \times 320$. The learning rates are tuned for each model. We deploy models on the Nvidia Jetson Nano 4G by using the MMDeploy library OpenMMLab (a) and profile the latency with Nvidia TensorRT profiling tools. As shown in Table 3, our model outperforms MobileViT XXS and EdgeNeXt XXS in both mAP and latency.

**Table 3:** The SSDLite with our searched model reaches the highest mAP and lowest end-to-end latency on Nano among all hybrid alternatives on COCO2017 object detection dataset.

| SSDLite Backbone | mAP | Lat. (ms) |
|---|---|---|
| MobileNetV2 | 20.5 | 54.4 |
| MobileViT XXS | 19.1 | 65.4 |
| EdgeNeXt XXS | 19.3 | 73.3 |
| Nano_trt@13ms (Ours) | 22 | 60.6 |

## 7 CONCLUSION

We propose a unified NAS framework that searches for hybrid networks with MobileNetV2-speed for low-cost commodity edge devices. Our framework incorporates different device-friendly operations for diverse edge devices with different hardware designs. To manage the large search space, we propose a hardware-aware evolutionary search to accelerate the process. Our experiments show our hybrid models match MobileNetV2-speed on Edge CPUs, Edge GPUs, and USB accelerators with better accuracy than MobileNetV2. Our framework is effective, generalizes to other vision tasks, and is applicable to various low-cost commodity devices.

ACKNOWLEDGEMENTS

This work was supported in part by NSF CCF Grant No. 2107085, iMAGiNE - the Intelligent Machine Engineering Consortium at UT Austin, and a UT Cockrell School of Engineering Doctoral Fellowship.

## REFERENCES

Abien Fred Agarap. Deep learning using rectified linear units (relu). *arXiv preprint arXiv:1803.08375*, 2018.

Berkin Akin, Suyog Gupta, Yun Long, Anton Spiridonov, Zhuo Wang, Marie White, Hao Xu, Ping Zhou, and Yanqi Zhou. Searching for efficient neural architectures for on-device ml on edge tpus. In *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition*, pp. 2667–2676, 2022.

Lukas Bossard, Matthieu Guillaumin, and Luc Van Gool. Food-101–mining discriminative components with random forests. In *Computer Vision–ECCV 2014: 13th European Conference, Zurich, Switzerland, September 6-12, 2014, Proceedings, Part VI 13*, pp. 446–461. Springer, 2014.

Han Cai, Chuang Gan, Tianzhe Wang, Zhekai Zhang, and Song Han. Once for all: Train one network and specialize it for efficient deployment. In *International Conference on Learning Representations*, 2020. URL https://arxiv.org/pdf/1908.09791.pdf.

Minghao Chen, Houwen Peng, Jianlong Fu, and Haibin Ling. Autoformer: Searching transformers for visual recognition. In *Proceedings of the IEEE/CVF international conference on computer vision*, pp. 12270–12280, 2021a.

Minghao Chen, Kan Wu, Bolin Ni, Houwen Peng, Bei Liu, Jianlong Fu, Hongyang Chao, and Haibin Ling. Searching the search space of vision transformer. *Advances in Neural Information Processing Systems*, 34:8714–8726, 2021b.

Tri Dao. FlashAttention-2: Faster attention with better parallelism and work partitioning. 2023.

Tri Dao, Daniel Y. Fu, Stefano Ermon, Atri Rudra, and Christopher Ré. FlashAttention: Fast and memory-efficient exact attention with IO-awareness. In *Advances in Neural Information Processing Systems*, 2022.

Jia Deng, Wei Dong, Richard Socher, Li-Jia Li, Kai Li, and Li Fei-Fei. Imagenet: A large-scale hierarchical image database. In *2009 IEEE conference on computer vision and pattern recognition*, pp. 248–255. Ieee, 2009.

Lukasz Dudziak, Thomas Chau, Mohamed Abdelfattah, Royson Lee, Hyeji Kim, and Nicholas Lane. Brp-nas: Prediction-based nas using gcns. *Advances in Neural Information Processing Systems*, 33:10480–10490, 2020.

Chengyue Gong and Dilin Wang. Nasvit: Neural architecture search for efficient vision transformers with gradient conflict-aware supernet training. *ICLR Proceedings 2022*, 2022.

Benjamin Graham, Alaaeldin El-Nouby, Hugo Touvron, Pierre Stock, Armand Joulin, Hervé Jégou, and Matthijs Douze. Levit: a vision transformer in convnet's clothing for faster inference. In *Proceedings of the IEEE/CVF international conference on computer vision*, pp. 12259–12269, 2021.

Dan Hendrycks and Kevin Gimpel. Gaussian error linear units (gelus). *arXiv preprint arXiv:1606.08415*, 2016.

Gao Huang, Yu Sun, Zhuang Liu, Daniel Sedra, and Kilian Q Weinberger. Deep networks with stochastic depth. In *Computer Vision–ECCV 2016: 14th European Conference, Amsterdam, The Netherlands, October 11–14, 2016, Proceedings, Part IV 14*, pp. 646–661. Springer, 2016.

