Enabling ImageNet-Scale Deep Learning on MCUs for Accurate and Efficient Inference

Sulaiman Sadiq, Jonathon Hare, Simon Craske, Partha Maji, Geoff Merrett

***Abstract*—Conventional approaches to TinyML achieve high accuracy by deploying the largest deep learning model with highest input resolutions that fit within the size constraints imposed by the microcontroller's (MCUs) fast internal storage and memory. In this paper, we perform an in-depth analysis of prior works to show that models derived within these constraints suffer from low accuracy and, surprisingly, high latency. We propose an alternative approach that enables the deployment of efficient models with low inference latency, but free from the constraints of internal memory. We take a holistic view of typical MCU architectures, and utilise plentiful but slower external memories to relax internal storage and memory constraints. To avoid the lower speed of external memory impacting inference latency, we build on the TinyOps inference framework, which performs operation partitioning and uses overlays via DMA, to accelerate the latency. Using insights from our study, we deploy efficient models from the TinyOps design space onto a range of embedded MCUs achieving record performance on TinyML ImageNet classification with up to 6.7% higher accuracy and 1.4x faster latency compared to state-of-the-art internal memory approaches.**

***Index Terms*—Edge AI, Internet of Things (IoT), Artificial Intelligence of Things (AIoT), TinyML, Microcontrollers, Embedded Systems, Machine Learning, Deep Learning, Neural Networks (NNs), Convolutional Neural Networks (CNNs).**

I. Introduction

Deep Neural Networks (DNN) have found success in a variety of fields including Image Classification [1], [2], Natural Language Understanding [3], [4] and Gameplay [5] amongst several others. At the same time, Internet of Things (IoT) devices are increasingly pervading society with forecasts that there will be 29.4 billion connected IoT devices by 2030 [6]. Typically, DNN inference on IoT devices is applied with the traditional cloud computing paradigm where data collected on the IoT device is processed in the cloud [7]. However, issues of reliability, privacy, latency and cost associated with cloud based solutions [8] has led to the development of the field of Tiny Machine Learning (TinyML) which aims to develop models and frameworks suitable for processing data locally on IoT devices without streaming the raw data to the cloud. This is desirable for applications including smart homes [9], health analytics [10] and industrial sensing [11].

Corresponding Author: Sulaiman Sadiq
Sulaiman Sadiq, Jonathon Hare and Geoff Merrett are with the Department of Electronics and Computer Science, University of Southampton, UK (email: sulaiman.sadiq@soton.ac.uk, jsh2@ecs.soton.ac.uk, gvm@ecs.soton.ac.uk)
Simon Craske is with ARM Ltd, UK (email: simon.craske@arm.com)
Partha Maji is with Tenstorrent, UK (email: pmaji@tenstorrent.com)

A significant challenge in TinyML is the severe resource constraints of the IoT devices. While modern GPUs have memory in the range of gigabytes and thousands of cores for parallel processing of DNN workloads, IoT devices host microcontrollers (MCUs), with limited memory in the order of kilobytes, a small or even no cache, and a single core for processing. Considering these constraints and the trend of model sizes getting larger [12], efficient inference on MCUs requires optimising across the inference stack by designing efficient models and inference software that effectively utilises the devices resources to maximise its compute capability.

The majority of approaches [13]–[17] to DNN deployment in TinyML take a simplified view of MCU architectures where they consider only limited internal storage and memory which have low power consumption and access latency. However, these works do not consider the limitations on model performance imposed by size constraints of this internal memory design space. In this article, we analyse state-of-the-art models derived from the constrained internal memory design space to show that they suffer from low accuracy and, somewhat counter-intuitively, high latency as well. This suggests the need for an alternative approach, free from internal memory size constraints. While larger external memories are available, they have lower performance and energy efficiency for inference, as they are slower and have higher power consumption. To effectively utilise external memories, we build on the TinyOps inference framework [18], which accelerates inference from external memory to combine advantages of speed of internal memory and design flexibility of external memories using operation partitioning and DMA based overlaying. We perform experiments on ARM® Cortex® M class devices and utilise insights from our study to derive efficient models from the TinyOps design space demonstrating that it outperforms prior approaches in terms of accuracy, latency and energy efficiency as shown in Figure 1.

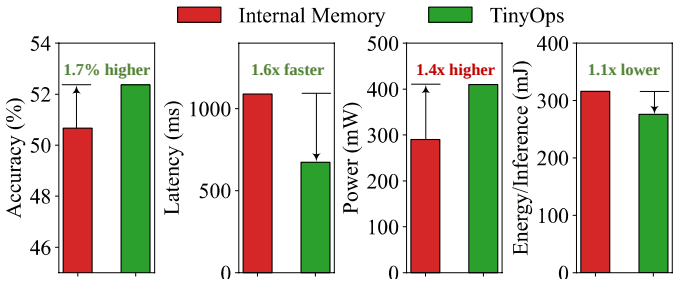

Fig. 1. The TinyOps design space achieves higher accuracy with lower latency and energy efficiency than the internal memory design space.

The main contributions of this work are as follows:

- We study DNN deployment on MCUs, and show that deriving models within the constraints of the internal memory design space leads to deployment of degenerate architectures with low accuracy and high computation focused in inefficient operations leading to high latency.
- We build on the TinyOps inference framework [18] to accelerate inference from external memory by up to 2x, enabling a new efficient design space for tiny deep learning on MCUs.
- We derive efficient models from the TinyOps design space for a range of devices, with up to 6.7% higher accuracy and 1.4-2.1x faster inference latency than the previous state-of-the-art [13], [17].

