YOLOBench: Benchmarking Efficient Object Detectors on Embedded Systems Supplemental Material

Ivan Lazarevich Matteo Grimaldi Ravish Kumar Saptarshi Mitra
Shahrukh Khan Sudhakar Sah
Deeplite
ivan.lazarevich@deeplite.ai

A. Latency measurements.

Details regarding hardware platforms used to collect latency measurements are outlined in Table S1. Figures S1 and S2 show the difference of latency value distributions between devices computed for the full initial *YOLOBench* architecture space consisting of ∼1000 models. While generally good correlation is observed between model inference latencies on different devices (see also Figure S3), notably latency values measured on Khadas VIM3 NPU differ significantly from latency values on other devices. That is, for models with roughly the same latency on Jetson Nano GPU or Raspi4 ARM CPU, the difference in VIM3 NPU latency could be up to several times. This difference between NPU values from other common GPU/CPU-based platforms highlights the necessity to develop hardware-aware architecture design and search methods. The difference in the NPU benchmark is also reflected in the structure of model Pareto frontiers (Figs. 1, S6, S4) and the performance of zero-cost predictors in identifying Pareto-optimal models (Figs. 4, S18).

B. *YOLOBench* Pareto frontiers for different datasets.

YOLOBench Pareto frontiers for SKU-110k, WIDER FACE, and COCO datasets are shown in Figs. S4, S5, S6, correspondingly. Note that while mAP$_{50-95}$ values for VOC, SKU-110k, and WIDER FACE datasets are obtained by fine-tuning COCO pre-trained weights (all trained at 640x640 image resolution) on multiple image resolutions considered in *YOLOBench* (11 values from 160 to 480 with a step of 32), the mAP$_{50-95}$ values on the COCO dataset are obtained by directly evaluating pre-trained COCO weights, without fine-tuning on the corresponding target image resolutions. This corresponds to the situation of deployment of pre-trained COCO weights without any additional training.

Table S2 shows the identified Pareto-optimal YOLO models on 3 different datasets and 4 hardware platforms under several latency thresholds. It can be noted that under the same latency threshold on a given hardware platform, the optimal YOLO model family and input image resolution are typically dataset-dependent.

Figures S7 and S8 show the statistics of architecture scaling parameters (width factor, depth factor, image resolution) in Pareto-optimal models on Raspberry Pi4 CPU and VIM3 NPU, respectively. Although some differences are observed between devices and datasets (in particular depth factor distributions), there is a general trend in all computed Pareto fronts where a variation in depth/width factors is observed at higher resolutions, and resolution is reduced when the depth/width factors (especially the width factor) already have low values.

C. Performance of zero-cost accuracy predictors on *YOLOBench*.

The performance of zero-cost accuracy predictors used in neural architecture search [3] is empirically evaluated on *YOLOBench* models on VOC and SKU-110k. Table S3 shows the Kendall-Tau scores and Pareto-optimal model prediction recall values obtained by a variety of zero-cost predictors. The zero-cost predictor values are computed using a randomly sampled batch of test set data with batch size $= 16$ (the used batch was the same for all ZC metrics). MAC count and the number of parameters are computed for models in evaluation mode, with normalization layers fused into preceding convolutions (if possible), and RepVGG-style blocks [4] also fused, if present in the model. Hence, the performance of MAC and parameter counts might slightly differ if computed for models in training mode. Most predictors perform poorly and are outperformed by the MAC count baseline, except for the NWOT score (in particular the pre-activation version of it). The good performance of NWOT can be also observed in Fig. S10, where scatter plots of fine-tuned model mAP$_{50-95}$ vs. zero-cost predictor value are shown for a few predictors. Some predictors (notably parameter count, ZiCo, and Zen-score) can be observed to produce very close values for subsets of models with signif-

Table S1. Details on hardware platforms and corresponding runtimes used for benchmarking.

	Raspberry Pi 4 Model B	**Jetson Nano (NVIDIA)**	**Khadas VIM3**	**Lambda tensorbook**
CPU	Quad Core Cortex-A72, 64-bit SoC @1.8GHz	Quad Core Cortex-A57 MPCore, 64-bit SoC @1.43GHz	Quad Core Cortex-A73 @2.2Ghz, Dual Core Cortex-A53 @1.8Ghz	Intel® Core™i7-10875H CPU @ 2.30GHz
Memory	4GB LPDDR4-3200 SDRAM	4 GB 64-bit LPDDR4, 1600MHz 25.6 GB/s	4GB LPDDR4/4X	64GB DDR4 SDRAM
AI-chip	-	NVIDIA Maxwell GPU, 128 NVIDIA CUDA® cores	Custom NPU INT8 inference up to 1536 MAC	NVIDIA RTX 2080 Super Max-Q
Ops	-	472 GFLOPs	5.0 TOPS	-
Framework/runtime	TensorFlow Lite (FP32, XNNPACK backend)	ONNX Runtime (FP32, GPU)	AML NPU SDK (INT16)	OpenVINO (CPU, FP32)

icantly different accuracy. This is an indication of the fact that these predictors perform poorly in estimating accuracy differences in models when the underlying architecture is fixed, but the input image resolution is varied.

We also test the performance of a training-based predictor on *YOLOBench* which is the mAP$_{50-95}$ values of models trained on a representative dataset (VOC) from scratch for 100 epochs. This predictor sets a strong baseline to be outperformed by training-free predictors, as it is generally found to perform well on a variety of datasets (see Fig. S9), including datasets from different visual domains (e.g. SKU-110k).

We further look into the robustness of the results obtained with the pre-activation NWOT estimator. Since this zero-cost estimator does not require computing the loss function, the main parameters that could influence its performance are the exact batch of data sampled, the batch size, and the dataset split (training or test data) used to sample the batch. Figure S11 shows the global Kendall-Tau scores achieved with pre-activation NWOT on VOC *YOLOBench* models with different batches sampled, different batch sizes and different data splits used. There is an observed variance in performance depending on the sampled batch, which is higher when the test set data are used (with an absolute difference of up to 0.05 in global Kendall-Tau score). Notably, scores computed on training set data (with augmentations) performed better on average compared to test set data, and performance is observed to decrease with increasing batch size. Furthermore, Table S4 shows the mean and standard deviation of Kendall-Tau scores for the standard and pre-activation versions of NWOT on VOC *YOLOBench* models computed on 5 different batches of size 16. We also estimate the performance of the mean predictor values averaged over the 5 sampled batches, which is expectedly found to outperform predictors computed on single batches. Moreover, we compute the pre-activation NWOT scores for all layers in YOLO models except the ones contained in detection heads. This is motivated by the fact that the larger distances between binary activation codes in NWOT are meant to correlate with better performance for the feature extraction layers (e.g. layers in the backbone and neck of YOLO), not the last layers used to compute model predictions. We find an overall performance boost in terms of Kendall-Tau scores for the case when the NWOT score is computed only for the layers in the backbone and neck (Table S4).

D. Pareto-optimal model prediction using training-free proxies.

We evaluate the training-free accuracy predictors (and the training-based one, VOC training from scratch) for the task of predicting Pareto-optimal models. That is, if one computes the ZC values for each model and determines the Pareto set of models in the ZC value-real latency two-dimensional space, we want to estimate how many models in that Pareto set are going to also be present in the actual Pareto frontier (computed in the two-dimensional mAP$_{50-95}$-latency space). Two metrics are of importance here: recall (how many of actual mAP-latency Pareto-optimal models are captured by a ZC-based Pareto set) and precision (how many of ZC-based Pareto set models are actually Pareto optimal in the real mAP-latency space). Additionally, one could consider the first N ($N = 1, 2, 3, ...$) ZC-based Pareto sets to expand the set of potential model candidates. We look at how precision and recall values change with N for a few well-performing predictors (NWOT, pre-activation NWOT, MAC count, and VOC training from scratch) with latency values taken from different target devices.

Recall values for several zero-cost predictors for Pareto models on Jetson Nano GPU and VOC dataset are shown in Fig. S12. Corresponding precision values for a few well-performing predictors on 3 different HW platforms are shown in Fig. S13. Recall values for these best-performing predictors on the SKU-110k dataset are shown in Fig. S18.

A different way to evaluate the predictors on *YOLOBench* is to treat models with the same architectures but different input image resolutions as identical data points. That is, if a certain architecture is predicted by ZC-based Pareto front to be optimal on a certain resolution, we count that as a correct prediction if that same architecture on a different resolution is found to be really Pareto-optimal. Such a way to evaluate ZC performance stems from the fact that in practice one typically wishes to predict the most promising architectures, not necessarily the particular optimal image resolution (since that architecture would be pre-trained with a certain fixed resolution, e.g. 640x640 on a dataset like COCO for further fine-tuning on the target dataset). Recall and precision values for such an evaluation protocol for the VOC dataset are shown in Figs. S14, S15.