Huggingface. pytorch-image-models. URL https://github.com/huggingface/pytorch-image-models/tree/main.

Intel. Openvino. URL https://github.com/openvinotoolkit/openvino.

Alex Krizhevsky, Geoffrey Hinton, et al. Learning multiple layers of features from tiny images. 2009.

Hayeon Lee, Sewoong Lee, Song Chong, and Sung Ju Hwang. Help: Hardware-adaptive efficient latency prediction for nas via meta-learning. *arXiv preprint arXiv:2106.08630*, 2021.

Yanyu Li, Ju Hu, Yang Wen, Georgios Evangelidis, Kamyar Salahi, Yanzhi Wang, Sergey Tulyakov, and Jian Ren. Rethinking vision transformers for mobilenet size and speed. In *Proceedings of the IEEE/CVF International Conference on Computer Vision*, pp. 16889–16900, 2023.

Tsung-Yi Lin, Michael Maire, Serge Belongie, James Hays, Pietro Perona, Deva Ramanan, Piotr Dollár, and C Lawrence Zitnick. Microsoft coco: Common objects in context. In *Computer Vision–ECCV 2014: 13th European Conference, Zurich, Switzerland, September 6-12, 2014, Proceedings, Part V 13*, pp. 740–755. Springer, 2014.

Ilya Loshchilov and Frank Hutter. Decoupled weight decay regularization. *ICLR*, 2019.

Gangzhao Lu, Weizhe Zhang, and Zheng Wang. Optimizing depthwise separable convolution operations on gpus. *IEEE Transactions on Parallel and Distributed Systems*, 33(1):70–87, 2021.

Muhammad Maaz, Abdelrahman Shaker, Hisham Cholakkal, Salman Khan, Syed Waqas Zamir, Rao Muhammad Anwer, and Fahad Shahbaz Khan. Edgenext: efficiently amalgamated cnn-transformer architecture for mobile vision applications. In *European Conference on Computer Vision*, pp. 3–20. Springer, 2022.

Sachin Mehta and Mohammad Rastegari. Mobilevit: light-weight, general-purpose, and mobile-friendly vision transformer. *arXiv preprint arXiv:2110.02178*, 2021.

onnx.ai. Onnx: Open neural network exchange. URL https://github.com/onnx/onnx.

OpenMMLab. Mmdeploy, a. URL https://github.com/open-mmlab/mmdeploy.

OpenMMLab. Mmdetection, b. URL https://github.com/open-mmlab/mmdetection.

Junting Pan, Adrian Bulat, Fuwen Tan, Xiatian Zhu, Lukasz Dudziak, Hongsheng Li, Georgios Tzimiropoulos, and Brais Martinez. Edgevits: Competing light-weight cnns on mobile devices with vision transformers. In *European Conference on Computer Vision*, pp. 294–311. Springer, 2022.

Omkar M Parkhi, Andrea Vedaldi, Andrew Zisserman, and CV Jawahar. Cats and dogs. In *2012 IEEE conference on computer vision and pattern recognition*, pp. 3498–3505. IEEE, 2012.

Adam Paszke, Sam Gross, Francisco Massa, Adam Lerer, James Bradbury, Gregory Chanan, Trevor Killeen, Zeming Lin, Natalia Gimelshein, Luca Antiga, et al. Pytorch: An imperative style, high-performance deep learning library. *Advances in neural information processing systems*, 32, 2019.

Ilija Radosavovic, Raj Prateek Kosaraju, Ross Girshick, Kaiming He, and Piotr Dollár. Designing network design spaces. In *Proceedings of the IEEE/CVF conference on computer vision and pattern recognition*, pp. 10428–10436, 2020.

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*, pp. 4510–4520, 2018.

Dimitrios Stamoulis, Ruizhou Ding, Di Wang, Dimitrios Lymberopoulos, Bodhi Priyantha, Jie Liu, and Diana Marculescu. Single-path nas: Designing hardware-efficient convnets in less than 4 hours. In *Joint European Conference on Machine Learning and Knowledge Discovery in Databases*, pp. 481–497. Springer, 2019.

Mingxing Tan and Quoc Le. Efficientnetv2: Smaller models and faster training. In *International conference on machine learning*, pp. 10096–10106. PMLR, 2021.

Chen Tang, Li Lyna Zhang, Huiqiang Jiang, Jiahang Xu, Ting Cao, Quanlu Zhang, Yuqing Yang, Zhi Wang, and Mao Yang. Elasticvit: Conflict-aware supernet training for deploying fast vision transformer on diverse mobile devices. *arXiv preprint arXiv:2303.09730*, 2023.

Hugo Touvron, Matthieu Cord, Matthijs Douze, Francisco Massa, Alexandre Sablayrolles, and Hervé Jégou. Training data-efficient image transformers & distillation through attention. In *International conference on machine learning*, pp. 10347–10357. PMLR, 2021.

Colin White, Mahmoud Safari, Rhea Sukthanker, Binxin Ru, Thomas Elsken, Arber Zela, Debadeepta Dey, and Frank Hutter. Neural architecture search: Insights from 1000 papers. *arXiv preprint arXiv:2301.08727*, 2023.