Our analysis demonstrates the strength of the TinyOps design space and suggests that it should be studied further for tiny deep learning applications on MCUs.

II. Background and Related Work

Due to the limited compute capabilites of microcontrollers, prior works have focused on improving efficiency across a number of different dimensions.

a) Quantisation: A common approach to reducing the memory footprint of models is reducing the precision of weights and activations. Varying fixed point precisions have been shown to work including 2-bit, 4-bit, 8-bit or mixed-bit [15], [19]–[21], in addition to vector quantisation approaches [22], [23]. In the case of MCUs, we use the industry standard 8-bit quantisation due to the DSP/SIMD instructions of the architectures that enable faster throughput.

b) Pruning: Another approach is to prune less important connections in a network to induce sparsity. [24]–[27]. These approaches reduce the memory footprint of the model allowing deployment within tight memory constraints. However, unstructured pruning approaches tend to not carry over to microcontrollers due to lack of efficient sparse matrix manipulation kernels. In our study, we use the conventional approach of scaling existing mobile models by treating the width and resolution as hyper-parameters which are selected to meet memory and storage requirements [15], [28].

c) Efficient Network Design: For maximum performance, architectures can be hand-crafted for low memory and computational footprint [28]–[30]. Recently Neural Architecture Search has also been used with latency-aware [31]–[33] and resource-aware search [13], [14], [17], [34]. We analyse these works in Section III to find that models derived with these operation agnostic approaches under internal storage and memory constraints yield sub-optimal accuracy and latency.

d) Frameworks and Kernels: Microcontrollers have a number of frameworks for DNN inference [13], [35]–[38]. The frameworks use optimised kernels to efficiently execute operations in the inference graph [13], [15], [39], [40]. However, these works either use external or internal memory exclusively for weights or tensors, yielding sub-optimal accuracy or latency [13], [35]–[37], or utilise platform specific peripherals which limits portability [38] while others only support a subset of DNN architectures [39].

In our work, we build on the TinyOps [18] inference framework which utilises the open-source TensorflowLite-Micro (TFLiteMicro) [35] with CMSIS-NN [40] kernels. We adopt a generic model of the MCU which allows portability across devices. We experiment with ARM® Cortex® M class devices and utilise external memory interfaces including the Flexible Memory Controller (FMC) and Octo/Quad Serial Port Interface (O/QSPI) to supplement the limited internal memory and storage with SDRAM and NOR Flash respectively as shown in Figure 2. Further, we utilise the DMA peripheral to provide data transfers between memories with minimal CPU intervention.

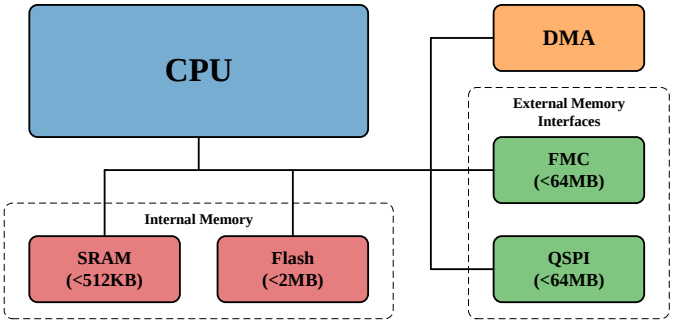

Fig. 2. Block Diagram of Typical MCU Architecture.

III. Design Limitations of Internal Memory

We analysed models designed for internal memory and studied their accuracy, computational complexity (MACs) and inference latency. Following the setup of prior works, we studied scalings of mobile models (ProxylessNAS [41], MNASNet [32], MobileNetV3 [42], EfficientNet [30]) in addition to the NAS based MCUNetV1/V2 models [13], [17]. When scaling mobile models, the storage and memory requirement dictated by the parameter count and tensor sizes were controlled via the number of filter channels or width of the model and input resolution respectively. We used 8-bit precision and scaled down from baseline models by varying the width in increments of 0.05 from 0.10 to 1.00, and resolution in increments of 16 ranging from 48 to 224. We looked at ARM® Cortex® M class MCU devices by STMicroelectronics with diverse internal storage and memory constraints including the L552 (512KB Flash/192KB SRAM), F469 (1MB/256KB), F746 (1MB/320KB) and the H743 (2MB/512KB). A mobile model scaled to a width of 0.10 and a resolution of 192, is referred to as {ProxylessNAS/MNASNet/MobileNetV3}-w0.10-r192. Models derived with the MCUNetV1/V2 algorithms for any device with 4 or 8-bit precision are referred to as MCUNet{V1/V2}-{F469/F746}-int{4/8}, e.g. an 8-bit model derived for the F746 would be MCUNetV1-F746-int8.

A. Below the Pareto Frontier

We determined the pareto frontier for ImageNet classification by training scaled variations of mobile models free from any constraints by varying the width and resolution and compared it with models derived under internal storage and memory constraints of the MCU platforms.