We also evaluate the performance of the best training-free predictor (pre-activation NWOT) in predicting Pareto-optimal models, when the latency values used are different from actual latency measurements, but either are computed via a latency proxy like MAC count or measurement on another device. Note that in the case of MAC count as a latency predictor, the whole Pareto-frontier computation process is zero-cost: the approximation for mAP is given by the pre-activation NWOT score, the approximation for latency by MAC count. One might wonder how such a fully zero-cost approach performs in practice. Figures S16 and S17 show the recall and precision values when accuracy predictor is taken to be pre-activation NWOT and latency predictors are varied from MAC count to latencies from other (proxy) devices. Interestingly, MAC count is found to perform relatively well in terms of recall, specifically for Raspberry Pi 4 CPU. Notably, none of the latency proxies work well to predict Pareto-optimal models on VIM3 NPU. Also, perhaps not surprisingly, using Intel CPU latency measurements works well to predict Pareto-optimal models on Raspberry Pi 4 CPU, but does not significantly outperform MAC count.

Finally, we test the pre-activation NWOT accuracy estimator to predict potentially well-performing models out of a set of YOLO models we generated with different CNN backbones from the `timm` package [6]. We have computed the NWOT-latency Pareto set for YOLO-PAN-C3 models with `timm` backbones on input images of 480x480 resolution, with latency measured on Raspberry Pi 4 ARM CPU (TFLite, FP32). The neck structure (PAN-C3) for each of the candidate models was taken to be that of YOLOv5s and the detection head to be that of YOLOv8 (same as for all *YOLOBench* models), with Hardswish activations in the neck and head, and activation function(s) in the backbone kept the same as originally implemented in `timm`. Table S5 shows examples of predicted Pareto-optimal models (a subset of the full NWOT-latency Pareto set). Based on these observations, we have selected FBNetV3-D as a potential backbone of a YOLO model to be trained on the COCO dataset and compared it to a reference YOLOv8 model in a similar latency range (YOLOv8s).

Table S7 shows COCO minival mAP$_{50-95}$ and inference latency results for a YOLO-FBNetV3-D-PAN-C3 model trained on the COCO dataset for 300 epochs and profiled on 640x640 input resolution on Raspi4 CPU with TFLite. We observe that the choice of activation function significantly affects TFLite model inference latency, so for a more fair comparison we also train and profile a Hardswish-based version of YOLOv8s in addition to its default SiLU-based version. While we observe a significant reduction in inference latency with a negligible mAP drop shifting from SiLU to Hardswish, the FBNetV3-based model still outperforms YOLOv8s-HSwish. Furthermore, we train and profile a ReLU-based version of YOLO-FBNetV3-D-PAN-C3 (with activation functions in the backbone kept to be those of the original backbone, i.e. Hardswish, but neck and detection head activations replaced with ReLU) and observe further latency improvements at the cost of $\sim 0.56\%$ drop in mAP$_{50-95}$. However, this model is still found to outperform YOLOv8s in terms of both accuracy and latency (see Table S7). Furthermore, we train the same models for 500 epochs with a batch size of 256, which is found to achieve better results on COCO minival and test (Table 4). Although we could not exactly reproduce COCO minival mAP results for YOLOv8s reported by Ultralytics [5], we find that the FBNetV3-based model outperforms both our YOLOv8s mAP results as well as those of Ultralytics, with lower latency on Raspberry Pi 4 CPU. The COCO minival mAP$_{50-95}$ values reported in Table 4 were obtained using `pycocotools` [2] (with IoU threshold for NMS $= 0.6$ and object confidence threshold for detection $= 0.001$), and mAP values on test-dev were obtained using the same evaluation parameters by submitting to the competition server [1]. More details on the performance comparison of models on COCO test-dev are shown in Table S6.

References

[1] Coco detection challenge. https://codalab.lisn.upsaclay.fr/competitions/7384. 3

[2] Pycocotools pypi package. https://pypi.org/project/pycocotools/. 3

[3] Mohamed S Abdelfattah, Abhinav Mehrotra, Łukasz Dudziak, and Nicholas D Lane. Zero-cost proxies for lightweight nas. *arXiv preprint arXiv:2101.08134*, 2021. 1

[4] Xiaohan Ding, Xiangyu Zhang, Ningning Ma, Jungong Han, Guiguang Ding, and Jian Sun. Repvgg: Making vgg-style convnets great again. In *Proceedings of the IEEE/CVF conference on computer vision and pattern recognition*, pages 13733–13742, 2021. 1

[5] Glenn Jocher, Ayush Chaurasia, and Jing Qiu. YOLO by Ultralytics, Jan. 2023. 3

[6] Ross Wightman. Pytorch image models. https://github.com/rwightman/pytorch-image-models, 2019. 3

Table S2. Pareto-optimal *YOLOBench* models on 3 datasets and 4 hardware platforms. Shown are the best models in terms of mAP$_{50-95}$ under a given latency threshold (max. latency). For each model, the scaling parameters are given (d33w25 means depth factor $= 0.33$ and width factor $= 0.25$), corresponding input resolution of the models is indicated in brackets.

HW/max. latency	VOC model	VOC mAP$_{50-95}$	SKU-110k model	SKU-110k mAP$_{50-95}$	WIDERFACE model	WIDERFACE mAP$_{50-95}$
Nano/0.5 sec	YOLOv8 d67w1 (448)	0.726	YOLOv7 d1w75 (480)	0.593	YOLOv7 d1w75 (480)	0.382
Nano/0.3 sec	YOLOv7 d1w5 (480)	0.701	YOLOv7 d1w5 (480)	0.589	YOLOv7 d1w5 (480)	0.369
Nano/0.1 sec	YOLOv7 d1w5 (288)	0.657	YOLOv8 d1w25 (480)	0.567	YOLOv7 d1w25 (480)	0.336
VIM3/0.3 sec	YOLOv8 d67w1 (448)	0.726	YOLOv7 d1w75 (480)	0.593	YOLOv7 d1w75 (480)	0.382
VIM3/0.1 sec	YOLOv6l d67w5 (384)	0.669	YOLOv8 d1w25 (480)	0.567	YOLOv6m d33w5 (480)	0.350
VIM3/0.05 sec	YOLOv6l d67w25 (416)	0.620	YOLOv6s d33w25 (480)	0.556	YOLOv6m d67w25 (480)	0.318
Intel/0.08 sec	YOLOv8 d1w75 (416)	0.719	YOLOv7 d1w75 (480)	0.593	YOLOv7 d1w75 (480)	0.382
Intel/0.04 sec	YOLOv7 d1w5 (480)	0.701	YOLOv7 d1w5 (480)	0.589	YOLOv7 d1w5 (480)	0.369
Intel/0.02 sec	YOLOv6l d6w5 (448)	0.682	YOLOv6l d33w5 (480)	0.576	YOLOv6l d33w5 (480)	0.346
Raspi4/3 sec	YOLOv8 d1w75 (416)	0.719	YOLOv7 d1w75 (480)	0.593	YOLOv7 d1w75 (480)	0.382
Raspi4/1 sec	YOLOv7 d1w5 (480)	0.701	YOLOv7 d1w5 (480)	0.589	YOLOv7 d1w5 (480)	0.369
Raspi4/0.5 sec	YOLOv6l d67w5 (384)	0.669	YOLOv4 d1w25 (480)	0.569	YOLOv7 d1w25 (480)	0.336

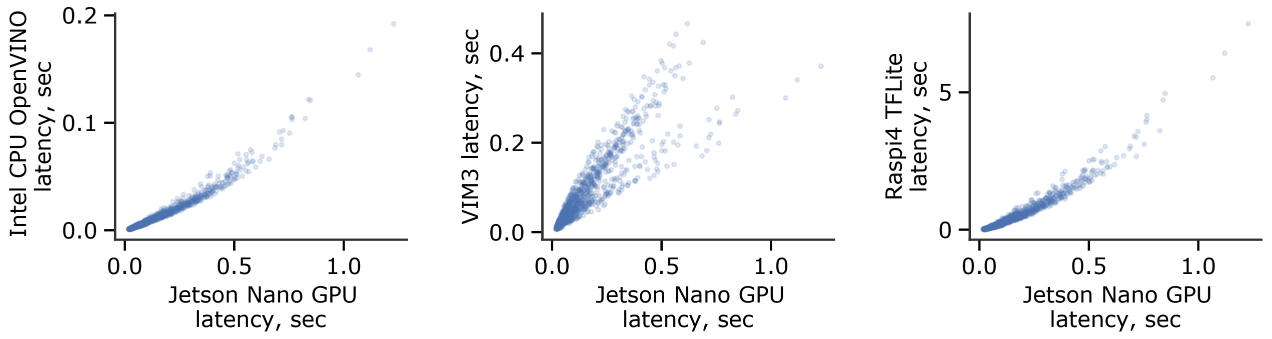

Figure S1. Scatter plots of latency values measured for *YOLOBench* models on the Jetson Nano GPU (ORT, FP32 precision) vs. latency values on other hardware platforms.

Table S3. Performance of training-free accuracy predictors on *YOLOBench* models and two datasets (VOC and SKU-110k, from COCO-pretrained weights) compared to using mAP$_{50-95}$ of models trained from scratch on the VOC dataset as a predictor.