Jiahui Yu and Thomas S Huang. Universally slimmable networks and improved training techniques. In *Proceedings of the IEEE/CVF international conference on computer vision*, pp. 1803–1811, 2019.

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 *Computer Vision–ECCV 2020: 16th European Conference, Glasgow, UK, August 23–28, 2020, Proceedings, Part VII 16*, pp. 702–717. Springer, 2020.

Pengfei Zhang, Eric Lo, and Baotong Lu. High performance depthwise and pointwise convolutions on mobile devices. In *Proceedings of the AAAI Conference on Artificial Intelligence*, volume 34, pp. 6795–6802, 2020.

# A PRELIMINARY STUDY

In our preliminary study, we use EfficientFormerV2 S0 Li et al. (2023) as our base architecture and experiment with different FFNs, activations, and self-attention ratios (expansion of $V$ dimension). The feedforward networks (convolution only) have the same structure as MobilenetV2 blocks Sandler et al. (2018) except for using GELU activation and bias in convolution layers. We follow EfficientFormerV2 and use the term, feedforward network, for convolution layers.

**Zero-cost proxies *vs.* Latency profiling.** Zero-cost proxies such as the number of parameters and floating-point operations (FLOPs) that estimate the model complexity and latency are widely used in NAS due to their immediate availability. However, FLOPs do not accurately capture the actual edge device latency due to the intricacies introduced by the hardware implementation and compiler optimization. We show, in Figure 1, the normalized latency of different operators on different devices. Fused FFNs, despite having more FLOPs and parameters, have lower latency than MHSA layers on Nano and NCS2. Only ARM Cortex-A57 CPU follows the zero-cost proxies. Therefore, methods that estimate model complexity with zero-cost proxies fail to generalize to a wide range of edge devices.

**Unified FFN *vs.* Fused FFN.** Although unified FFNs learn spatial relationships in the feature maps by a depthwise convolution with fewer parameters and fewer theoretical FLOPs, they are highly memory-bound with low arithmetic intensity. On the other hand, the fused FFNs use vanilla convolutions that are compute-bound, as shown in Figure 1. The latency greatly depends on the input feature map size and on the hardware platform. Therefore, prior work Akin et al. (2022); Tan & Le (2021) replaces early stages with fused convolutions. In our preliminary study, the fused FFNs not only improve the accuracy by $1.4\%$ as shown in Table 4 but are also well supported by some edge devices and are more suited for the early network stages. As Figure 1 shows, NCS2 favors fused FFNs over unified FFNs. We include two operators in our supernet to support more subnet choices for different edge devices.

**Feedforward network *vs.* Multi-head self-attention layer.** MHSA layers are the major bottleneck in improving latency and are not well supported by most edge devices (*cf.* Figure 1). However, MHSAs learn the long-range dependencies in the feature map which greatly boosts performance. As shown in Table 4, reducing the value ratio from $4$ to $2$ in the self-attention layers ($\dim(V) = \dim(\text{Input}) \times \text{ratio}$) hurts the accuracy by $2.4\%$. Our framework searches *the number and the position* of MHSA layers along with their expansion ratios for the value matrices. For a device that is not suitable for MHSA, our framework reduces the number of MHSA layers in the architecture. In contrast, if a compiler or implementation improves the support on the device Dao et al. (2022); Dao (2023), our search adaptively increases the number of MHSA layers in the model to boost the accuracy under the same latency constraint.

**Table 4:** We evaluate the accuracy (Acc.) on ImageNet 1k with different variants of EfficientFormerV2 S0 and show end-to-end latency (ms) on different devices in the last column.

| FFN | Act. | $V$ ratio | Acc. (%) | Cortext / Nano TRT / NCS2 (ms) |
|---|---|---|---|---|
| Unified | GELU | 4 | 75.7 | 190.3 / 21.4 / 47.0 |
| Unified | GELU | **2** | 73.3 | 168.7 / 18.1 / 42.1 |
| Unified | **ReLU** | 4 | 74.4 | 105.6 / 18.3 / 30.6 |
| **Fused** | GELU | 4 | 77.1 | 237.6 / 26.2 / 36.6 |
| **Fused** | **ReLU** | 4 | 76.6 | 187.7 / 22.3 / 26.6 |

**GELU *vs.* ReLU.** While GELU activation Hendrycks & Gimpel (2016) improves accuracy, it is often not well supported by low-cost edge devices. In contrast, most devices and compilers support conv-relu fusion which provides optimal latency onnx.ai; Intel. As shown in Table 4, the latency is greatly improved by 3.1ms ($14.4\%$) by replacing GELU with ReLU Agarap (2018) in the EfficientFormerv2 S0. We include GELU and ReLU as the activation choices for different layers in our supernet. Our framework searches for the optimal activation combinations in the model architectures.

## B DUAL *vs.* ENTANGLED FFNS

We show two different designs of the supernet to accommodate two types of FFN in Figure 5. The dual feedforward networks are composed of two sets of separated weight matrices, while entangled feedforward networks share the expansion and projection weight matrices. Only one type of FFN will be activated (unified or fused) in a block during the training time. We empirically find the dual FFN converges slightly better than entangled FFN and has higher accuracy of the subnets, as shown in Figure 6. We use dual FFN in the supernet for all experiments.

