TinySegPrune: Multi-Objective Pruning for Semantic Segmentation Networks on TinyML Hardware

Zhuoran Xiong, Warren J. Gross, Brett H. Meyer
Department of Electrical and Computer Engineering, McGill University, Montreal, Canada
zhuoran.xiong@mail.mcgill.ca, {warren.gross, brett.meyer}@mcgill.ca

***Abstract*—Network pruning is critical for supporting the efficient deployment of convolutional neural networks (CNNs) on resource-limited TinyML hardware. While pruning has proven effective in many cases, semantic segmentation models remain challenging to prune for deployment to TinyML devices due to their large output feature maps and consequent memory accesses, necessitating more fine-grained compression strategies. We propose TinySegPrune: a multi-objective pruning framework that parameterizes the pruning rate of each block in the network. We employ a latency predictor that estimates latency from block-wise pruning rates for fast hardware performance evaluation. We validate the effectiveness of our framework by pruning two different segmentation networks for a TinyML processor. Our framework finds a pruned model that reduces latency by 30.3% with only 1% accuracy reduction, outperforming benchmark methods that achieve only 14.7% latency reduction—a 2.1× larger improvement. Leveraging latency feedback, our framework effectively identifies critical blocks, reducing their latency by 30% to 50%, and significantly improving overall efficiency.**

***Index Terms*—Multi-Objective Optimization, Fine-Grained Compression, Latency Prediction, Memory-Efficient CNNs**

I. Introduction

Semantic segmentation, which assigns a semantic label to each pixel in an image, is a practically important yet highly challenging task in computer vision. Improvements in deep CNNs have resulted in substantial advances within this domain [1], [2]. However, existing semantic segmentation networks typically have a large number of parameters, and therefore require substantial computational resources and memory accesses [3], [4]. This poses a challenge for efficiently deploying them on devices with limited memory, such as smart glasses with an ultra-low-power processor. For example, the GAP8 processor [5] has an on-chip L2 cache of 512 KB, whereas the peak memory usage of Mobile DeepLabV3 (DLV3), reported as a downstream application in [6], is 3.7 MB. While larger, high-latency off-chip RAM can be utilized, reliance on off-chip memory often results in latency growth that is not smooth but instead exhibits abrupt, stepwise behavior. For example, when running the Mobile DLV3 network on a GAP8 processor with 256 KB of cache and 8 MB of SDRAM, the latency of its inverted residual blocks (IRBs) increases in a stepwise manner as the number of filters grows, as illustrated in Fig. 1. We refer to this phenomenon as the 'latency ladder.'

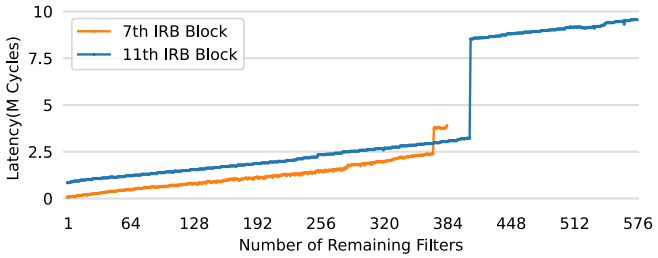

Fig. 1. Latency ladder phenomenon when pruning inverted residual blocks (IRB) with increasing number of filters. The sharpness and onset vary from block to block.

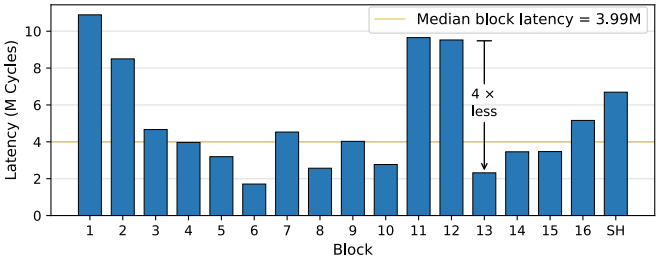

Fig. 2. Per-block latency of Mobile DeepLabV3 on GAP8 [5], including all inverted residual blocks and the segmentation head (SH).

Previous work [7], [8] has employed network pruning—a widely used model compression technique—to improve the efficiency of semantic segmentation networks on GPU or FPGA devices, but have overlooked the bottleneck effect caused by the latency ladder on TinyML processors. The varying sizes of feature maps and weights across different blocks result in substantial differences in memory footprints, which in turn cause significant latency variation across the network. Fig. 2 illustrates the per-block latency of this model on the GAP8 processor. We observe that block 12's latency is 4.1× that of block 13, despite having comparable computational complexity, indicating that off-chip memory accesses amplify the latency differences between blocks. This large inter-block latency variance implies that uniform pruning across blocks is inherently suboptimal on memory-constrained hardware.

We hypothesize that fine-grained, latency-aware pruning is

the key to finding efficient semantic segmentation networks for TinyML hardware. We propose a multi-objective, block-wise filter pruning framework that parameterizes each block's pruning rate across the backbone and segmentation head. Our framework explores the space of pruning configurations and returns models that trade off accuracy and inference latency, using a predictor to quickly estimate each candidate's latency.

In summary, this work makes the following contributions:

- we empirically observe a block-level 'latency ladder' effect on TinyML processors, where pruning triggers non-linear latency reductions due to memory-hierarchy effects;
- motivated by the latency ladder, we propose a fine-grained multi-objective block-wise pruning method that exploits such block-dependent behavior and achieves up to $2.1\times$ larger latency reduction than uniform pruning at similar accuracy;
- we expand the design space to include the segmentation heads. The resulting Pareto front dominates that which results from only pruning the backbone;
- we employ a novel predictor that efficiently and accurately estimates the latency of a pruned model based on its block-wise pruning rates. Our predictor surpasses prior state-of-the-art with a 4.8% improvement in accuracy, and $3.9\times$ inference speedup.

The code used in this work is publicly available at https://github.com/paulxiongfree/TinySegPrune.