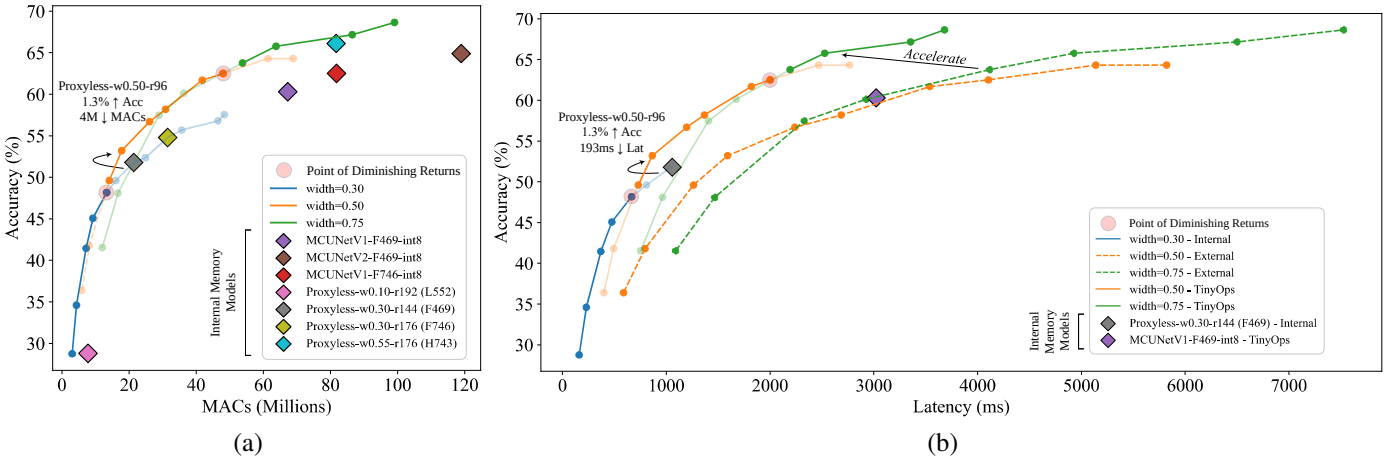

Fig. 3. **(a)** The Accuracy-MACs pareto curve is composed of models with different widths or parameters. The platforms which the ProxylessNAS scalings were derived for are given in parentheses. **(b)** The Accuracy-Latency pareto frontier is pushed forward by the proposed TinyOps inference framework which accelerates inference of models in external memory as described further in Section IV.

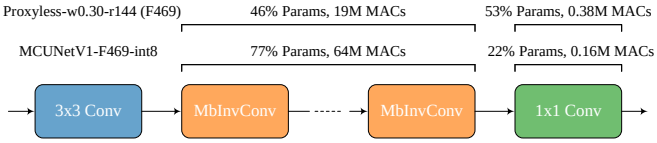

Fig. 4. MCUNet architectures employ more parameters in compute intensive MobileInvertedConv modules to achieve higher accuracy at the cost of extra computation.

Our analysis revealed that models designed within the constraints fell well below the pareto frontier as shown in Figure 3a. As expected, we found the constraint on model size and input resolution imposed by the internal memory design space limited the achievable accuracy. However, we further observed the internal memory models to have a significantly higher computational complexity compared to models on the pareto frontier achieving the same accuracy. This is explained by looking at how prior works attempt to achieve maximum accuracy and observing the range of resolutions that are optimal for a particular width multiplier imposed by internal storage constraints. We observed that, when increasing the resolution for any particular width multiplier, there is an input resolution till which that width multiplier is optimal. We term this the *point of diminishing returns* as increasing the resolution past this point increases the computation significantly while returning a diminished increase in accuracy. As shown in Figure 3a, it is sub-optimal to keep on using the width multiplier past this point as a higher width scaling on the pareto frontier is able to achieve better accuracy with the same computation. However, in order to achieve the highest accuracy, prior works utilise the largest width and highest input resolution that can be accomodated by internal storage and memory. For a range of diverse internal memory constraints of MCU devices, it can be observed that this approach leads to deployment of scalings situated well past the point of diminishing returns which fall below the pareto curve.

Similary, we studied the MCUNet [13], [17] family of models derived via NAS for deployment within internal storage and memory constraints to find that they also fell short of the pareto frontier as shown in Figure 3a. The MCUNet framework achieves higher accuracy by maximising computation (MACs) in the models which is taken as a proxy for accuracy. As shown in Figure 4, we found that this is accomplished by lowering the parameters in the storage intensive classification (Fully Connected) layer and reusing the saved internal storage in compute intensive MobileInvertedConv blocks in the architectures. Indeed, we found that the change to micro-architecture significantly increased computation in the model. However, this approach also returned a diminished increase in accuracy for the amount of computation involved with the models falling below the pareto curve.

In either case, we found that the internal memory models could be outperformed by larger models free from internal memory size constraints on the pareto frontier. However, these would require larger external memory for deployment resulting in high inference latency due to slow memory access, even though models had low computational complexity as observed by comparing Figure 3a and 3b. We build on the TinyOps inference framework [18] to accelerate latency of inference from external memory while retaining the design flexibility of the design space. This enables us to advance the pareto curve as shown in Figure 3b and deploy models from the TinyOps design space that outperform internal memory models.

B. Sub-Optimal Deployment Latency

We further analysed the models by studying the operations that composed the model, including convolution (Conv) and depthwise convolutions (DepConv). We benchmarked the throughput of the operations in addition to looking at the distribution of computation across the operations in models derived via scaling or NAS. For models derived from the internal memory design space using either approach, we found that, in addition to achieving limited accuracy with high computational requirements as discussed in Section III-A, the computation was focused in inefficient operations leading to high deployment latency.

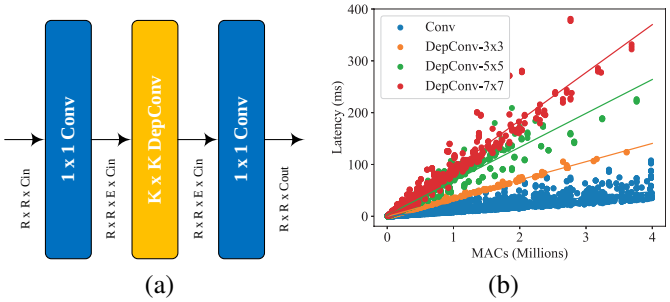

Fig. 5. **(a)** Structure of MobileInvertedConv block. **(b)** Latency of operations of varying complexity used in MobileInvertedConv blocks.