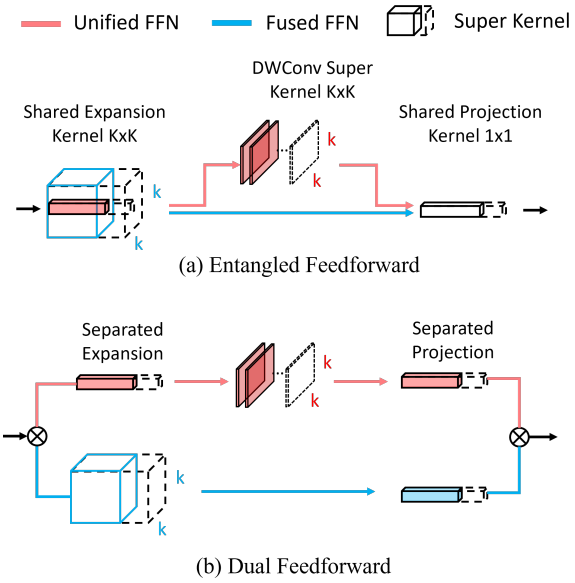



**Figure 5:** The supernet design of (a) Entangled FFN and (b) Dual FFN.

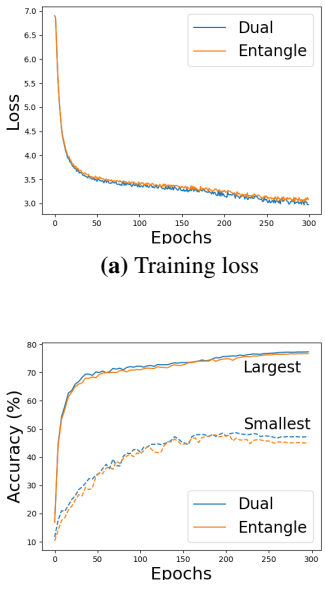



**Figure 6:** Dual *vs.* Entangled FFNs.

## C ACCURACY PREDICTOR

Since evaluating thousands of subnets on the validation set in the search process is not practical, we train a neural network $\chi$ to predict the accuracy, similar to Cai et al. (2020). Every block is encoded with a 24-bit length binary string where stages (4-bit), in/output width (8-bit), expansion ratios (6-bit), FFN types (2-bit), kernel sizes (2-bit), and activation functions (2-bit) are one-hot encoded. The binary string is stacked as a matrix $n \times 24$ and padded to 44 rows for a $n$-block subnet resulting in a $44 \times 24$ matrix. The network $\chi$ is built with a sequence of 1-D convolution followed by linear layers and it outputs a scalar accuracy prediction. We train the accuracy predictor using L1 loss, such that

$$\arg\min_{\chi} \quad |\chi(\alpha_i, w_i) - \text{Acc}(\alpha_i, w_i)| \quad .$$

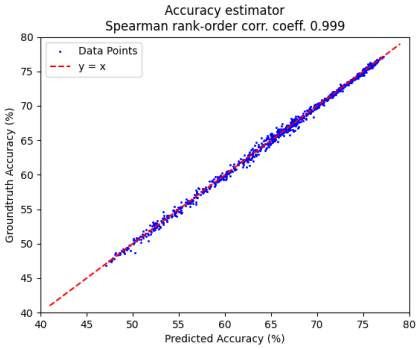



**Figure 7:** The quality of the accuracy predictor

We collect 6k subnet-accuracy pairs on ImageNet Deng et al. (2009) validation set, where the accuracy is evaluated with the weights directly inherited from the supernet (without fine-tuning). The collected pairs are divided into 5k for training and 1k for validation. Our goal of the accuracy predictor is to preserve the accuracy rank of the subnets. Therefore, we use the best predictor that has the highest