	VOC, mAP$_{50-95}$			SKU-110k, mAP$_{50-95}$		
Predictor metric	global τ	top-15% τ	%Pareto pred. (GPU)	global τ	top-15% τ	%Pareto pred. (GPU)
GraSP	-0.011	-0.068	0.062	0.040	0.032	0.025
Plain	0.029	0.069	0.015	-0.388	-0.176	0.025
JacobCov	0.095	-0.078	0.015	0.541	0.136	0.025
ZiCo	0.195	0.016	0.015	0.115	0.081	0.025
Zen	0.255	0.092	0.062	0.146	0.121	0.050
GradNorm	0.262	0.173	0.015	-0.331	-0.072	0.025
Fisher	0.280	0.156	0.015	-0.380	-0.096	0.025
L2 norm	0.326	0.090	0.015	0.189	0.118	0.025
SNIP	0.336	0.217	0.015	-0.290	-0.059	0.025
#params	0.399	0.372	0.031	0.256	0.119	0.050
SynFlow	0.558	0.227	0.062	0.512	0.254	0.100
MACs	0.739	0.520	0.123	0.604	0.314	0.125
NWOT	0.756	0.622	0.262	0.703	0.321	**0.200**
NWOT (pre-act)	**0.827**	**0.623**	**0.292**	**0.765**	**0.406**	**0.200**
VOC training from scratch (mAP$_{50-95}$)	0.847	0.665	0.369	0.739	0.374	0.425

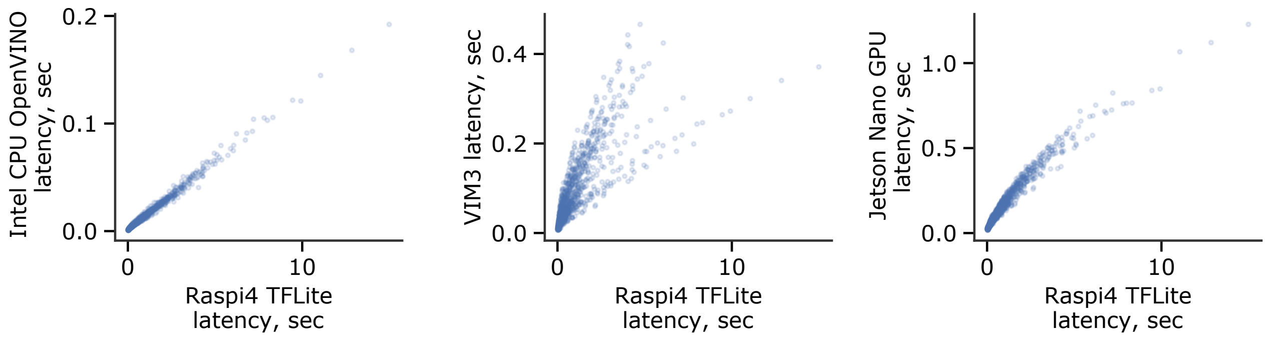

Figure S2. Scatter plots of latency values measured for *YOLOBench* models on the Raspberry Pi 4 CPU (TFLite with XNNPACK, FP32 precision) vs. latency values on other hardware platforms.

Table S4. Mean and standard deviation of the global Kendall-Tau scores for NWOT metrics computed for 5 different randomly sampled batches of size 16 on VOC *YOLOBench* models. The metric denoted as ”no head” was computed only for the layers contained in the neck and backbone of YOLO models, not in the detection head. The second column shows Kendall-Tau scores for prediction with the mean ZC metric values averaged over the 5 batches.

ZC metric	global τ	global τ (prediction with mean ZC value)
NWOT	0.7839 (0.0159)	0.7895
NWOT (pre-act)	0.8402 (0.0191)	0.8486
NWOT (pre-act, no head)	**0.8472 (0.0194)**	**0.8570**

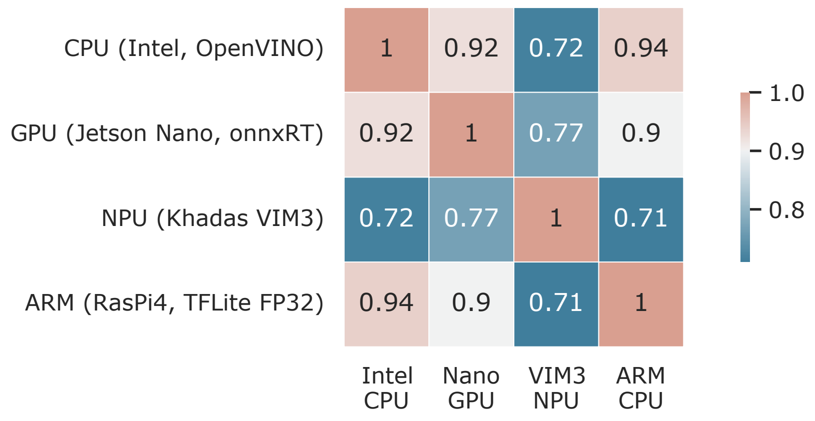

Figure S3. Correlation matrix (Kendall-Tau scores are shown) for latency values on 4 hardware platforms/runtimes considered in *YOLOBench*.

Table S5. Example YOLO-PAN-C3 models with `timm` backbones identified in the NWOT-latency Pareto frontier, with pre-activation NWOT score computed on the VOC dataset. Latency values are measured on Raspberry Pi 4 ARM CPU with TFLite (FP32), batch size 1.

Model name	Input resolution	Raspi4 CPU latency, sec	NWOT (pre-act)
`yolo_pan_efficientnet_b4`	480	1.72	511.84
`yolo_pan_tf_efficientnet_b4_ap`	480	1.71	511.77
`yolo_pan_gc_efficientnetv2_rw_t`	480	1.41	508.73
`yolo_pan_tf_efficientnet_lite4`	480	1.08	506.67
`yolo_pan_fbnetv3_d`	480	0.71	502.71
`yolo_pan_tf_efficientnet_lite1`	480	0.61	493.48
`yolo_pan_efficientnet_lite1`	480	0.61	493.32
`yolo_pan_mobilenetv2_110d`	480	0.54	480.92
`yolo_pan_mobilenetv2_075`	480	0.45	480.14
`yolo_pan_tf_mobilenetv3_large_075`	480	0.45	468.85
`yolo_pan_mobilenetv2_035`	480	0.37	457.41
`yolo_pan_tf_mobilenetv3_small_minimal_100`	480	0.36	451.10

Table S6. COCO test mAP values and inference latency on Raspberry Pi 4 CPU (TFLite with XNNPACK backend, FP32) for YOLOv8s vs. a model identified from the NWOT-latency Pareto frontier (YOLO-FBNetV3-D-PAN). For mAP values, the mean and standard deviation over three random seeds are shown. For inference time, mean and standard deviation of inference time over 5 runs (each one 100 iterations) are shown.

Model	AP^{test}_{50-95}	AP^{test}_{50}	AP^{test}_{75}	AP^{test}_{S}	AP^{test}_{M}	AP^{test}_{L}	Latency, ms
YOLOv8s	43.17% (0.12%)	60.53% (0.09%)	46.5% (0.08%)	**22.7%** (0.14%)	47.13% (0.17%)	57.0% (0.22%)	1476.09 (1.49)
YOLOv8s-HSwish	42.90% (0.0%)	60.3% (0.0%)	46.30% (0.0%)	22.46% (0.09%)	47.0% (0.08%)	56.39% (0.08%)	1381.62 (7.34)
YOLO-FBNetV3-D-PAN	**43.87%** (0.05%)	**61.53%** (0.09%)	**47.23%** (0.05%)	22.67% (0.19%)	**47.87%** (0.05%)	**58.36%** (0.12%)	**1355.21** (9.93)