II. Related Work

A. Efficient Semantic Segmentation Networks

Several efficient semantic segmentation networks [6], [9] have been proposed to enable deployment on resource-constrained edge devices. These methods utilize mobile-optimized backbones, such as MobileNetV2, combined with lightweight segmentation heads to achieve a favorable balance between accuracy and computational efficiency. Nevertheless, as highlighted in [10], although models optimized for mobile devices reduce resource demands, they remain poorly suited for TinyML deployments, where memory budgets are typically constrained to kilobytes or low megabytes.

Others [11]–[13] have proposed lightweight networks based on U-Net for TinyML devices, achieving reasonable performance on certain datasets. However, the U-Net architecture heavily relies on connections between the encoder and decoder, which leads to a large number of memory accesses [14]. As a result: 1) the networks are constrained to very small parameter sizes (e.g., TinySeg has only 0.10 M parameters), limiting their learning capacity; and, 2) the input and output image resolutions are restricted (e.g., TinySeg [12] uses 120×80 pixels and L3UNet [11] uses 88×88 pixels), which hampers the model's ability to learn from datasets with relatively larger image sizes. Our work uses backbones originally designed for mobile devices and adapts them to TinyML scenarios by modifying block stride to reduce the size of intermediate feature maps, thereby lowering the parameter count to fit the limited memory budget.

B. Network Pruning as Neural Architecture Search

Prior studies have explored structured pruning to remove redundant weights and connections in deep neural networks, thereby reducing model size and computational cost [15]–[18]. Some works [19]–[22] focus on pruning semantic segmentation models, rather than exclusively targeting image classification. SIRFP [23] and CAP [21] improve the task-specific effectiveness of network pruning by leveraging spatial information redundancy and contextual information, respectively. They aim to prune more parameters while minimizing the resulting accuracy drop. ACoSP [22] designs a selection mechanism for each convolutional layer based on a temperature annealing schedule. By gradually annealing the weights of less important filters to zero while retaining a selected subset of filters, it is able to maintain acceptable performance even after removing more than 93% of the parameters. However, these works are not designed with specific hardware performance in mind, but rather focus on reducing the number of remaining parameters. As will be shown in Section III-A, parameter count is not strongly correlated with inference latency, leading to suboptimal improvements in real-world hardware performance.

C. Hardware-aware network optimization

Many hardware-aware network optimization methods have been proposed [24], [25] to optimize latency and accuracy. MnasNet [24] adopts a reinforcement learning (RL) framework that uses a hardware-aware reward, formulated as a weighted sum of accuracy and latency. Instead, work [25] uses a genetic algorithm to perform a multi-objective search for deployment on FPGA devices and has been proven to be better than RL-based methods. Their method relies on a look-up-table (LUT)-based latency model, which stores the latencies of different model operations. However, on TinyML processors, a simple LUT-based lookup cannot accurately estimate the latency of an operation, as the latency depends on where the operation's inputs reside in the memory hierarchy: the same operations have different latencies depending on whether their inputs are in L1, L2, or L3 cache, or some combination therein. Our method also uses a genetic algorithm for multi-objective search, but our latency predictor is based on a gradient boosting tree model, which takes only the block-wise pruning rates of the network as input. By sampling and training on various configurations, the model learns to accurately and efficiently predict latency under different pruning rates.

III. Motivations and Preliminaries

A. Hardware Latency Information

We hypothesize that fine-grained pruning, informed by inference latency, is the key to finding efficient segmentation networks. This is motivated by the phenomenon that we call latency ladder and the fact that latency is not related to latency proxies. The latency ladder is a sharp reduction in block inference time when pruning a single block running on memory-constrained hardware, as shown in Fig. 1. As the latency ladder phenomenon emerges, we observe sudden drops in per-operation data transfer and latency, indicating a

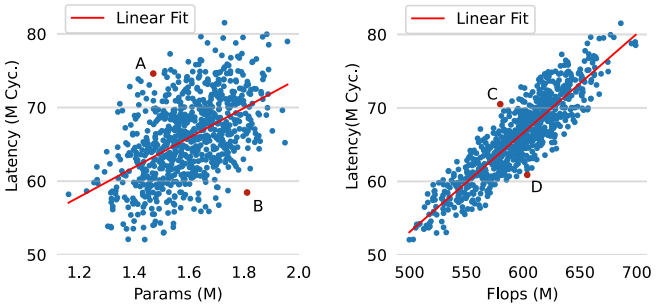

Fig. 3. Latency comparison with FLOPs and parameter count. The relationship is not strictly linear, indicating that FLOPs and parameters are imperfect proxies for latency.

significant reduction in the cost of transferring intermediate data and suggesting a qualitative shift in memory usage. The latency ladder phenomenon occurs at different pruning thresholds for different blocks in Mobile DeepLabV3. For example, in Block 7, a sharp latency ladder occurs at a pruning rate of 0.045, where the latency drops sharply by 41% (1.25 M cycles) and $3.2\times$ reduction in per-operation latency. In contrast, Block 11 experiences a less sharp latency ladder than Block 7 at a pruning rate of 0.31, resulting in a 65% latency reduction (5.5 M cycles). This block-specific behavior reveals that pruning certain blocks can lead to nonlinear gains in latency. Recognizing and exploiting these latency ladders is therefore essential for designing efficient block-wise pruning strategies on memory-constrained devices.

Due to the aforementioned effect of memory accesses, the relationship between latency proxies (e.g. FLOPs or parameters) and the hardware latency is not linear. Taking Mobile DLV3 as an example, we sampled 800 pruned networks defined by their block-wise pruning rates and measured their inference latency, parameters and FLOPs, as illustrated in Fig. 3. As the annotated points in each subgraph show, the relationship between latency and each proxy is not linear. For example, as shown in the params-latency plot, point A has 20.2% fewer parameters than point B, yet its latency is 26.2% higher. This highlights the critical need to optimize for actual latency instead of indirect proxies like FLOPs or parameter count, which may not accurately reflect real-world performance.