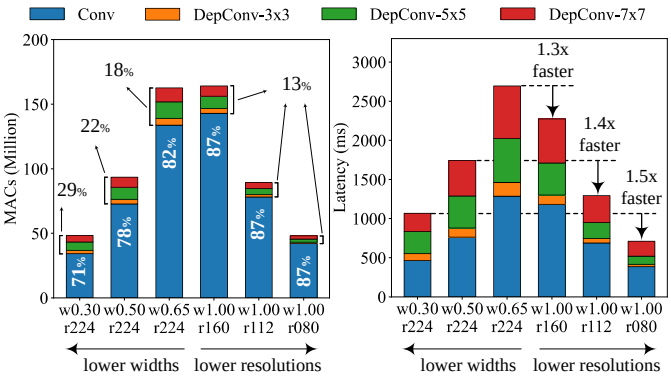

Fig. 6. Computation (MACs) and Latency Distribution of operations in models of varying complexity derived by scaling down width or resolution.

While prior approaches use the conventional approach of scaling to meet memory or computational constraints, we looked at how scaling different parameters of the model affected inference latency on MCU devices. We analysed the effect of uniformly scaling width and resolution on the distribution of computation in the model to find that scaling down width skewed the distribution to concentrate computation in DepConv operations. This is explained by looking at the structure of the MobileInvertedConv module which is the building block of state of the art mobile models [13], [17], [30], [41], [42]. These blocks with varying parameters are cascaded together to construct the models and comprise 95% of the computation in the network. As shown in Figure 5a, these blocks consist of sequential Conv-DepConv-Conv operations where the Conv operations with 1x1 kernel size are used to project representations to a higher or lower number of channels and the DepConv operations with varying filter sizes perform feature extraction.

We looked at how computation is scaled in operations within these blocks when the model is scaled down to meet any constraint. For some width and resolution multipliers, $\alpha, \beta < 1$ the computation can be calculated as below

$$MACs_{conv1} = \boldsymbol{\alpha^2} \times \boldsymbol{\beta^2} \times R^2 E C_{in}^2 \quad (1)$$

$$MACs_{depconv} = \boldsymbol{\alpha} \times \boldsymbol{\beta^2} \times R^2 E C_{in} K^2 \quad (2)$$

$$MACs_{conv2} = \boldsymbol{\alpha^2} \times \boldsymbol{\beta^2} \times R^2 E C_{in} C_{out} \quad (3)$$

where R is the input resolution, E is the expansion ratio and C_{in} and C_{out} are the input and output channels respectively of the block.

It can be observed that lowering the width through a uniform width multiplier, α, quadratically decreases computation by α^2 in Conv operations, while decreasing only linearly by α in DepConv operations. This skews the distribution of computation with a higher percentage of computation performed in DepConvs as shown in Figure 6 where progressively reducing the width from 1.00 to 0.30 increases computation in DepConvs from 18% to 29%. On the other hand, we observe in Equations 1-3 that meeting any particular MAC budget by lowering resolution by β results in a quadratic reduction in computation by β^2 in either Conv or DepConv operations. This preserves the distribution of computation across operations as shown in Figure 6 with a constant 13% in DepConvs with the remaining 87% carried out in Conv operations.

We benchmarked the throughput of operations (1x1 Conv, 3x3, 5x5 and 7x7 DepConv) of varying complexity sampled from scaled variations of mobile models. As shown in Figure 5b, we found that DepConv operations are less efficient than Conv operations, with 3x3 DepConvs being the most efficient within DepConv operations of different filter sizes. Combined with the insight that reducing width focuses computation in DepConvs, we found that lower width scalings have high inference latency compared to higher width scalings under the same MAC budget, as shown in Figure 6 (Right). However, focusing only on internal storage forces the usage of a low width multiplier leading to high latency in addition to low accuracy as discussed in Section III-A.

We note that, although our study of the structure of the MobileInvertedConv block revealed that higher width multipliers would yield lower latency, it would be possible for these to be sub-optimal when considering accuracy. This can be observed in Figure 3a and 3b where, for lower accuracy ranges, the pareto frontier is composed of lower width scalings with the higher width scaling of 0.75 only being pareto superior in models using $>$ 52M MACs. As such, a naive approach that simply selected the highest width would be sub-optimal and achieving efficient performance for any MAC budget would require finding the right trade-off between width and resolution.

We observed a similar limitation of NAS based MCUNet architectures which lead to high latency. As the NAS algorithm maximises the computation in the model in an operation agnostic manner, a significant amount of the computation is concentrated in 5x5 and 7x7 DepConvs, which we observed to have low throughput as shown in Figure 5b. This leads to higher inference latency compared to models with the same or higher MAC budget which utilise only efficient operations.

C. Discussion

We demonstrated that models derived under internal memory constraints through conventional scaling or NAS fell below the pareto frontier. Further, we found that lower width scalings of mobile models derived through uniform scaling suffered from high latency due to the structure of the MobileInvertedConv module. It is possible the limitation of uniformly

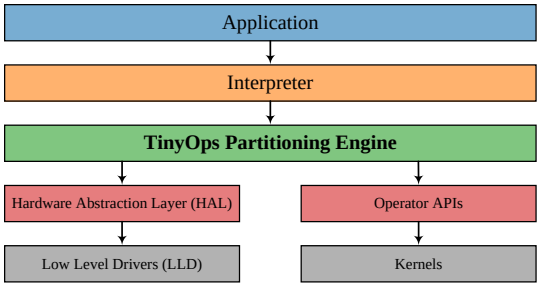

Fig. 7. TinyOps partitioning engine transparently integrates with the interpreter and low-level kernels and device drivers.