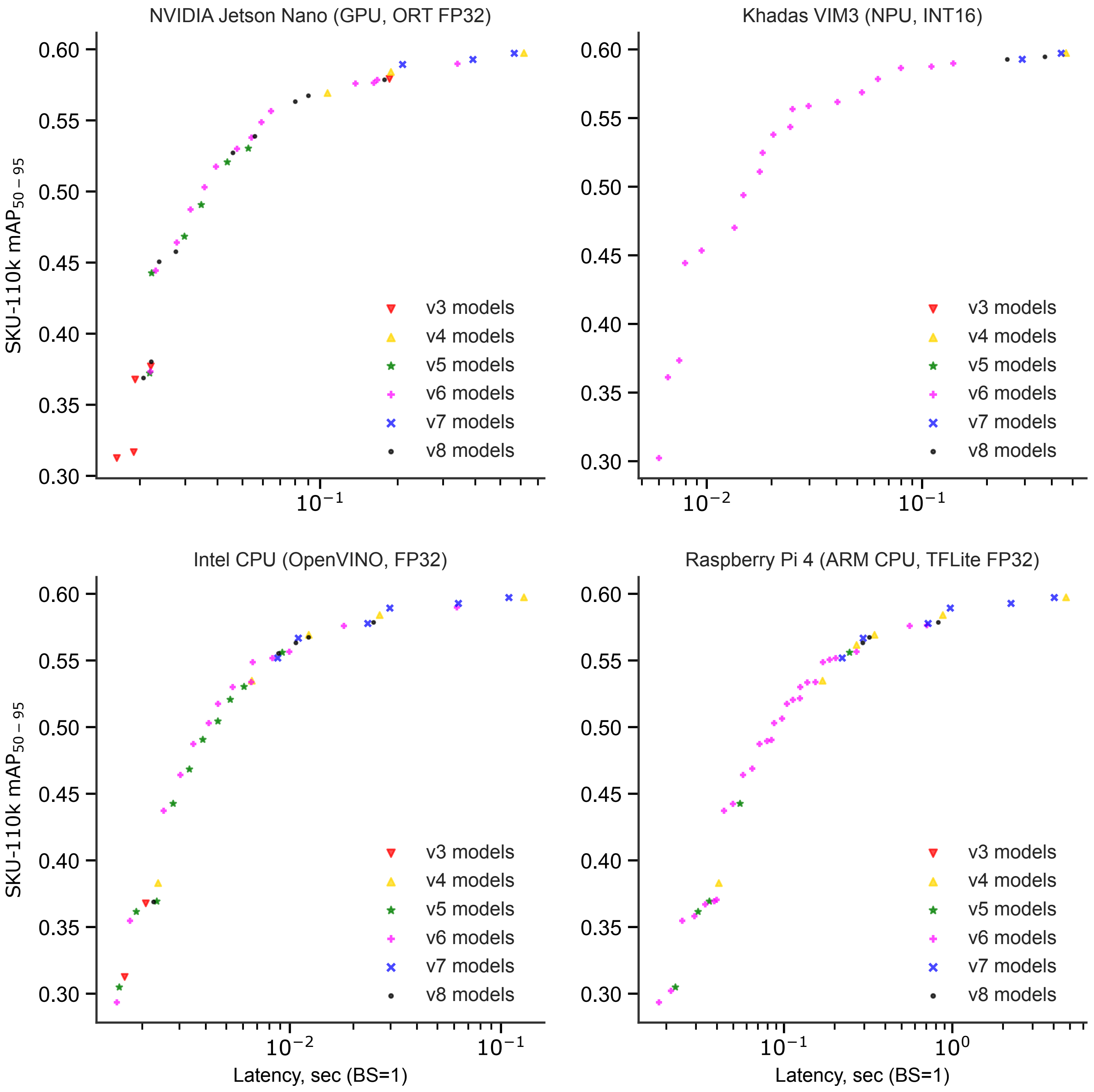

Figure S4. Pareto frontiers of *YOLOBench* models fine-tuned on the SKU-110k dataset (on several target resolutions) from COCO-pretrained weights on 4 different hardware platforms. Each point represents a single model in the mAP-latency space, with the model family coded with color and marker size (all YOLOv6-3.0 models are represented by the same color).

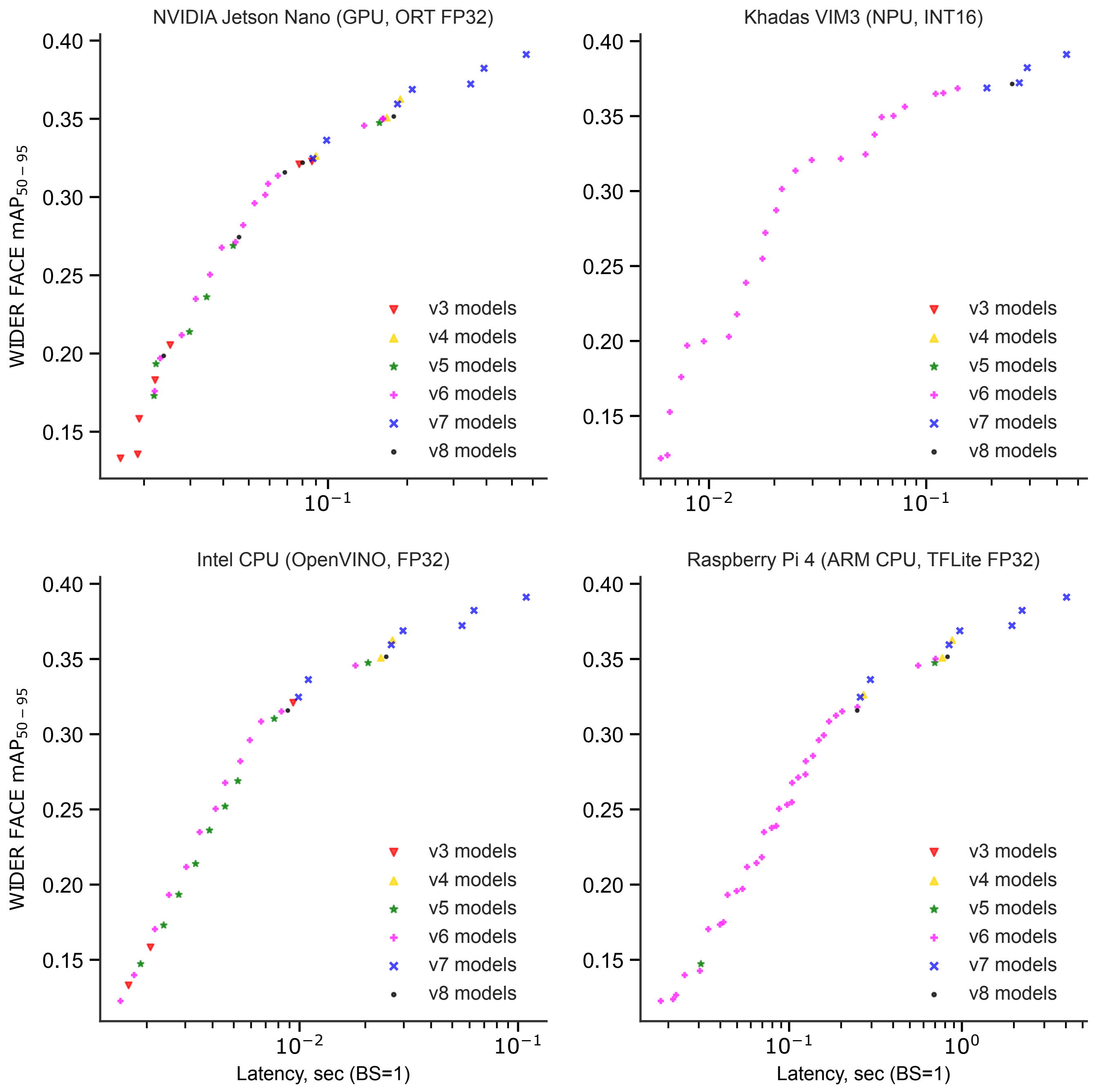

Figure S5. Pareto frontiers of *YOLOBench* models fine-tuned on the WIDER FACE dataset (on several target resolutions) from COCO-pretrained weights on 4 different hardware platforms. Each point represents a single model in the mAP-latency space, with the model family coded with color and marker size (all YOLOv6-3.0 models are represented by the same color).