B. Structured Filter Pruning

Structured filter pruning is a hardware-friendly pruning strategy that removes filters from layers and speeds up the model without dedicated accelerators [15], [26]. In this work, we employ a local independent pruning scheme, where we assign an independent pruning rate to each pruned structure, either a layer or a building block, rather than a uniform rate shared across the network. Such pruning enables fine-grained control over the pruning rate at different parts of the network, which we hypothesize can lead to greater latency reduction with minimal compromise in accuracy. Assume that a CNN has K building blocks and $\mathcal{C}^i$ is the i-th block and n_i is the number of filters in $\mathcal{C}^i$. When filter pruning is performed on the i-th block, filters are divided into two groups according to an importance criterion, for example L-1 norm [27]. Let n_{i1} and n_{i2} denote the numbers of retained and pruned filters, respectively. The pruning ratio or pruning rate δ_i, which determines the proportion of filters to be pruned from block i, is:

$$\delta_i = \frac{n_{i2}}{n_i} \tag{1}$$

Then, we define a pruning rate set for a model with L blocks as:

$$\mathbf{\Delta} = \{\delta_1, \delta_2, \delta_3, \ldots, \delta_L\} \tag{2}$$

Local pruning with a uniform pruning rate, such as HRank, uses the same pruning rate in each block, but as a consequence they sacrifice the opportunity to target more memory intensive layers where it may be possible to reduce latency without substantially compromising accuracy. In contrast, we propose a framework that automatically determines the set of pruning rates in a latency-aware manner, resulting in models that significantly improve overall latency with minimal task performance overhead.

IV. Methodology

We propose a multi-objective automatic pruning framework that parameterizes block-wise pruning rates, driven by the block-level latency ladder behavior observed on memory-constrained TinyML hardware. We use NSGA-II [28] to explore the design space of the pruning rate sets for pre-trained semantic segmentation networks. The search algorithm generates candidate pruned models with their block-wise pruning rates. For each candidate, the latency predictor estimates inference latency and accuracy is evaluated on a validation set. After several generations, it produces Pareto non-dominated trade-offs between accuracy and latency. We fine-tune and deploy selected models to determine final trade-offs.

A. Pre-trained Networks

We consider two efficient segmentation networks for our TinyML application: the widely adopted Mobile DLV3 [6] and our customized MobileOne-based [9] network, which we refer to as MobileOne DLV3. We choose these two networks because GPU-oriented architectures, such as DeepLabV3+ [2], cannot fit within the constrained memory of TinyML devices, whereas the selected models employ efficient designs that are well-suited for resource-constrained hardware. Both Mobile DLV3 and MobileOne DLV3 follow the common design of semantic segmentation networks, consisting of a backbone that extracts hierarchical feature representations from the input image and a segmentation head that converts these features into dense pixel-wise predictions. Mobile DLV3 combines a MobileNetV2 [6] backbone with a reduced Atrous Spatial Pyramid Pooling (ASPP) segmentation head that uses only two branches, making it more memory-efficient than the standard five-branch ASPP in DLV3 [1]. MobileOne DLV3 also employs a reduced-ASPP segmentation head, but uses MobileOne [9] as its backbone. MobileOne adopts a multi-branch architecture during training, which is reparameterized

into an efficient single-branch network for inference without compromising accuracy. In each case, we increase the backbone output stride to 32 to reduce the spatial resolution of the backbone feature maps to $1/32$ of the input image.

B. Pruning Strategy

We adopt a block-wise structured pruning strategy that removes filters from convolutional layers at the granularity of *blocks* [26], [29], where each block is a group of layers that are repeated (e.g., the inverted residual block in MobileNetV2). We decide to prune at the granularity of blocks because the layers within the same building block follow the same computational topology. In addition, modern networks are often very deep, frequently exceeding fifty layers, which would lead to an excessively large search space and hinder convergence if pruning rates are assigned at the individual layer levels.

For the MobileNetV2 backbone, we adopt the pruning scheme proposed in [26] to prune the inverted residual bottleneck blocks. Specifically, we remove depth-wise filters and their corresponding point-wise filters in the inverted residual bottleneck blocks. For the MobileOne backbone, we prune both the training network and its reparameterized inference network. For the training network, we prune depth-wise convolution filters for each MobileOne block according to the pruning rate and remove the corresponding channels in all branches. Then, the pruned training network is reparameterized into a single-branch inference network for deployment.

In the reduced ASPP segmentation head, we propose a pruning method tailored to its multi-branch architecture. We first prune the convolutional layers in both branches of the atrous spatial pyramid, and then further prune the projection layer that takes the concatenated outputs from these branches. The information about the unpruned filters in each branch is propagated to the projection layer, ensuring that its input channels and corresponding weights are pruned consistently.

We predefine a set of candidate pruning rate values for each block to be pruned. Drawing inspiration from MCUNetV2 [29], which varied the expansion factor from 3 to 6 to search for sub-networks, our available pruning rates for each block are $\{0, 1/6, 2/6, 3/6\}$ in both MobileNetV2 and MobileOne backbone. This helps prevent the exponential growth of the search space. For the segmentation head, which consists of only three layers, we adopt a finer-grained set of pruning rates, from $\{9/16, 10/16, \ldots, 1\}$, ensuring that the minimum pruning step of 16 corresponds to a power-of-two granularity to improve hardware parallelism.

C. Latency Predictor

We propose an inference latency predictor that estimates hardware latency based on the model's pruning rate set $\mathbf{\Delta}$, enabling fast latency feedback during the search process. We use a gradient boosting tree regressor, inspired by [30]. Our regressor takes $\mathbf{\Delta}$ as input and outputs the network's latency. To train and validate our approach, we randomly sample 800 pruning rate sets from the design space and record the corresponding pruned models' latencies on TinyML hardware. This dataset is used for hyperparameter tuning via grid search and for training the final predictor. We then evaluate on a disjoint test set of another 800 randomly sampled pruned models. For the GAP8, the final configuration for Mobile DLV3's latency predictor was: `max_depth=3`, `learning_rate=0.12`, `n_estimators=250`, and `subsample=0.6`. For MobileOne DLV3, the only difference in the configuration is `max_depth=6`. We use the XGBoost squared-error objective, and the predictor input is the full block-wise pruning-rate vector.