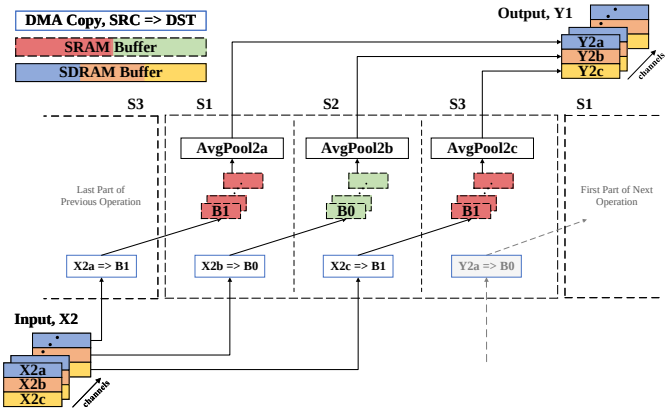

Fig. 8. TinyOps partitioning and overlaying based inference pipeline.

scaling down width yielding high latency may not hold true for different search spaces. As such, this limitation could possibly be addressed through search space engineering with structural modifications to the MobileInvertedConv module which preserved the distribution of computation in the model under uniform scaling. Indeed, our work suggests this might yield better latency. However, this could also have adverse effects on other dimensions of performance such as accuracy. Similarly, for NAS based approaches [13], [37], expanding the search space within the size constraints of internal memory by including hyper-parameters used by other works [33], [43] such as dilation factor, stride or network topology, might lead to discovery of efficient architectures. However, the representational capacity would still be limited by the number of parameters which might limit achievable accuracy. In either case, different search spaces would require extensive experimentation to evaluate and compare against the pareto frontier of the mobile search space which has been used by prior works to demonstrate state-of-the-art performance on a number of tasks on MCUs [13], [14], [37], [44].

In our work, we make use of insights gained from our study and use larger external memories to deploy balanced scalings of the MobileNetV3 model, employing only efficient 1x1 Conv and 3x3 DepConv operations. As we discuss further in the next section, we build on the partitioning and overlaying approach of the TinyOps framework [18] to accelerate inference from external memory. This allows us to outperform internal memory models as we demonstrate with our experiments in Section V. We note that if internal memory sizes were to increase in the future allowing deployment of pareto optimal models within internal memory size constraints, the TinyOps approach would be costlier from a latency perspective due to overheads of the overlaying approach. However, as we discuss in Section V-C, internal memory can be expensive to manufacture, in which case, TinyOps could be used to lower bill of materials with only a slight latency overhead (1.1x↑) compared to internal memory. Further, we believe the trend of increasing model sizes [12] for better performance also suggests the need for external memory based solutions.

IV. TinyOps: An Alternative Approach for Deep Learning on MCUs

Our analysis of the internal memory design space revealed that the constraints on model size lead to deployment with sub-optimal accuracy and latency. In this section, we explore an alternative approach for deep learning on MCUs. We propose relaxing the internal storage and memory constraints by using larger external memories. This gives us the flexibility to deploy higher width models from the pareto frontier. However, the slow access latency of external alternatives becomes a bottleneck in computation as the CPU stalls while the data used in computation, such as intermediate tensors and weights, is fetched from slower external memory. This leads to higher inference latency, even for models which have lower computational complexity than internal memory models. To mitigate the high inference latency, we build on the TinyOps framework which accelerates inference from external memory. The design flexibility of the external memory design space coupled with the TinyOps inference framework allow us to deploy efficient models with high accuracy and low latency as can be observed in Figure 3b.

The TinyOps framework utilises a partitioning and DMA based overlaying engine that seamlessly integrates into the inference software stack as shown in Figure 7. When an application invokes an inference, the interpreter runs through the inference graph and sequentially invokes the operations (Conv, DepConv, Add, Pool etc) in the graph. TinyOps executes the operations by interfacing with the low-level kernels and DMA through hooks to perform platform independent execution of the operations. In our work, TinyOps uses the interpreter based TfLiteMicro, but we note that the layered design does not limit applicability to other inference frameworks.

A. Execution Pipeline

Prior works [13], [14], [16], [17] use the entirety of internal storage (Flash) and memory (SRAM) for model parameters and activation tensors. TinyOps uses external memory (SDRAM) as main memory for data including intermediate tensors and weights of operations, to relax size constraints of limited Flash and SRAM (Figure 2). We make use of two key insights to prevent the CPU waiting on external memory fetches and accelerate inference of efficient models in the external memory design space. Firstly, that computation on a tensor can be divided such that the sub-computation performed on smaller blocks of the tensor are independent of each other. Secondly, that the CPU is able to process data

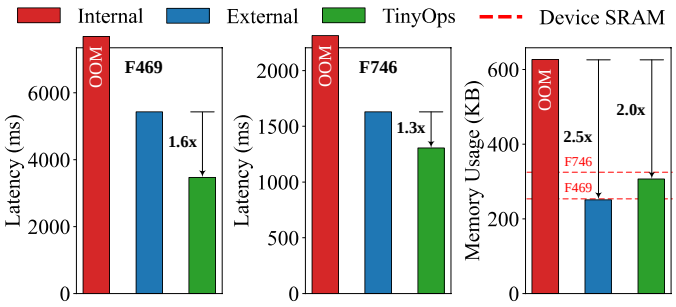

Fig. 9. TinyOps adapts the partitioning strategy according to the available memory budget when accelerating inference.