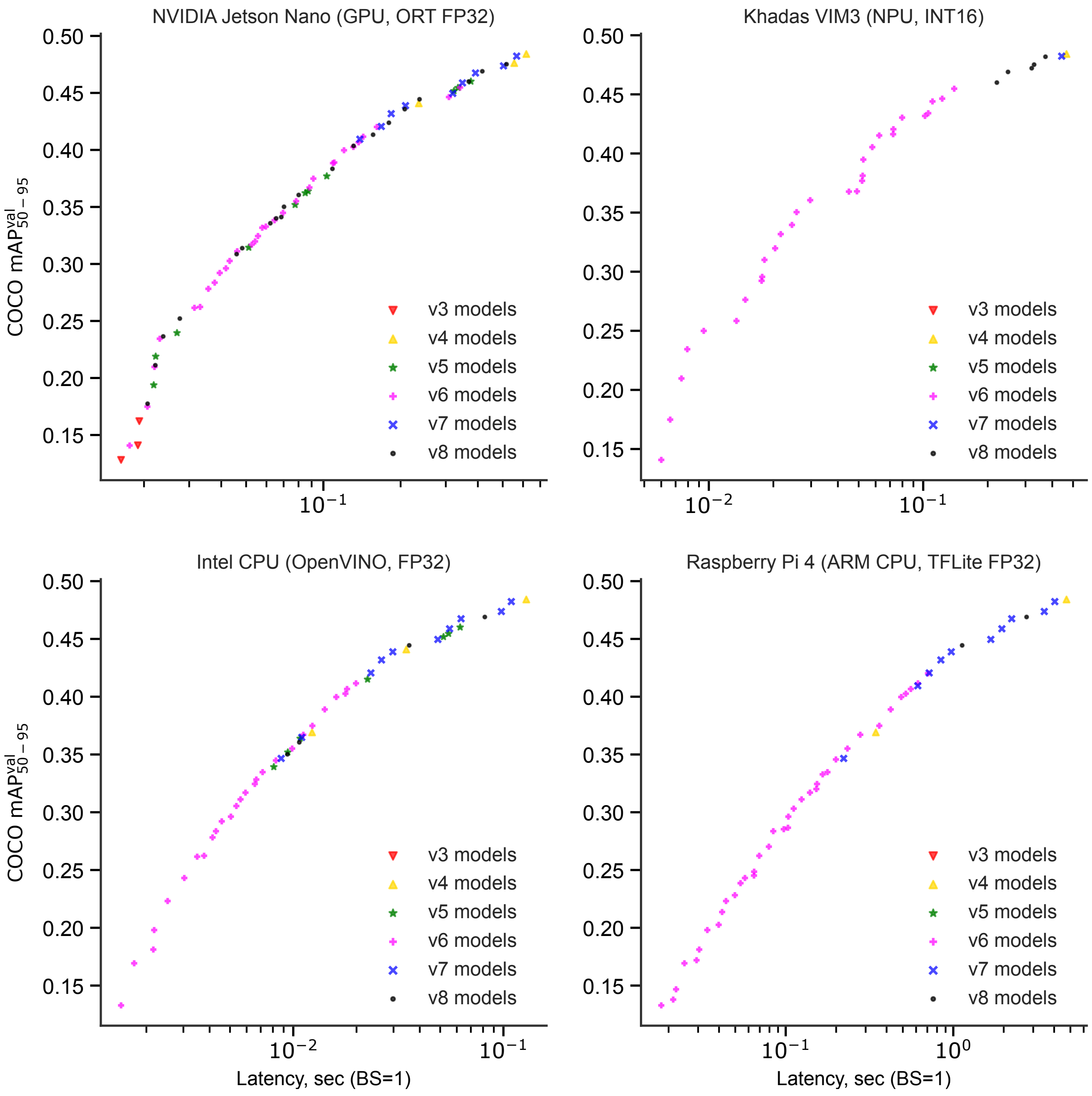

Figure S6. Pareto frontiers of *YOLOBench* models trained on the COCO dataset (640x640 image resolution) and validated on several target image resolutions on COCO minival (without additional fine-tuning) on 4 different hardware platforms. Each point represents a single model in the mAP-latency space, with the model family coded with color and marker size (all YOLOv6-3.0 models are represented by the same color).

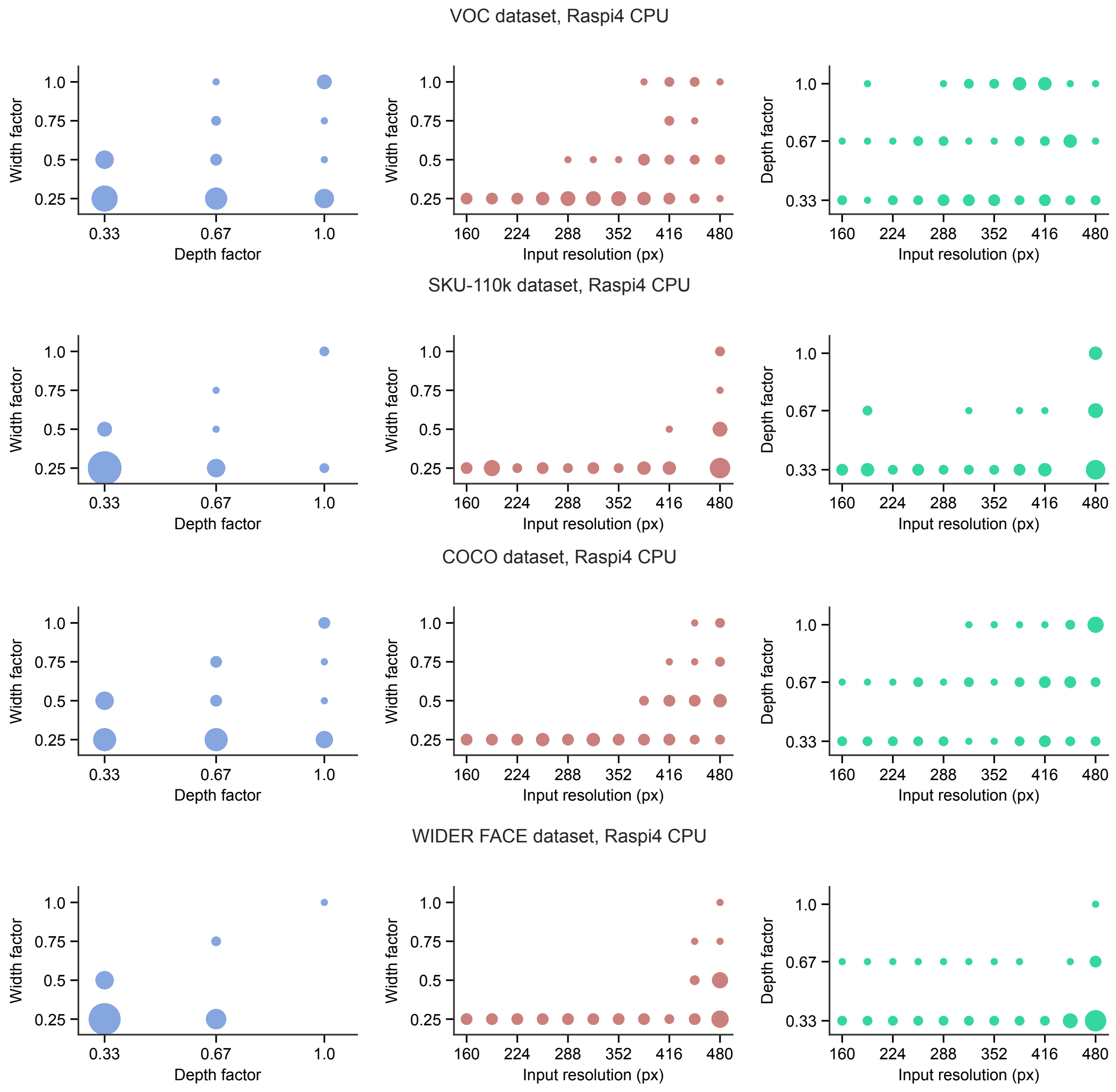

Figure S7. Statistics of model scaling parameters (depth factor, width factor, input resolutions) in Pareto-optimal models on 4 different datasets with latency measured on the Raspberry Pi 4 ARM CPU (TFLite, FP32). The size of each point (circle) is proportional to the number of models for that parameter combination.