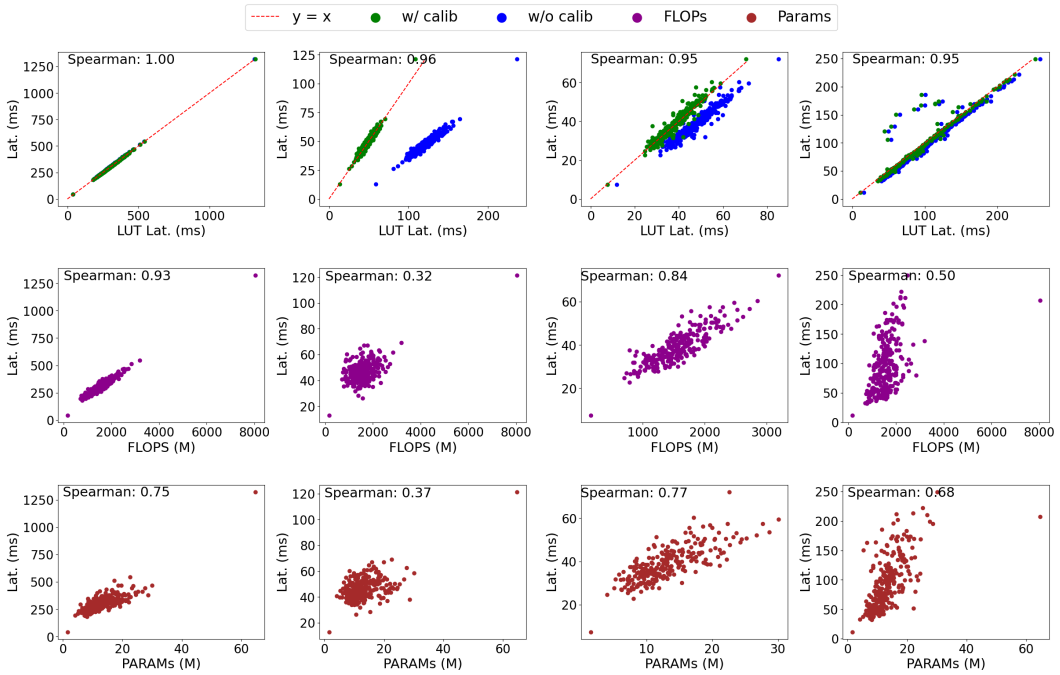
Spearman's rank correlation as Lee et al. (2021) and Dudziak et al. (2020). Our accuracy predictor is simple yet effective and highly preserves the ranking order, as shown in Figure 7. We use the trained accuracy predictor as the proxy, and therefore, our objective becomes

$$\alpha^*, w^* = \underset{\alpha \in \mathcal{A}, w \in \mathcal{W}}{\arg\max} \quad \chi(\alpha_i, w_i)$$
$$\text{s.t.} \quad \zeta_i(\alpha, w) < c_i, \quad i = 1, 2, ..., n \quad .$$

We note that the accuracy predictor is device-agnostic, and can be reused in the search process for all devices once it is trained.

## D Comparing LUTs with Zero-cost Proxies

We compare our method with zero-cost proxies in Figure 8. The results show that the zero-cost proxies such as the number of floating point operations (FLOPs) and the number of parameters do not reflect the real latency on diverse devices. We profile all possible blocks on the devices and build latency lookup tables (LUTs). However, LUTs overestimate the latency estimation since the devices usually cache the intermediate feature maps. To this end, we additionally profile 10 random subnets to calibrate our latency LUTs. The resulting latency estimations from the calibrated lookup tables not only highly preserve the order of the latency (high Spearman's rank correlation), but also accurately estimate the subnet on-device latency (The green dots fit $y = x$ line closely).




(a) Cortex-A57 (b) Intel NCS2 (c) Nano TensorRT (d) Nano CUDA

**Figure 8:** We compare the latency estimations with lookup tables (LUTs), number of floating point operations (FLOPs), and number of parameters (Params). We calibrate the LUTs with an additional 10 subnet end-to-end latency (Green points). Best viewed in color.

## E Supernet Training Details

We apply the sandwich rule to train our supernet similar to Yu & Huang (2019) which samples the smallest subnet, the biggest (full) subnet, and $M$ randomly sampled subnets ($M$ = 2 in our experiments). Additionally, we first sample the FFN structures for the largest and smallest subnets and use the sampled FFN in all blocks. For the $M$ randomly sampled subnets, we randomize the

FFN selection in every block. The smallest subnet is the one with the smallest width, depth, and kernel size in FFNs, and not containing MHSAs in the last two layers. Only MHSA downsampling is used in the third embedding layer in the smallest subnet. In contrast, the largest subnet has the largest width, depth, and kernel size, and MHSAs are applied before all the FFNs in layers $S3$ and $S4$. The FNNs and MHSAs are dropped by drop path Huang et al. (2016) with probabilities. We scale the FFN dropping probabilities according to the stage depth so that the last FFN in the stage has the highest probability of being dropped. All MHSAs are dropped with a constant probability so that the number as well as the position of MHSAs in the stage can be searched.

## F Search Space

We construct our supernet based on EfficientFormerV2 backbone. The supernet consists of four stages and three embedding layers. The embedding layers adjust the model width, *i.e.*, the channel dimension $C_i$, for the next stage. Each block in the first two stages, $S_1$ and $S_2$, only consists of FFNs. Every block in the last two stages, $S_3$ and $S_4$, consists of a MHSA followed by a FFN. Our search algorithm determines the width $C_i$, the number of FFN blocks $N_{\text{ffn}}$ in every stage $S_i$, the number of MHSA $N_{\text{mhsa}}$ in the last two stages for the network, and the expansion ratios $E_i^j$ (FFN and MHSA) and kernel size $K_i^j$ (FFN only) for every block $j$ in stage $i$. The full search space is shown in Table 5.