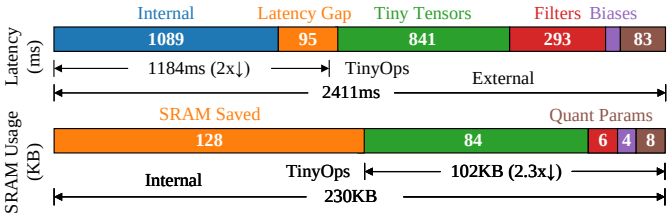

Fig. 10. TinyOps overlays frequently accessed data to reduce latency.

while the DMA copies data in parallel. Using these insights, we first reduce the memory requirement of large models by partitioning operations in the inference graph into tiny operations, which sequentially process the smaller independent blocks or tiny tensors of the input tensor. This allows data for tiny operations to be fetched from SDRAM into smaller fast buffers that can be accommodated in SRAM which the CPU accesses with low latency. The latency of data fetching from external memory is then overlapped with the computation performed by the CPU. This is accomplished by offloading the data fetching to the DMA and using a double buffering strategy to establish a pipeline between the DMA and CPU at the tiny operation level where the data copying is performed by the DMA, while the CPU processes data in SRAM buffers in parallel. This allows us to efficiently hide the latency of fetching data from external memory and enables the CPU to seamlessly process data in external memory with low latency from fast internal memory.

This is shown in Figure 8 where the input tensor, X2, of the Average Pooling operation is divided into three tiny operations. While the CPU processes the first tiny tensor, X2a, in the SRAM buffer, B1, the DMA copies in the second tiny tensor, X2b, from SDRAM to B0 in parallel to hide the data copying latency. After processing the tiny tensor in B1, the CPU frees the buffer and processes the second tiny tensor, X2b, in B0 which has been readied by the DMA which begins to load the third tiny tensor, X2c, into the newly freed buffer, B1, by the CPU and so on. In this manner, the CPU sequentially executes the tiny operations until the entire operation has been performed without ever having to wait for data to be fetched from the slower SDRAM. The tensors stored in HWC (Height-Width-Channel) format were partitioned in the H dimension since this allowed the DMA to copy contiguous blocks of data.

TinyOps uses an adaptive partitioning strategy according to the available memory budget. The size of the fast buffers on SRAM is determined by the largest tiny operation which is partitioned to meet the budget. The remaining operations were then partitioned to fit within the allocated fast buffers. This minimised overheads of multiple kernel calls, DMA initialisation, cache maintenance and interrupt handling operations. As shown in Figure 9, for a MobileNetV3-w1.00-r160 model requiring 4.2MB of storage and 627KB of memory, a memory budget of 256KB or 320KB for the F469 or F746 devices can be met while accelerating inference up to 1.6x.

B. Overlaying Strategy

To reduce SRAM usage, TinyOps only overlayed frequently accessed data. This included quantisation parameters in addition to filters and bias weights to further reduce inference latency as shown in Figure 10 where the memory usage is as low as 102KB for a Proxyless-w0.30-r144 model on the F469 while accelerating inference by 2x compared to external memory inference. Sparsely accessed data such as output tensors, which are written to only once for multiple reads of inputs and weights, were not overlayed as shown in Figure 8. Similarly, tensor structures and meta-data of the inference graph were placed in SDRAM. In total, TinyOps declared seven fast buffers on SRAM, with four used for ping-pong buffering of tiny tensors, one for biases and two for quantisation parameters. For tiny input tensors, four buffers were required to support two-input operations (Add). As two of these buffers would be unused in one-input parametric operations (Conv, DepConv), we reused these for overlaying filters and the Im2Col scratch buffer required by Conv and DepConv operations. In our work, we modified the buffer allocation strategy such that any unused memory in the memory budget was allocated to the buffer reused for filter overlaying. This allowed overlaying of more filters in the network and reduced latency up to 10%. Bias and quantisation parameter buffer sizes were chosen according to the operation with the largest requirement.

V. Experimental Results

We studied the accuracy, latency and energy efficiency of models in the internal and TinyOps design space on ImageNet classification. Experiments were performed on ARM® Cortex® M class devices using off-the-shelf Discovery and Evaluation boards by STMicroelectronics which supplemented internal storage and memory as shown in Table I. The F469 and F746 were supplemented with 8MB of SDRAM and 8MB Flash on the FMC and QSPI interface respectively, while the L552 used 2MB of SRAM on the FMC interface and a 32GB SD Card for non-volatile storage. Accuracies are reported with standard post training 8-bit quantisation, unless stated otherwise. Due to unavailability of latency metrics for MCUNetV1 8-bit models [13] with the MCUNet inference framework, these were deployed with TinyOps. Statistics for MCUNetV1 4-bit models [13] and MCUNetV2 [17] were taken from the authors' paper, since models were unavailable. Power measurements were made with the Qoitech® Otii Arch.

TABLE I
CORTEX-M BASED OFF-THE-SHELF PLATFORMS USED WITH VARYING SPECIFICATIONS AND CONSTRAINTS.