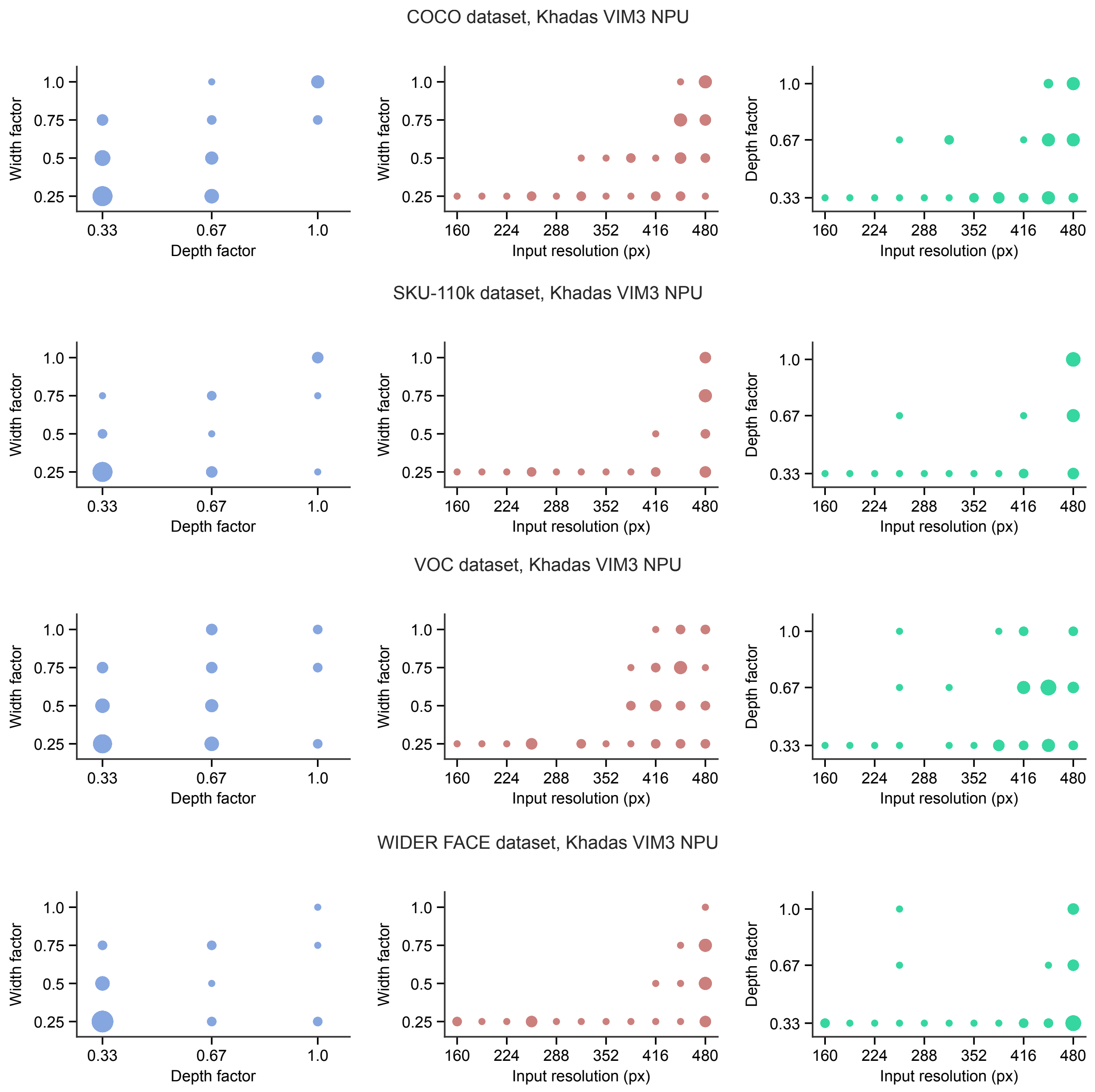

Figure S8. Statistics of model scaling parameters (depth factor, width factor, input resolutions) in Pareto-optimal models on 4 different datasets with latency measured on Khadas VIM3 NPU (INT16). The size of each point (circle) is proportional to the number of models for that parameter combination.

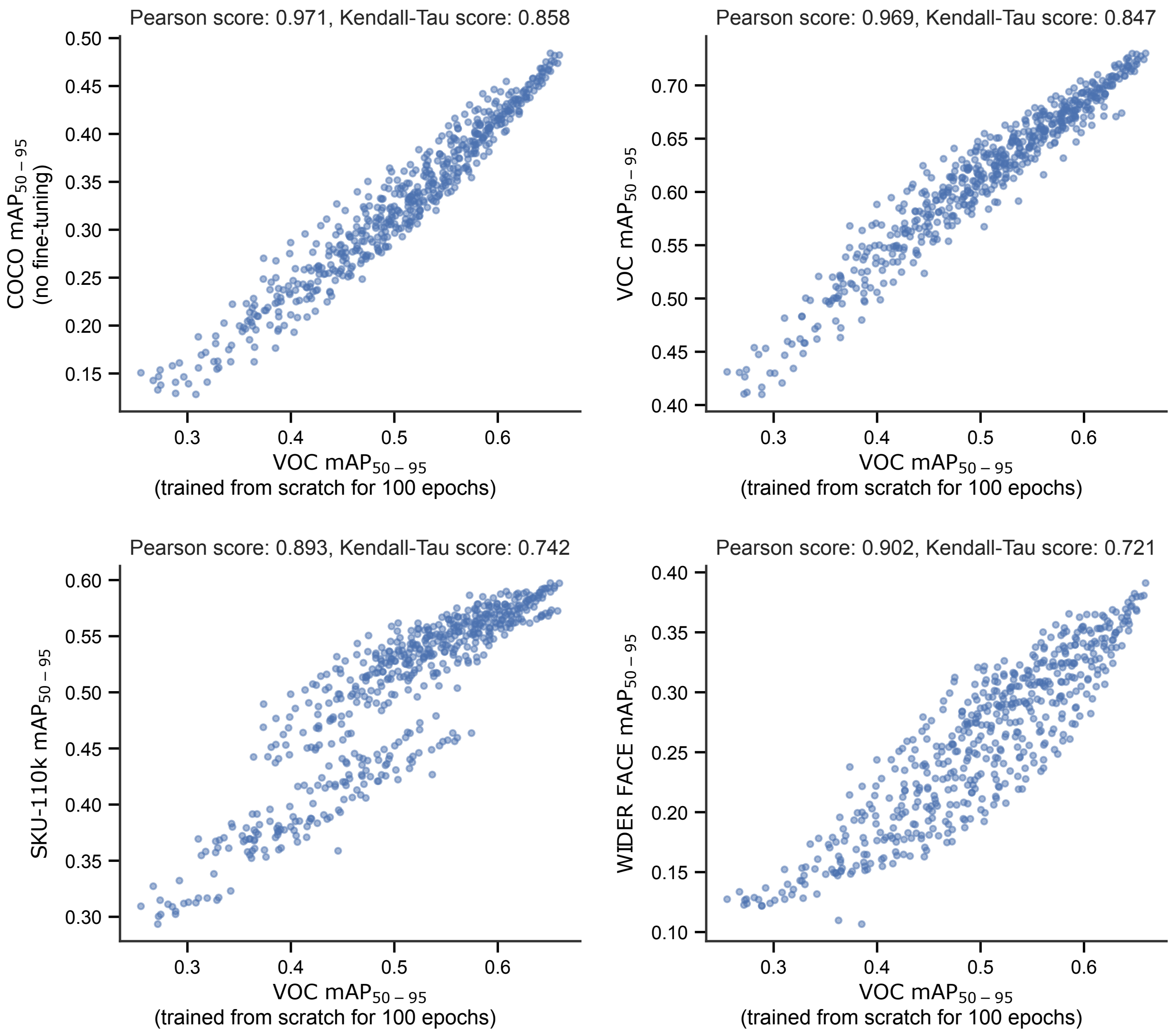

Figure S9. Scatter plots of mAP$_{50-95}$ values obtained on 4 *YOLOBench* datasets for models fine-tuned from COCO-pretrained weights (for all datasets except COCO) vs. mAP$_{50-95}$ of models trained on the VOC dataset from scratch for 100 epochs. High correlation of target metric and mAP of VOC training from scratch is observed for several datasets.

Table S7. COCO minival mAP and inference latency on Raspberry Pi 4 CPU (TFLite with XNNPACK backend, FP32) for YOLOv8s vs. a model identified from the NWOT-latency Pareto frontier (YOLO-FBNetV3-D-PAN). Mean and standard deviation of inference time over 5 runs (each one 100 iterations) are shown. The input image resolution used was 640x640, batch size $= 1$ for latency measurements. Models were trained for 300 epochs, with batch size $= 64$.

Model	COCO mAP$^{val}_{50-95}$	Raspberry Pi 4 ARM CPU latency, ms
YOLOv8s	43.64%	1476.09 (1.49)
YOLOv8s-HSwish	43.55%	1381.62 (7.34)
YOLO-FBNetV3-D-PAN	44.63%	1355.21 (9.93)
YOLO-FBNetV3-D-PAN-ReLU	44.07%	1344.50 (8.06)

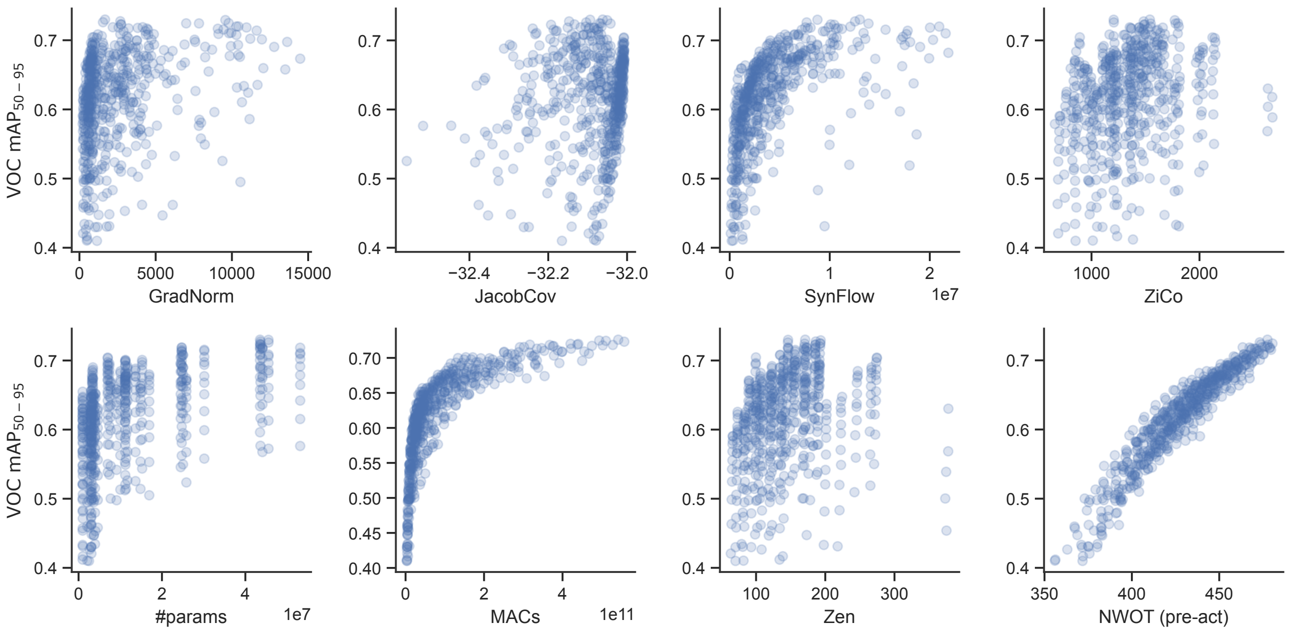

Figure S10. Scatter plots of zero-cost predictor values computed on randomly initialized *YOLOBench* models vs. mAP$_{50-95}$ of these models fine-tuned on VOC from COCO-pretrained weights.