**Table 5:** Full search space. Our search space includes fused FFN (Fused), Unified FFN (unified), and Multi-head self-attention (MHSA). The third embedding layer (Embed 3) is an MHSA down-sampling layer (DS). We search the number of blocks in each stage ($N_{\text{ffn}}$ and $N_{\text{mhsa}}$) as well as the activation (GELU (G) or ReLU (R)), expansion ration $E_i^j$, and kernel size $K_i^j$ in each block. We abbreviate output resolution (Res.), and output channel width (Ch.). The channel width is represented in (min, max, step size).

| Stage | Type | Res. | Ch. ($C_i$) | $E_i^j$ | $K_i^j$ | Act. | #n | Depth |
|---|---|---|---|---|---|---|---|---|
| Stem | Conv | 1/4 | $C_1$ = (24, 36, 4) | - | 3 | {G, R} | 8 | - |
| Stage 1 | {Fused, Unified} | 1/4 | $C_1$ | {2, 3, 4} | {3, 5} | {G, R} | 96 | $N_{\text{ffn}}$ = {2, 3} |
| Embed 1 | Conv | 1/8 | $C_2$ = (40, 64, 8) | - | 3 | - | 16 | - |
| Stage 2 | {Fused, Unfied} | 1/8 | $C_2$ | {2, 3, 4} | {3, 5} | {G, R} | 96 | $N_{\text{ffn}}$ = {2, 3} |
| Embed 2 | Conv | 1/16 | $C_3$ = (96, 132, 12) | - | 3 | - | 16 | - |
| Stage 3 | MHSA | 1/16 | $C_3$ | {2, 3, 4} | - | {G, R} | 24 | $N_{\text{mhsa}} = \{n \mid 0 \leq n \leq N_{\text{ffn}}\}$ |
| | {Fused, Unified} | 1/16 | $C_3$ | {2, 3, 4} | {3, 5} | {G, R} | 96 | $N_{\text{ffn}}$ = {6, 7, 8, 9} |
| Embed 3 | MHSA DS | 1/32 | $C_4$ = (176, 248, 24) | {2, 3, 4} | - | {G, R} | 96 | - |
| Stage 4 | MHSA | 1/32 | $C_4$ | {2, 3, 4} | - | {G, R} | 24 | $N_{\text{mhsa}} = \{n \mid 0 \leq n \leq N_{\text{ffn}}\}$ |
| | {Fused, Unified} | 1/32 | $C_4$ | {2, 3, 4} | {3, 5} | {G, R} | 96 | $N_{\text{ffn}}$ = {4, 5, 6} |

## G Experimental Setup

We follow the setup of Li et al. (2023). Our models are implemented with PyTorch Paszke et al. (2019) framework and Timm library Huggingface. We use 24 A5000 GPUs to train the supernet for 300 epochs with a total batch size of 3072 and use 8 A5000 GPUs to fine-tune the searched subnets for 150 epochs with a total batch size of 2048 on ImageNet 1k training set Deng et al. (2009). Models are validated on the ImageNet validation set. Both training and validation are using the standard resolution $224 \times 224$. We also use AdamW optimizer Loshchilov & Hutter (2019), set the initial learning rate to $10^{-3} \times \text{batch size}/1024$ to train the supernet, and use $10^{-4} \times \text{batch size}/1024$ to fine-tune the subnets. The cosine decay is applied in both training and fine-tuning. RegNetY-16GF Radosavovic et al. (2020) with 82.9% top-1 accuracy are used in supernet training and subnet fine-tuning as the teacher model for hard distillation, as Li et al. (2023) and Touvron et al. (2021).

## H Details of the Comparison

We show the results in Figure 4 and list the details of the comparison in Table 6. The details of searched architectures are listed in Appendix I. Once-for-all (OfA, Conv only) is also shown

in Table 6 and Figure 4 for reference. The models searched from our framework are named ours_{*platform*}_{*compiler*}@{*lat*}ms where *lat* is the search latency constraint. For Cortex-A57, we use only the ONNX default compiler, so we omit {*compiler*} in Table 6. Our hybrid models outperform MobileNetV2 in accuracy while maintaining the same level of inference latency. Hybrid model counterparts, while having lower FLOPs or model size, fail to achieve MobileNetV2 latency.

**Table 6:** The table shows the details of the model comparison. The naming follows ours_{*platform*}_{*compiler*}@{*lat*}ms. We also pivot Once-for-all (OfA, Conv only) for reference. We abbreviate the types of model as Hybrid (H) and Convolution (C).