TABLE I
PERFORMANCE OF OUR MOBILE DEEPLABV3 LATENCY PREDICTOR ON THE TEST DATASET.

Method	Accuracy ($\pm 5\%$)	MAPE	ADRS	Speed
BRP-NAS [31]	85.6%	8.8	10.3	16.3ms
LitePred [32]	87.1%	7.3	8.7	8.2ms
Ours	90.4%	4.2	5.2	4.1ms

We sampled an additional 800 pruned models and measured their accuracy and latency on the target hardware to construct a test set. We compared our predictor with the GCN-based and CNN-based latency predictor used in BRP-NAS [31] and LitePred [32], evaluating their performance in terms of prediction accuracy within a $\pm 5\%$ error bound, mean absolute percentage error (MAPE), average distance from the reference set (ADRS) [33] distance to the ground-truth accuracy-latency Pareto front, and prediction speed. As shown in Table I, our tree-based latency predictor for Mobile DLV3 achieves better prediction accuracy across multiple metrics. It also significantly outperforms benchmarks in prediction speed.

D. Genetic Algorithm Search

In our NSGA-II formulation, the chromosome $\mathbf{C}$ of each individual i in the genetic algorithm's population encodes $\mathbf{\Delta}_i$ (Eq. (2)). Given a baseline model with L blocks, the initial population consists of: (i) The unpruned model, equivalent to a pruning rate set where all pruning rates are zero. (ii) L minimally pruned models. The l-th pruning rate set Δ_l is defined as:

$$\Delta_l = \{\delta_1^l, \delta_2^l, \ldots, \delta_l^l, \ldots, \delta_L^l\}, 1 \leq l \leq L \tag{3}$$

where δ_l^l is set to the smallest available non-zero pruning rate, while all other pruning rates are set to 0. (iii) Models with uniformly sampled pruning rate sets. The second group directs the search toward models that are minimally pruned, as less pruned models tend to have higher accuracy and users are often very sensitive to accuracy degradation when performing network pruning.

The fitness function evaluates and orders the pruned networks (chromosomes) in the population based on their accuracy and latency. Accuracy is measured using mean Intersection over Union (mIoU): the overlap between the predicted segmentation and the ground truth, divided by the total area covered by both. During search, each candidate is briefly fine-

tuned for 5 epochs [29] and then evaluated on the validation set, while latency is estimated by our predictor.

Our NSGA-II implementation uses a population size of 40. Chromosomes are encoded in binary, with each gene specifying one of the discrete pruning choices for a block. The initial population consists of the unpruned model, the minimally pruned models described above, and additional uniformly sampled pruning-rate sets to fill the remaining slots. We use half-uniform crossover and bit-flip mutation [34], and rank the combined parent–offspring population using non-dominated sorting and crowding distance [28].

At the end of each generation, the framework checks for convergence of the Pareto non-dominated front. Searching for efficient semantic segmentation networks is known to be time-consuming [35]. Therefore, we define the termination condition as follows: if the Pareto non-dominated front shows no improvement within 3% of the maximum mIoU obtained for 5 consecutive generations, we terminate our search. The threshold of 3% mIoU degradation is acceptable for semantic image segmentation [6]. If the termination criterion is not yet met, crossover and mutation are applied to the population to generate new candidate solutions. This process repeats until termination.

After obtaining the Pareto non-dominated set, we further fine-tune the 10 models with the highest mIoU, given user sensitivity to accuracy degradation. For the final comparison in Section V, these selected models are fine-tuned for 100 epochs using the same learning-rate rewinding strategy [36]; this resets the learning rate to the initial learning rate of the pre-trained unpruned model to help models escape local performance minima and recover better.

V. Experiments and Results

A. Experimental Setup

We conducted experiments on the Pascal VOC semantic image segmentation dataset [37], a widely used benchmark that provides annotated images across 21 classes. Unlike autonomous-driving datasets, which often contain high-resolution images in specialized scenarios, Pascal VOC offers diverse scenes and object categories, making it broadly applicable to various TinyML applications. The original images vary in resolution and can be rescaled to meet different application requirements; in our experiments, we downsampled them to 256 $\times$ 256 pixels to satisfy the memory constraints and resolution limits typical of TinyML platforms.

We evaluated model latency on the GreenWaves GAP8 [5] processor, an ultra-low-power high-performance RISC-V-based processor, with its default Autotiler network scheduler that can provide detailed memory allocation information. It has L1 and L2 cache of 50 KB and 256 KB, respectively, and L3 off-chip memory of 8 MB.

All methods are evaluated using the same dataset split, image resolution, validation metric, and hardware measurement pipeline. Where applicable, we also use the same training and fine-tuning protocol across methods.

TABLE II
Performance comparison of TinySegPrune Pruned Models and baselines

Model	FLOPs (M)	Params (M)	mIoU (%)	Lat. (M)	Speedup
Mobile DLV3 [6]	776	2.12	69.52	96.91	-
RL-Pruner (R) [17]	708	1.71	68.39	83.90	1.15×
HRank (H) [16]	732	1.76	68.45	82.70	1.17×
Global (G) [15]	718	1.68	68.31	91.41	1.06×
Isomorphic (S) [18]	713	1.72	67.88	86.90	1.11×
M1	722	1.95	**69.22**	81.94	1.18×
M2	682	1.75	68.47	67.57	1.43×
M3	672	1.64	68.11	**62.39**	**1.55**×
MobileOne DLV3 [9]	532	1.34	69.80	58.92	-
RL-Pruner (L)	503	1.16	69.25	53.60	1.10×
HRank (I)	506	1.28	69.05	53.08	1.11×
Global (J)	510	1.12	69.16	56.07	1.05×
Isomorphic (C)	508	1.14	68.71	50.50	1.17×
O1	501	1.29	**69.67**	55.08	1.07×
O2	472	1.12	69.32	47.75	1.23×
O3	443	1.01	69.13	**45.97**	**1.31**×