Platform	Architecture	Core	Clock (MHz)	Internal			External	
				D-Cache	SRAM (KB)	Flash (KB)	FMC (KB)	O/QSPI (KB)
L552	ARMv8	M33	110	✗	192	512	2048	-
F469	ARMv7	M4	180	✗	256	1024	8192	8192
F746	ARMv7	M7	216	4KB	320	1024	8192	8192

TABLE II
ACCURACY, LATENCY AND ENERGY PER INFERENCE COMPARISON OF MODELS FROM TINYOPS AND INTERNAL MEMORY DESIGN SPACE

Platform	Model	Design Space	MACs (M)	Params (M)	QAT	Acc (%)	Latency (ms)	Power (mW)	Energy (mJ)
L552	Proxyless-w0.10-r192	Internal	8	0.19	N	27.01	855	**65**	55.6
	MobileNetV3-w0.25-r080	**TinyOps**	3	0.50	N	**30.00**	**230**	145	**33.4**
F469	Proxyless-w0.30-r144	Internal	21	0.72	N	50.67	1089	**290**	316
	MobileNetV3-w0.50-r112	**TinyOps**	16	1.33	N	**52.37**	**674**	410	**276**
	MCUNetV1-F469-int8 [13] (Repr.)	Internal	67	0.73	N	59.47	2980	**360**	1073
	MNASNet-w1.00-r080	TinyOps	48	4.38	N	60.83	2146	435	933
	MobileNetV3-w0.75-r128	**TinyOps**	44	2.49	N	**62.58**	**1442**	395	**570**
	MCUNetV1-F469-int4 [13]	Internal	135	1.4	Y	62.00	-	-	-
	MCUNetV2-F469 [17]	Internal	119	<1	Y	64.90	-	-	-
	MobileNetV3-w1.00-r160	**TinyOps**	111	3.96	N	**68.19**	3472	405	1406
F746	Proxyless-w0.30-r176	Internal	32	0.72	N	53.68	686	**645**	442
	MobileNetV3-w0.55-r128	**TinyOps**	28	1.55	N	**58.29**	**460**	805	**370**
	MCUNetV1-F746-int8 [13] (Repr.)	Internal	82	0.74	N	61.47	1838	**765**	1406
	MCUNetV1-F746-int4 [13]	Internal	170	1.4	Y	63.50	-	-	-
	MNASNet-w1.00-r128	TinyOps	104	4.38	N	68.01	1367	775	1059
	MobileNetV3-w1.00-r160	**TinyOps**	111	3.96	N	**68.19**	**1307**	795	**1038**

A. Accuracy and Latency

Utilising insights gained from our analysis of operation throughput on MCUs and the pareto frontier, we manually derived balanced scalings of the MobileNetV3 model from the TinyOps design space to compare with models designed for internal memory.

As shown in Table II, the MobileNetV3 scalings outperformed optimal scalings of a ProxylessNAS model derived for internal memory constraints of the L552, F469 and F746 in addition to the MCUNetV1/2 models derived via NAS on the F469 and F746. On the F746, we outperformed MCUNetV1-F746-int8 with 6.4% higher accuracy and 1.4x lower latency with the MobileNetV3-w1.00-r160 model. We observed that even with 8M and 29M higher MACs than MNASNet-w1.00-r80 and MCUNetV1-F746-int8 respectively, the MobileNetV3-w1.00-r160 model had lower latency due to computation in more efficient 1x1 Conv and 3x3 DepConv operations as shown in Figure 11. On the F469, the MobileNetV3-w0.75-r128 architecture outperformed MCUNetV1-F469-int8 with 3% higher accuracy and 23M lower MACs to achieve 2.1x lower inference latency.

As latency metrics for MCUNetV2 and 4-bit MCUNetV1 models were not published, we compared with accuracy and MACs for these models. As can be observed, on the F469 and F746, the MobileNetV3-w1.00-r160 scaling achieved 4.7% and 6.2% higher accuracy with 24M and 49M lower MACs than MCUNetV1-F469-int4 and MCUNetV1-F746-int4 respectively. Similarly, compared to MCUNetV2-F469-int8, the MobileNetV3-w1.00-r160 yielded 4% higher accuracy with 8M lower MACs without the need to perform quantisation aware training (QAT). We note that as MCUNetV2 uses a NAS algorithm similar to MCUNetV1 to derive models, which maximises computation in an operation agnostic manner, our study suggests MCUNetV2-F469-int8 would have significantly higher latency than the MobileNetV3-w1.00-r160 model even under the same MAC budget due to the usage of inefficient 5x5 and 7x7 DepConv operations similar to the observation made for MCUNetV1-F746-int8 in Figure 11.

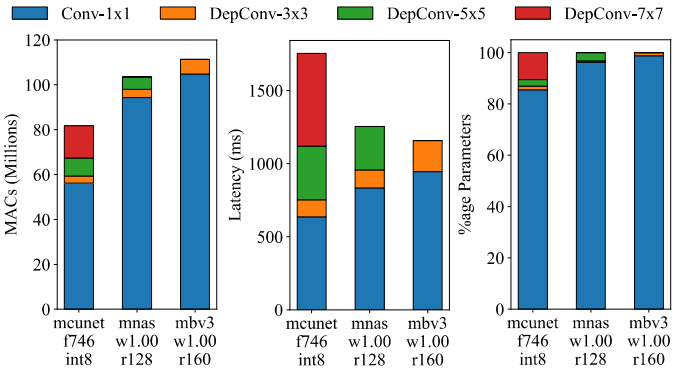

Fig. 11. The MobileNetV3 model only uses efficient 3x3-DepConvs.