| Method | Type | Acc. (%) | FLOPs (M) | Param. (M) | Cortex (ms) | Nano TRT / Cuda (ms) | NCS2 (ms) |
|---|---|---|---|---|---|---|---|
| MobileNetV2 | C | 72.2 | 307.5 | 3.5 | 95.5 | 14.2 / 23.7 | 20.9 |
| ofa-pixel1-40 | C | 74.9 | 259.0 | 6.0 | 120.6 | - / - | - |
| ofa-pixel2-35 | C | 73.4 | 224.5 | 5.1 | 117.5 | - / - | - |
| ofa-tx2-47 | C | 72.9 | 409.5 | 4.9 | - | 16.4 / 26.8 | 22.1 |
| ofa-tx2-96 | C | 75.8 | 546.7 | 6.2 | - | 23.0 / 37.3 | 30.4 |
| MobileViT-XXS† | H | 69 | 414.3 | 1.3 | 141.7 | 17.6 / 44.0 | -* |
| EdgeNeXt-XXS† | H | 71.2 | 259.3 | 1.3 | 118.5 | 26.5 / 64.4 | -* |
| EdgeViT-XXS | H | 74.4 | 555.2 | 4.1 | 162.7 | 31.4 / 67.4 | -* |
| ElasticViT T1@224‡ | H | 75.8 | 205.1 | 8.9 | **75.4** | 17.5 / 30.1 | -* |
| ElasticViT S2 | H | 78.7 | 318.5 | 11.0 | 124.4 | 25.0 / 51.3 | -* |
| EfficentformerV2 S0 | H | 75.7 | 402.9 | 3.6 | 190.3 | 21.4 / 29.0 | 47.0 |
| ours_cortex@95ms | H | 74 | 441.3 | 4.6 | 98.6 | - / - | - |
| ours_cortex@150ms | H | 75.7 | 790.8 | 9.7 | 153.0 | - / - | - |
| ours_nano_trt@13ms | H | 73.4 | 806.4 | 7.9 | - | **14.7** / - | - |
| ours_nano_trt@20ms | H | 76.1 | 1437.4 | 13.5 | - | 22.5 / - | - |
| ours_nano_cuda@25ms | H | 75.6 | 963.0 | 7.0 | - | - / **24.3** | - |
| ours_nano_cuda@30ms | H | 76.3 | 1061.6 | 8.8 | - | - / 28.8 | - |
| ours_ncs2_ov@20ms | H | 75.4 | 1814.9 | 19.5 | - | - / - | **19.2** |
| ours_ncs2_ov@30ms | H | 77.3 | 2921.2 | 36.8 | - | - / - | 29.2 |

# I SEARCHED ARCHITECTURE DETAILS

We show the architecture details of searched subnets in Table 7 and Table 8. Our search algorithm tends to adopt depthwise convolutions (unified FFNs) with large-size kernels (*e.g.,* 5) and MHSA for ARM Cortex-A57. In contrast, our search algorithm is in favor of vanilla convolutions (fused FFNs) with small-size kernels (*e.g.,* 3) over depthwise convolutions (unified FFNs) and MHSA for Nvidia Jetson Nano 4G with TensorRT compiler. GELU and MHSA which boost accuracy are placed in the model closer to the output with minimal latency impact. The result shows that our search optimizes the model architecture for different hardware implementations and characteristics. Therefore, we expect that our search algorithm can propose more competitive model architectures once the MHSA and GELU are optimized for the target hardware.

**Table 7:** Architecture details of searched subnets on ARM Cortex-A57. We abbreviate output resolution (Res.), output channel width (Ch.), fused FFNs (Fused), Unified FFNs (unified), and Multi-head self-attention layers (MHSA), embedding layers (Embed 3), activation layers (A), GELU (G), ReLU (R), expansion ratio (E), and kernel size (K) in the Table.

| | | Cortex@150ms | | | Cortex@150ms_@5M | | | Cortex@95ms | | |
|---|---|---|---|---|---|---|---|---|---|---|
| Stage | Res. | Ch. | Type | E / K / A | Ch. | Type | E / K / A | Ch. | Type | E / K / A |
| Conv | 1/4 | 32 | Conv | - / 3 / R | 32 | Conv | - / 3 / R | 24 | Conv | - / 3 / R |
| Stage 1 | 1/4 | 32 | Fused | 2 / 3 / R | 32 | Fused | 3 / 3 / R | 24 | Fused | 2 / 3 / R |
| | | | Fused | 2 / 3 / R | | Fused | 3 / 3 / R | | Fused | 2 / 3 / R |
| Embed 1 | 1/8 | 40 | Conv | - | 64 | Conv | - | 40 | Conv | - |
| Stage 2 | 1/8 | 40 | Fused | 2 / 3 / R | 64 | Unified | 2 / 3 / R | 40 | Unified | 2 / 3 / R |
| | | | Fused | 2 / 3 / R | | Fused | 2 / 3 / R | | Fused | 2 / 3 / R |
| Embed 2 | 1/16 | 96 | Conv | - | 96 | Conv | - | 96 | Conv | - |
| Stage 3 | 1/16 | 96 | Unified | 3 / 3 / R | 96 | Unified | 3 / 5 / R | 96 | Unified | 3 / 3 / R |
| | | | MHSA | 2 / - / R | | MHSA | 2 / - / R | | MHSA | 2 / - / R |
| | | | Unified | 3 / 3 / R | | Unified | 3 / 5 / R | | Unified | 3 / 3 / R |
| | | | MHSA | 2 / - / R | | MHSA | 2 / - / R | | MHSA | 2 / - / R |
| | | | Fused | 2 / 3 / R | | Unified | 3 / 5 / R | | Unified | 3 / 3 / R |
| | | | MHSA | 2 / - / R | | MHSA | 2 / - / R | | MHSA | 2 / - / R |
| | | | Unified | 4 / 3 / R | | Unified | 4 / 5 / R | | Unified | 4 / 3 / R |
| | | | Unified | 2 / 5 / R | | Unified | 4 / 5 / R | | Unified | 2 / 3 / R |
| | | | MHSA | 2 / - / R | | MHSA | 2 / - / R | | MHSA | 2 / - / R |
| | | | Unified | 3 / 3 / R | | Unified | 3 / 5 / R | | Unified | 3 / 3 / R |
| Embed 3 | 1/32 | 248 | MHSA DS | 4 / - / R | 224 | MHSA DS | 4 / - / R | 224 | MHSA DS | 2 / - / R |
| Stage 4 | 1/32 | 248 | MHSA | 2 / - / R | 224 | MHSA | 2 / - / G | 224 | MHSA | 2 / - / R |
| | | | Fused | 3 / 3 / R | | Unified | 3 / 5 / G | | Unified | 3 / 3 / R |
| | | | MHSA | 2 / - / G | | MHSA | 2 / - / G | | MHSA | 2 / - / R |
| | | | Fused | 3 / 3 / G | | Unified | 3 / 5 / G | | Unified | 3 / 3 / R |
| | | | MHSA | 2 / - / R | | MHSA | 2 / - / R | | MHSA | 2 / - / R |
| | | | Unified | 3 / 3 / R | | Unified | 4 / 5 / R | | Unified | 3 / 3 / R |
| | | | MHSA | 2 / - / G | | MHSA | 2 / - / R | | MHSA | 2 / - / R |
| | | | Fused | 3 / 3 / G | | Unified | 3 / 5 / R | | Unified | 3 / 3 / R |