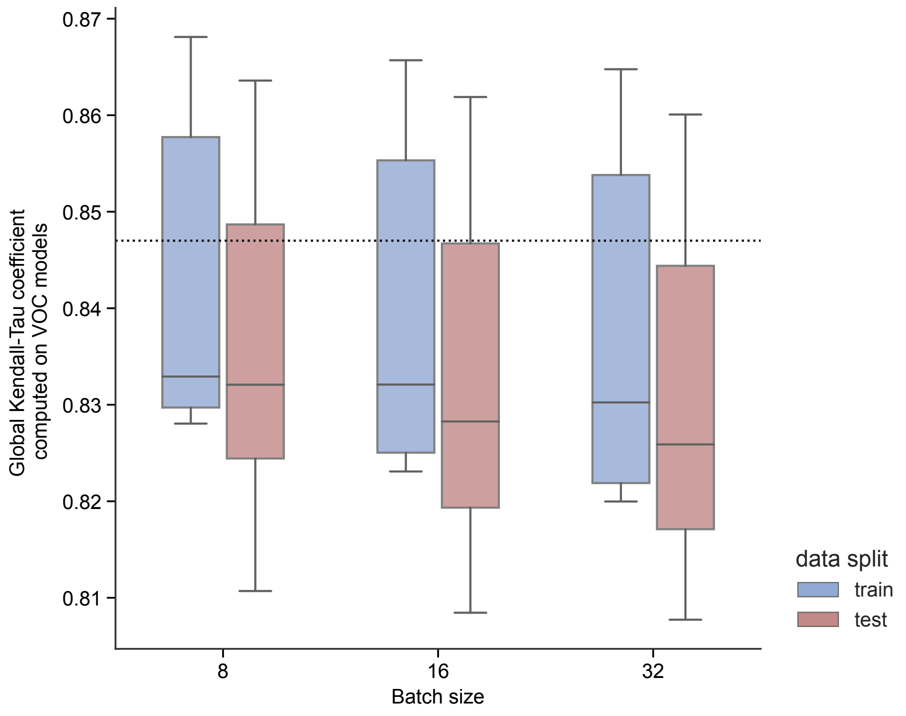

Figure S11. Robustness of pre-activation NWOT estimator in predicting mAP$_{50-95}$ values of VOC models in *YOLOBench*. Shown is Kendall-Tau score dependence on the batch size, as well as the data, split used to sample the batch (training (augmented) data vs. testing data (no augmentations)). The dotted vertical line corresponds to the performance of mAP$_{50-95}$ VOC in training from scratch used as a predictor.

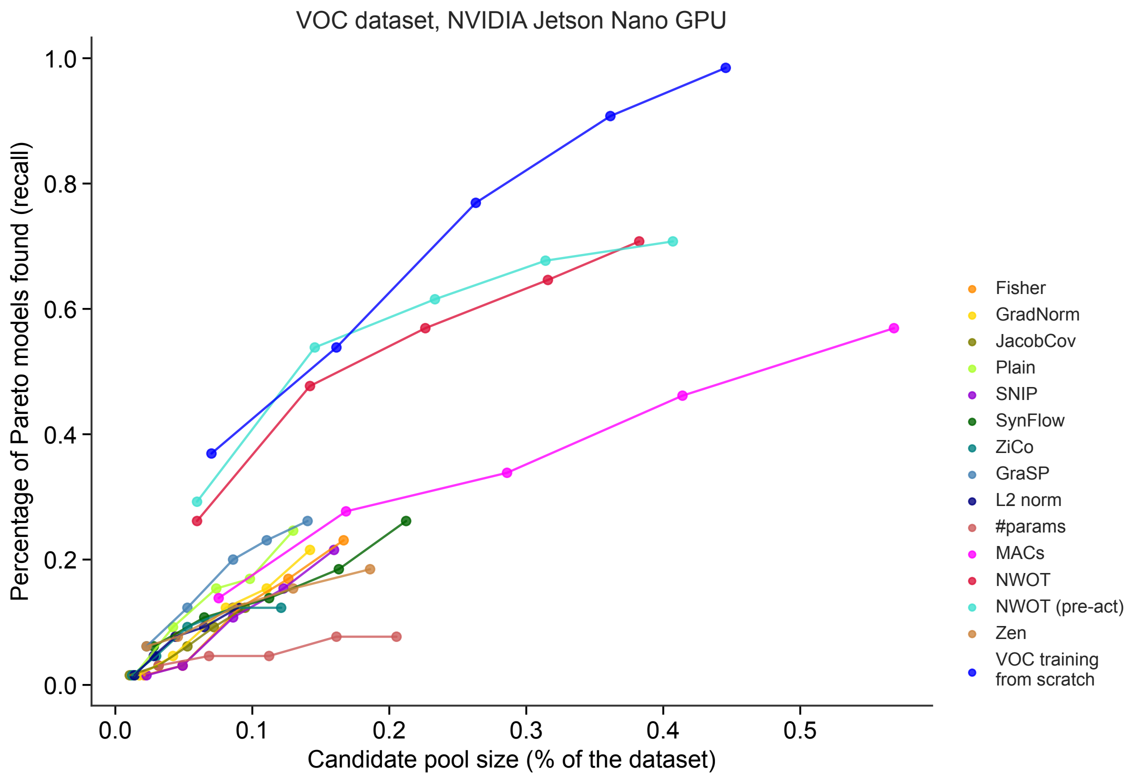

Figure S12. Percentage of all actual Pareto models (recall) found in the candidate pools consisting of first N (N from 1 to 5) ZC-based Pareto sets on the VOC dataset with latency measured on the Jetson Nano GPU (ORT, FP32). Data shown for all zero-cost estimators considered in this study, in addition to mAP$_{50-95}$ values of VOC training from scratch used as a performance predictor.

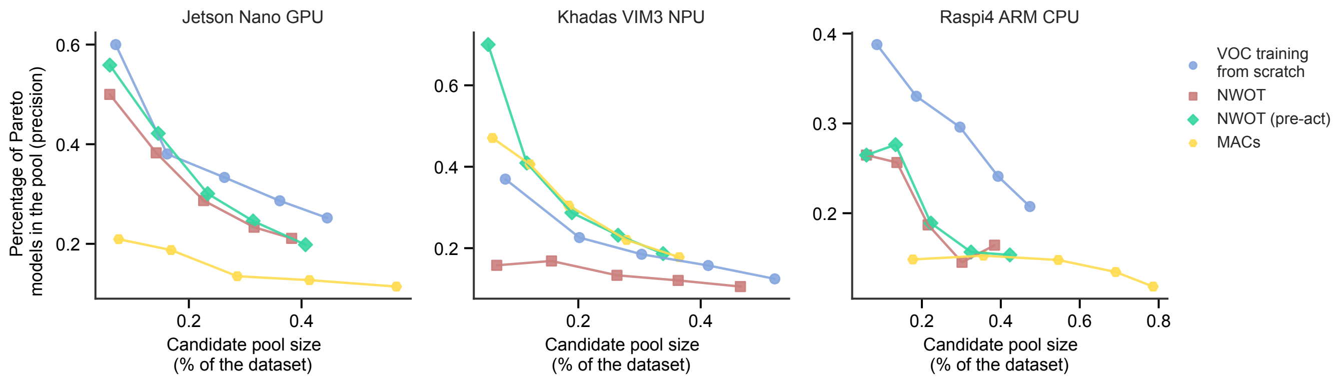

Figure S13. Percentage of Pareto models out of all models found in the candidate pools (precision) consisting of first N (N from 1 to 5) ZC-based Pareto sets on the VOC dataset depending on the hardware platform and performance predictor used.