We compare the performance of our proposed pruning framework with (a) pruning using a reinforcement learning algorithm, RL-Pruner [17], (b) block-wise pruning with a uniform pruning rate, HRank [16], (c) global pruning [15], and (d) isomorphic pruning [18]. We calculate the mean intersection over union (mIoU) that is averaged across the 21 classes to evaluate the accuracy of each pruned model. During our preliminary experiments with uniform block-wise pruning and global pruning, we found that when the maximum pruning rates were set to $8/48$ and $8/40$, the mIoU of models obtained from these two pruning methods decreased to about 66.5%, a reduction of approximately 3%. Therefore, dividing the pruning rates of $8/48$ and $8/40$ into eight equal parts, we derived the comparison models for each approach. For HRank, the uniform block-wise pruning rate is selected from $\{1/48, 2/48, \ldots, 8/48\}$. For global pruning, the global-wise pruning rate is selected from $\{1/40, 2/40, \ldots, 8/40\}$. Therefore, we compare methods under matched accuracy budgets rather than nominal pruning ratios alone. For each baseline family, we report representative points whose post-fine-tuning accuracies are closest to those of our selected models.

B. Results

We adopt an early stopping strategy by monitoring the convergence of high-accuracy models (e.g. mIoU $> 60\%$) on the Pareto front. If no improvement is observed within 5 epochs, the search is terminated to trade off between search time and solution quality. From the 25th to the 30th generation, there was no further improvement on the higher accuracy side of the Pareto non-dominated front for Mobile DLV3, with all of the improvements occurring in the low-accuracy part of the curve. Thus, we terminated our search after the 30th generation. The 10 highest-accuracy models were selected for fine-tuning. For MobileOne DLV3, we followed the same procedure and ended the search at the 34th generation. The

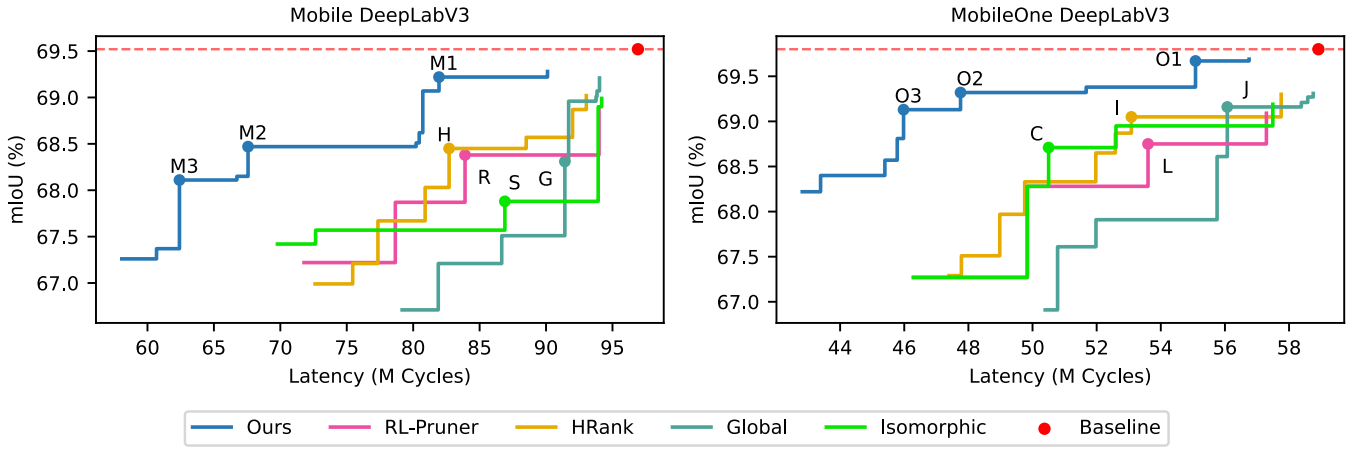

Fig. 4. Comparison of pruned model vs. benchmark methods. The Pareto front produced by our method dominates that of the comparison approaches.

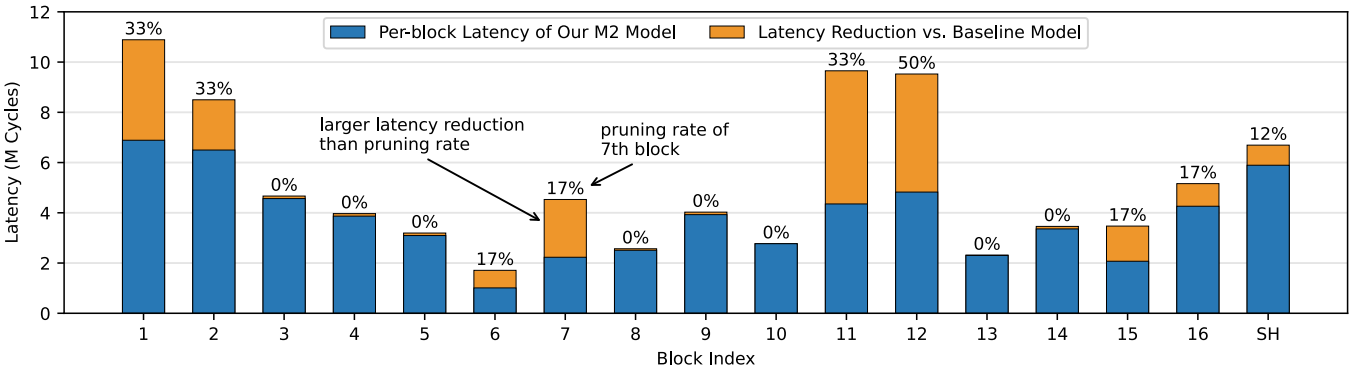

Fig. 5. Per block latency of M2 and latency reduction compared with the baseline. Notably, blocks 1, 7, and 11 have enormous latency reduction that exceeds the proportional reduction in pruning rate. Numbers above bars indicate block-wise pruning rates.

proposed framework converges in a practical amount of time, requiring approximately 118 hours for Mobile DLV3 and 115 hours for MobileOne DLV3 in our experimental setup.