**Table 8:** Architecture details of searched subnets on Nano with TensorRT. We abbreviate output resolution (Res.), output channel width (Ch.), fused FFNs (Fused), Unified FFNs (unified), and Multi-head self-attention layers (MHSA), embedding layers (Embed 3), activation layers (A), GELU (G), ReLU (R), expansion ratio (E), and kernel size (K) in the Table.

| | | Nano_tensorrt_@20ms | | | Nano_tensorrt_@20ms_@5M | | | Nano_tensorrt_@13ms | | |
|---|---|---|---|---|---|---|---|---|---|---|
| Stage | Res. | Ch. | Type | E / K / A | Ch. | Type | E / K / A | Ch. | Type | E / K / A |
| Conv | 1/4 | 32 | Conv | - / 3 / R | 32 | Conv | - / 3 / R | 32 | Conv | - / 3 / R |
| Stage 1 | 1/4 | 32 | Fused | 3 / 3 / R | 32 | Fused | 4 / 3 / G | 32 | Unified | 2 / 3 / R |
| | | | Fused | 3 / 3 / R | | Fused | 3 / 3 / R | | Fused | 3 / 3 / R |
| Embed 1 | 1/8 | 64 | Conv | - | 64 | Conv | - | 64 | Conv | - |
| Stage 2 | 1/8 | 64 | Fused | 3 / 3 / G | 64 | Fused | 2 / 3 / G | 64 | Unified | 2 / 3 / R |
| | | | Fused | 3 / 3 / G | | Fused | 2 / 3 / G | | Unified | 2 / 3 / R |
| Embed 2 | 1/16 | 96 | Conv | - | 120 | Conv | - | 96 | Conv | - |
| Stage 3 | 1/16 | 96 | Fused | 3 / 3 / R | 120 | Unified | 3 / 5 / R | 96 | Fused | 3 / 3 / R |
| | | | Fused | 3 / 3 / R | | Unified | 3 / 5 / R | | Fused | 3 / 3 / R |
| | | | Fused | 3 / 3 / R | | MHSA | 2 / - / G | | Fused | 3 / 3 / R |
| | | | Fused | 3 / 3 / R | | Unified | 2 / 5 / G | | Fused | 3 / 3 / R |
| | | | Fused | 3 / 3 / G | | Unified | 2 / 5 / R | | Fused | 3 / 3 / G |
| | | | Fused | 3 / 3 / G | | MHSA | 2 / - / G | | Fused | 3 / 3 / G |
| | | | Fused | 3 / 3 / R | | Unified | 3 / 5 / G | N/A | | |
| | | | Fused | 3 / 3 / G | | Unified | 3 / 5 / R | | | |
| | | | Fused | 3 / 3 / G | N/A | | | | | |
| Embed 3 | 1/32 | 224 | MHSA DS | 2 / - / G | 248 | MHSA DS | 2 / - / R | 224 | MHSA DS | 2 / - / R |
| Stage 4 | 1/32 | 224 | Fused | 3 / 3 / G | 248 | MHSA | 2 / - / G | 224 | MHSA | 2 / - / R |
| | | | MHSA | 2 / - / G | | Unified | 3 / 5 / G | | Fused | 2 / 3 / R |
| | | | Fused | 3 / 3 / G | | MHSA | 2 / - / G | | Fused | 2 / 3 / G |
| | | | Fused | 3 / 3 / G | | Unified | 3 / 5 / G | | MHSA | 2 / - / G |
| | | | Fused | 3 / 3 / G | | Unified | 3 / 5 / G | | Fused | 3 / 3 / G |
| | | | Fused | 3 / 3 / R | | MHSA | 2 / - / R | | Fused | 2 / 3 / G |
| | | | MHSA | 2 / - / G | | Unified | 3 / 5 / R | N/A | | |
| | | | Fused | 3 / 3 / G | N/A | | | | | |