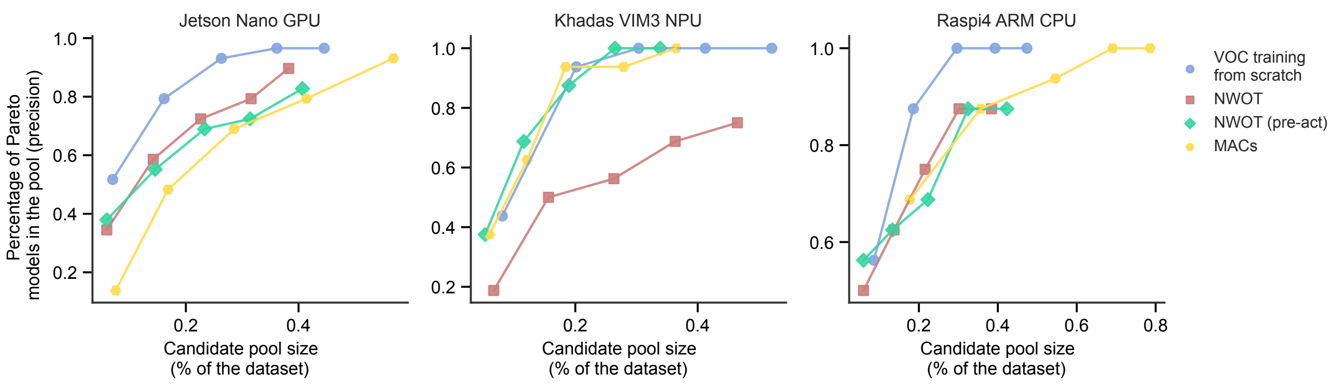

Figure S14. Percentage of all actual Pareto models (recall) found in the candidate pools consisting of first N (N from 1 to 5) ZC-based Pareto sets on the VOC dataset depending on the hardware platform and performance predictor used. Models with different input resolutions but the same architecture as treated as a single model.

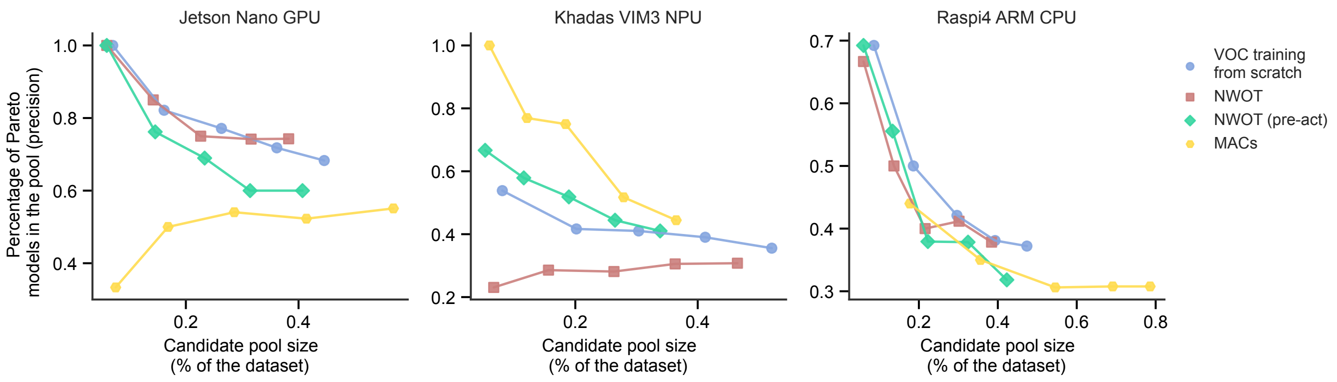

Figure S15. Percentage of Pareto models out of all models found in the candidate pools (precision) consisting of first N (N from 1 to 5) ZC-based Pareto sets on the VOC dataset depending on the hardware platform and performance predictor used. Models with different input resolutions but the same architecture as treated as a single model.

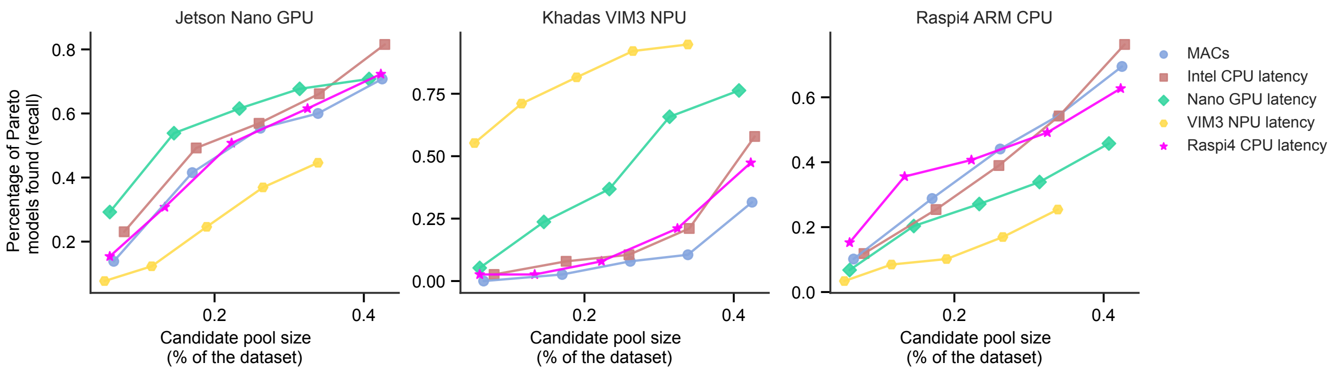

Figure S16. Percentage of all actual Pareto models (recall) found in the candidate pools consisting of first N (N from 1 to 5) ZC-based Pareto sets on the VOC dataset with pre-activation NWOT score as accuracy estimator depending on the hardware platform and latency proxy used. Aside from the actual on-device latency, other latency proxies considered are MAC count and latency on other devices.

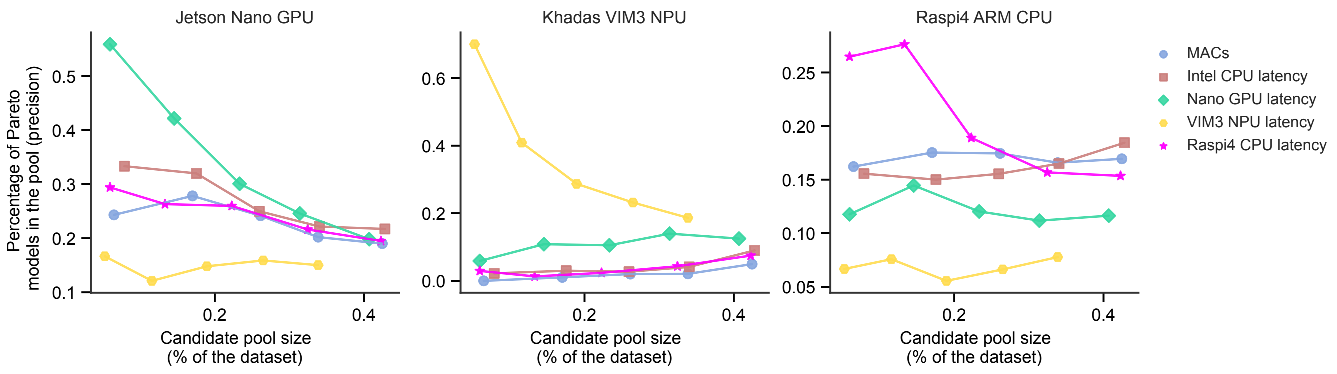

Figure S17. Percentage of Pareto models out of all models found in the candidate pools (precision) consisting of first N (N from 1 to 5) ZC-based Pareto sets on the VOC dataset with pre-activation NWOT score as accuracy estimator depending on the hardware platform and latency proxy used. Aside from the actual on-device latency, other latency proxies considered are MAC count and latency on other devices.

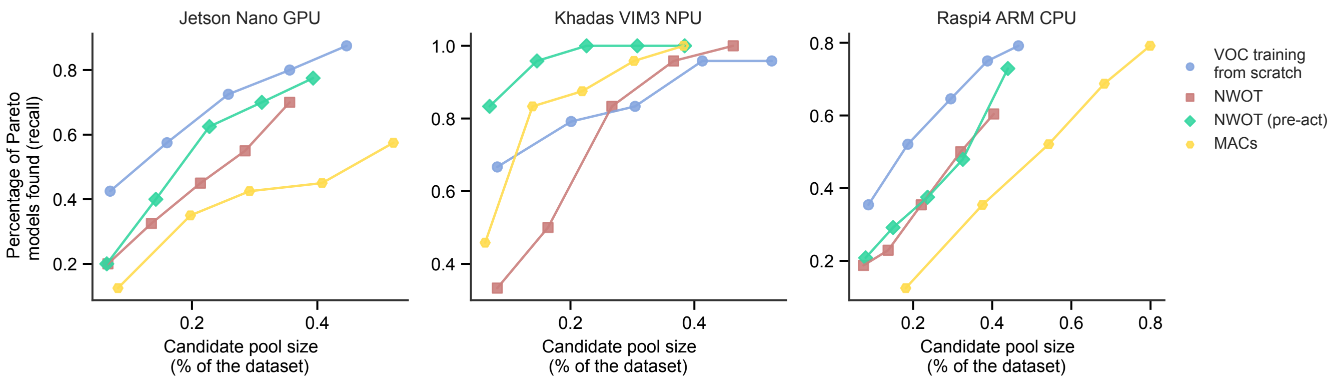

Figure S18. Percentage of all actual Pareto models (recall) found in the candidate pools consisting of first N (N from 1 to 5) ZC-based Pareto sets on the SKU-110k dataset with latency measured on the Jetson Nano GPU (ORT, FP32). Data shown for MAC count and NWOT score as performance estimators, in addition to mAP$_{50-95}$ values of VOC training from scratch used as a performance predictor on the SKU-110k dataset.