Fig. 4 shows the final Pareto non-dominated fronts after 100 epochs of fine-tuning. An ideal design lies in the top-left corner of the plot. For Mobile DLV3 (left), our proposed approach consistently outperforms the models generated by HRank pruning and the models from global pruning. A similar trend is observed for MobileOne DLV3 (right), where models produced by comparison strategies are again dominated by our method.

Table II reports a selection of models for comparison. Overall, our multi-objective block-wise pruning framework consistently discovers models with substantially lower latency than all competing methods at comparable accuracy levels. All comparison points are produced using the same training, hyperparameter tuning, and retraining pipeline, with closely matched final mIoU. Achieving slightly better than 68.5% mIoU, pruned networks derived from HRank, Global pruning, and other comparison strategies (e.g., H, G, R, and S) achieve modest latency reductions over the baseline, whereas our M2 model attains substantially larger reductions. Specifically, models H (derived from HRank) and G (derived from Global pruning) run 82.70 M cycles (14.7% faster than baseline) and 91.41 M cycles (5.7% faster), respectively. However, our M2 model has a latency of 67.57 M cycles (30.3% latency reduction). That is 2.1× more latency reduction than H. A similar trend is observed on MobileOne DLV3, where O3 achieves a higher mIoU than 69% while reducing latency by 21.2% (12.95 M), which is 2.1× greater than I's 9.9%. These results indicate that our approach consistently dominates existing pruning strategies in the accuracy–latency trade-off.

Fig. 5 illustrates the block-wise latency and pruning rate of M2. M2 has a higher latency reduction at blocks 1, 2, 7, 11, 12 and fewer reductions at other blocks. This confirms our hypothesis that we can achieve better acceleration by using fine-grained pruning, which adjusts per-block pruning rates. Furthermore, note that while the pruning rate of block 7 in the M2 model is 0.17, latency in that block is reduced more than 50%. We observe that the off-chip memory accesses for block 7 decreased from 254.2 KB to 38.5 KB. At the same time, the on-chip memory footprint shrank from 532 KB to 436 KB. Considering that off-chip accesses typically incur 5–20× higher latency than on-chip accesses, we can confidently state that the 85% reduction in off-chip accesses is the key driver of latency improvement. This shows that

fine-grained pruning enables a more latency-efficient memory mapping for this block. Similarly, blocks 1, 11, and 12 showed more than 33% latency reduction. This shows the importance of hardware feedback: by identifying blocks where pruning can substantially reduce off-chip memory accesses, our framework lowers latency on this memory-constrained device better than would be otherwise possible; FLOPs and parameter count are not reliable proxies for latency. This demonstrates two key observations outlined in Section III-A. First, fine-grained pruning can push the pruning rate beyond the threshold required to trigger steps in the latency ladder, leading to significantly large reductions in latency. Second, since pruning can substantially reduce off-chip memory accesses, FLOPs and parameter count—as traditional proxies for computational cost—are not reliable indicators of real latency. This also validates our framework's ability to identify blocks where more aggressive pruning amplifies latency reduction benefits, resulting in superior latency gains on memory-constrained hardware.

In addition to the GAP8 platform, we further deploy our M2 model on two representative embedded systems: the STM32F769 and the Raspberry Pi 4, and compare its performance with that of Model H. The STM32F769, equipped with an ARM Cortex-M7 processor and 512 KB of on-chip L2 cache, represents a low-power microcontroller-class device. The Raspberry Pi 4, on the other hand, integrates a Cortex-A7 processor with 2 MB of L2 cache, representing a more powerful edge-class processor. Experimental results on the STM32F769 show that the M2 model reduces inference latency from the baseline's 230 ms to 173 ms (25% reduction), outperforming the H model, which achieves 193 ms (15% reduction). This latency improvement on STM32F769 is accompanied by a reduction in energy consumption from 73.5 mJ to 60.7 mJ (17% reduction), indicating improved overall efficiency. On the Raspberry Pi 4, the M2 model reduces inference latency from 168 ms to 131 ms (22% reduction), compared to 151 ms (10% reduction) achieved by the H model. These results demonstrate that our model delivers substantial latency reductions with only minor accuracy degradation on resource-constrained hardware.

C. Ablation Studies

1) Pruning Segmentation Head: xuyao

To evaluate the impact of pruning the segmentation head, we compare against a baseline that prunes only the backbone while keeping the segmentation head intact. As illustrated in Fig. 6, the final Pareto non-dominated front achieved by our method (blue line) clearly dominates that of the backbone-only pruning approach (purple line). Achieving slightly better than 68% mIoU, X run 66.89 M cycles (31.0% faster than baseline), but at the same mIoU level, our M3 model has a latency of 62.39 M cycles (35.6% faster).

2) Initial Population Design: We performed another ablation study with two alternative initial population designs. The first consists of individuals with randomly sampled per-block pruning rates. The second replaces randomly sampled individuals with manually designed pruning rate sets, obtained by setting every δ_l^l in Eq. 3 equal to 0.33 and all other pruning rates to 0. Fig. 6 compares the three initial population designs using the same search settings and fine-tuning procedure as before. We observe that the random initial population (brown line) is to the bottom left of our approach (blue line). The maximum and minimum mIoU and latency are both smaller: this initial population results in searches that tend to explore models with lower latency and accuracy. Using additional manually designed initial individuals (orange line) leads to similar maximum latency and mIoU (still dominated by our M1), but the minimum accuracy is higher and minimum latency is larger: while it also directs the search to higher-latency models, it has not searched enough for lower-latency models.

VI. Conclusions

This work focuses on uncovering and exploiting hardware-induced latency phenomena on memory-constrained TinyML processors. Building on this insight, we present a multi-objective block-wise pruning framework for semantic segmentation networks. By parameterizing the pruning rate of each block and using a multi-objective genetic search algorithm to explore the search space of block-wise pruning rates based on hardware feedback and mIoU, our approach finds Pareto-optimal models that outperform benchmark pruning strategies. A representative model achieves 30.5% lower inference latency with less than 1% mIoU drop. This latency reduction is substantially larger than the corresponding reduction in parameter count, a consequence of reducing the number of costly off-chip memory accesses. The latency predictor enables efficient hardware-aware optimization with superior accuracy and speedup. Finally, pruning the segmentation head extends our framework to semantic segmentation and yields Pareto-optimal models beyond backbone-only pruning.

A full network-level memory-traffic analysis for all compared models would be valuable, but is beyond the scope of the current revision and is left for future work. Future work may explore joint optimization of pruning, architecture, and memory allocation to explicitly control transitions across latency ladder thresholds. One possible way to facilitate such joint optimization, beyond the specific networks studied in this paper, is to generalize block-wise pruning rates into parameterized functions over network parameters, enabling the framework to be applied to a broader range of architectures. In addition, the investigation of the TinyML application specific to the practical dataset remains an important research direction.

References

[1] L.-C. Chen, G. Papandreou, F. Schroff, and H. Adam, "Rethinking atrous convolution for semantic image segmentation," *arXiv preprint arXiv:1706.05587*, 2017.

[2] L.-C. Chen, Y. Zhu, G. Papandreou, F. Schroff, and H. Adam, "Encoder-Decoder with atrous separable convolution for semantic image segmentation," 2018, p. 833–851.

[3] J. Wang *et al.*, "Deep high-resolution representation learning for visual recognition," *IEEE Transactions on Pattern Analysis and Machine Intelligence*, vol. 43, no. 10, pp. 3349–3364, 2021.

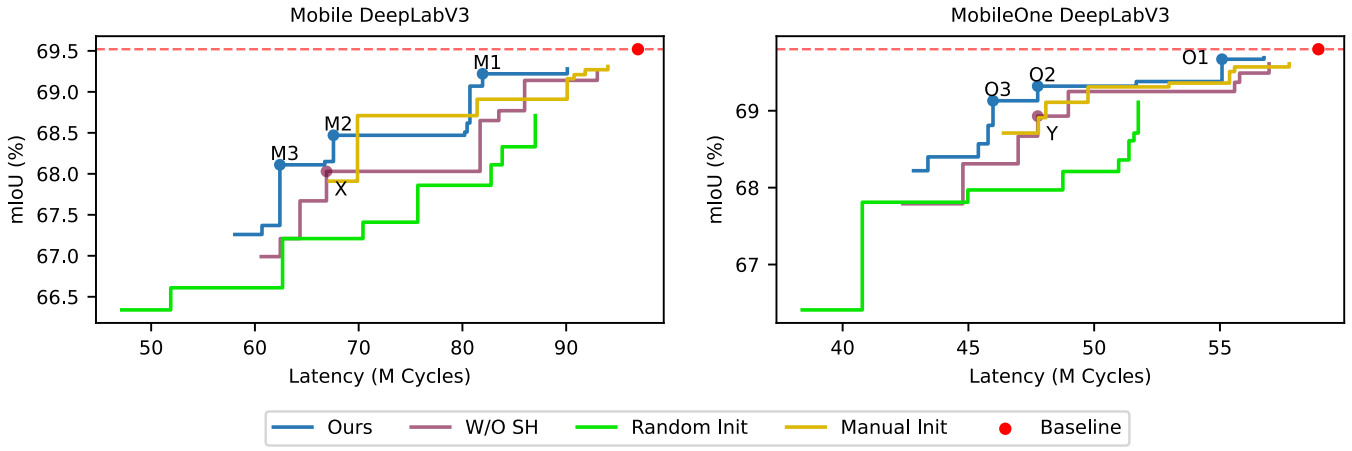

Fig. 6. Ablation study: compare with 1) search without segmentation head; 2) different initial populations. Our method yields a more favorable Pareto front.

[4] Z. Zhong *et al.*, "Squeeze-and-Attention networks for semantic segmentation," in *2020 IEEE/CVF Conference on Computer Vision and Pattern Recognition (CVPR)*, 2020, pp. 13 062–13 071.

[5] E. Flamand, D. Rossi, F. Conti, I. Loi, A. Pullini, F. Rotenberg, and L. Benini, "GAP-8: A RISC-V SoC for AI at the edge of the IoT," in *2018 IEEE 29th International Conference on Application-specific Systems, Architectures and Processors (ASAP)*, 2018, pp. 1–4.

[6] M. Sandler *et al.*, "MobileNetV2: Inverted residuals and linear bottlenecks," in *CVPR*, 2018, pp. 4510–4520.

[7] C. Park *et al.*, "Deprune: depth-wise separable convolution pruning for maximizing GPU parallelism," in *Proceedings of the 38th International Conference on Neural Information Processing Systems*, 2024.

[8] P. Morì *et al.*, "Accelerating and pruning CNNs for semantic segmentation on FPGA," in *Proceedings of the 59th ACM/IEEE Design Automation Conference*, ser. DAC '22, 2022, p. 145–150.

[9] P. K. A. Vasu *et al.*, "MobileOne: An improved one millisecond mobile backbone," in *2023 IEEE/CVF Conference on Computer Vision and Pattern Recognition (CVPR)*, 2023, pp. 7907–7917.

[10] J. Lin, L. Zhu, W.-M. Chen, W.-C. Wang, and S. Han, "Tiny machine learning: Progress and futures [feature]," *IEEE Circuits and Systems Magazine*, vol. 23, no. 3, pp. 8–34, 2023.

[11] O. E. Okman *et al.*, "L^3U-net: Low-latency lightweight U-Net based image segmentation model for parallel cnn processors," *CoRR*, vol. abs/2203.16528, 2022.

[12] B. Chae, J. Kim, and S. Heo, "Tinyseg: Model optimizing framework for image segmentation on tiny embedded systems," in *Proceedings of the 25th ACM SIGPLAN/SIGBED International Conference on Languages, Compilers, and Tools for Embedded Systems*, 2024, p. 23–33.

[13] Z. Xiong, W. J. Gross, and B. H. Meyer, "Zero-shot nas for tinyml semantic segmentation via weight sharing," *IEEE Embedded Systems Letters*, pp. 1–1, 2026.

[14] B. Chae and S. Heo, "Tinymo: Graph-level memory optimizer for tiny machine learning," *IEEE Embedded Systems Letters*, vol. 17, no. 3, pp. 196–199, 2025.

[15] G. Fang, X. Ma, M. Song, M. Bi Mi, and X. Wang, "Depgraph: Towards any structural pruning," in *2023 IEEE/CVF Conference on Computer Vision and Pattern Recognition (CVPR)*, 2023, pp. 16 091–16 101.

[16] M. Lin *et al.*, "HRank: Filter pruning using high-rank feature map," in *CVPR*, 2020, pp. 1526–1535.

[17] B. Wang and V. Kindratenko, "RL-pruner: Structured pruning using reinforcement learning for cnn compression and acceleration," *arXiv Preprint arXiv:2411.06463*, 2024.

[18] G. Fang *et al.*, "Isomorphic pruning for vision models," in *Proceedings of the 18th European Conference on Computer Vision*, 2024, p. 232–250.

[19] D. Wu *et al.*, "Structural pruning via spatial-aware information redundancy for semantic segmentation," in *AAAI-25*, 2025.

[20] X. Chen, Y. Zhang, and Y. Wang, "MTP: multi-task pruning for efficient semantic segmentation networks," in *IEEE ICME, 2022*.

[21] W. He, M. Wu, M. Liang, and S.-K. Lam, "CAP: Context-aware pruning for semantic segmentation," in *Proceedings of the IEEE/CVF Winter Conference on Applications of Computer Vision (WACV)*, January 2021.

[22] K. Ditschuneit and J. S. Otterbach, "Auto-compressing subset pruning for semantic image segmentation," in *Pattern Recognition*, B. Andres, F. Bernard, D. Cremers, S. Frintrop, B. Goldlücke, and I. Ihrke, Eds. Cham: Springer International Publishing, 2022, pp. 20–35.

[23] W. Wen, C. Wu, Y. Wang, Y. Chen, and H. Li, "Learning structured sparsity in deep neural networks," in *Proceedings of the 30th International Conference on Neural Information Processing Systems*, ser. NIPS'16. Red Hook, NY, USA: Curran Associates Inc., 2016, p. 2082–2090.

[24] M. Tan *et al.*, "Mnasnet: Platform-aware neural architecture search for mobile," in *2019 IEEE/CVF Conference on Computer Vision and Pattern Recognition (CVPR)*, 2019, pp. 2815–2823.

[25] P. Morì *et al.*, "Accelerating and pruning CNNs for semantic segmentation on FPGA," in *Proceedings of the 59th ACM/IEEE Design Automation Conference*, 2022, p. 145–150.

[26] C.-H. Tu, J.-H. Lee, Y.-M. Chan, and C.-S. Chen, "Pruning depthwise separable convolutions for mobilenet compression," in *2020 International Joint Conference on Neural Networks (IJCNN)*, 2020.

[27] S. Han, J. Pool, J. Tran, and W. J. Dally, "Learning both weights and connections for efficient neural networks," in *Proceedings of the 28th International Conference on Neural Information Processing Systems - Volume 1*, ser. NIPS'15, 2015, p. 1135–1143.

[28] K. Deb, A. Pratap, S. Agarwal, and T. Meyarivan, "A fast and elitist multiobjective genetic algorithm: NSGA-II," *IEEE Transactions on Evolutionary Computation*, vol. 6, no. 2, pp. 182–197, 2002.

[29] J. Lin, W.-M. Chen, H. Cai, C. Gan, and S. Han, "MCUNetV2: memory-efficient patch-based inference for tiny deep learning," in *NeurIPS*, 2024.

[30] N. Firouzian *et al.*, "Work-in-Progress: Utilizing latency and accuracy predictors for efficient hardware-aware NAS," in *2022 International Conference on Hardware/Software Codesign and System Synthesis*, 2022, pp. 15–16.

[31] L. Dudziak *et al.*, "BRP-NAS: prediction-based nas using gcns," in *Proceedings of the 34th International Conference on Neural Information Processing Systems*, ser. NIPS '20, 2020.

[32] C. Feng *et al.*, "Litepred: transferable and scalable latency prediction for hardware-aware neural architecture search," in *Proceedings of the 21st USENIX Symposium on Networked Systems Design and Implementation*, 2024.

[33] G. Palermo, C. Silvano, and V. Zaccaria, "Discrete particle swarm optimization for multi-objective design space exploration," in *2008 11th EUROMICRO Conference on Digital System Design Architectures, Methods and Tools*, 2008, pp. 641–644.

[34] J. Blank and K. Deb, "pymoo: Multi-objective optimization in python," *IEEE Access*, vol. 8, pp. 89 497–89 509, 2020.

[35] C. Liu *et al.*, "Auto-DeepLab: Hierarchical neural architecture search for semantic image segmentation," in *2019 IEEE/CVF Conference on Computer Vision and Pattern Recognition (CVPR)*, 2019, pp. 82–92.

[36] A. Renda, J. Frankle, and M. Carbin, "Comparing rewinding and fine-tuning in neural network pruning," in *8th International Conference on Learning Representations, ICLR*, 2020.

[37] M. Everingham *et al.*, "The pascal visual object classes (voc) challenge." *Int. J. Comput. Vis.*, vol. 88, no. 2, pp. 303–338, 2010.