We note that the scalings we deploy would not necessarily be the optimal models in the TinyOps design space. In our experiments, we manually searched for scaled variations of

TABLE III
POWER CONSUMPTION OF TINYOPS COMPARED TO INTERNAL AND EXTERNAL MEMORY DEPLOYMENT

Platform	Power (mW)			Latency (ms)		
	Int	Ext	TinyOps	Int	Ext	TinyOps
L552	65	205	140	855	2963	977
F469	290	435	380	1089	2411	1131
F746	645	770	798	685	1004	772

the MobileNetV3 model in the TinyOps design space via trial and error. As opposed to a naive approach which might select a sub-optimal scaling where the width or resolution might be too high or low, we searched for balanced scalings close to the pareto frontier guided by heuristics obtained from our study. On average, this required 3 trials to derive each model where each trial consisted of training and evaluating the candidate model. This approach resulted in a high search cost of 108 GPU hours to derive each model. Additionally, only modifying the hyper-parameters of width and resolution with limited trials would not guarantee convergence to the optimal architecture on the pareto frontier. We believe, further performance gains could be achieved with automated algorithms utilising weight sharing [13], [32], [33], [41]–[43] mechanisms to evaluate a larger number of architectures with a lower search cost. Moreover, other hyper-parameters such as the number and size of kernels in each layer, depth or network topology could be included in the search in order to find the optimal architecture in the TinyOps design space which makes this a useful avenue to explore in future work.

B. Power Consumption and Energy Efficiency

We compared the power consumption and energy efficiency of deployment from internal or external memory and the TinyOps approach.

As can be observed in Table II, models deployed from the TinyOps design space had up to 2.2x higher power consumption than the internal memory models. However, this was easily offset by 1.4x to 3.7x lower latency achieved by efficient scalings of the MobileNetV3 model from the TinyOps design space to yield lower energy-per-inference on all devices.

To analyse the power consumption of the different memory configurations, we deployed the same model under the various configurations on each platform with Proxyless-w0.10-r192, Proxyless-w0.30-r144 and Proxyless-w0.30-r176 models deployed on the L552, F469 and F746 respectively. As shown in Table III, we observed that on the L552 and F469, TinyOps had lower power consumption than external memory inference even though it used the additional DMA peripheral. This is due to external memory inference requiring repetitive high energy reads of filters and tensors from SDRAM for every stride of convolution in DepConv operations. On the other hand, TinyOps overlays data from SDRAM to SRAM, requiring only one read from SDRAM with subsequent reads made from the low power SRAM. The power saved from reduced SDRAM reads outweighed the overhead of the DMA resulting in an overall reduction for TinyOps. However, the same behavior was not observed on the F746 due to the presence of cache which reduces SDRAM reads in external memory inference by providing repetitive low energy data access from itself. In this case the additional DMA used by TinyOps outweighs the reduced SDRAM reads in external memory inference for a higher power consumption. Nevertheless, we observed that TinyOps lower latency outweighed the 3.5% higher power consumption, for lower energy-per-inference.

C. Bill of Materials

Contrary to internal memory only deployment approaches, TinyOps uses external Flash to store the model data with internal Flash used only for the inference framework code requiring approximately 100KB. This enables the usage of cheaper variants of the same MCUs, with less internal Flash which is expensive to manufacture. We compared the bill of materials for the internal memory and TinyOps approach for an ARM® Cortex® M4 deployment scenario to find that at the time of writing, the cost of using an STM32F469NE by STMicroelectronics with 320KB of internal flash, supplemented with 8MB of external SDRAM and NOR Flash was actually less than using an STM32F469NI with 2048KB of internal flash. This would suggest that the TinyOps approach is competitive when considering cost in addition to being superior in accuracy, latency and energy efficiency.

VI. CONCLUSIONS

In our study we showed that the constraint of internal storage and memory leads to deployment of degenerate architectures with sub-optimal accuracy and latency. We relaxed this constraint using cheaper available external memories and accelerated inference up to 2x using the TinyOps framework. Using insights from our study, we deployed efficient models from the TinyOps design space to outperform state-of-the-art internal memory approaches with up to 3.1-6.7% higher accuracy and 1.4-2.1x faster inference latency. In future work, we believe the high search cost of our approach which manually derived models via trial and error could be reduced through automated NAS approaches utilising efficient weight sharing mechanisms. Further performance gains could also be achieved by including extra hyper-parameters in the search, in addition to width and resolution, in order to derive the optimal architecture in the TinyOps design space. Nevertheless, our work demonstrates the strength of the TinyOps design space, which suggests it should be studied further for deep learning applications in TinyML.

ACKNOWLEDGEMENTS

This work was supported by the UK Research and Innovation (UKRI) [EP/S024298/1], the Engineering and Physical Sciences Research Council (EPSRC) [EP/S030069/1], and ARM Limited, Cambridge, UK. For the purpose of open access, the author has applied a Creative Commons Attribution (CC BY) licence to any Author Accepted Manuscript version arising. Code and data associated with this paper is available at https://github.com/sulaimansadiq/TinyOps, and https://doi.org/10.5258/SOTON/D2850.

REFERENCES

[1] K. He, X. Zhang, S. Ren, and J. Sun, "Deep residual learning for image recognition," in *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition*, 2016, pp. 770–778.

[2] M. Tan and Q. Le, "Efficientnet: Rethinking model scaling for convolutional neural networks," in *International Conference on Machine Learning*. PMLR, 2019, pp. 6105–6114.

[3] G. Hinton, L. Deng, D. Yu, G. Dahl, A.-r. Mohamed, N. Jaitly, A. Senior, V. Vanhoucke, P. Nguyen, B. Kingsbury *et al.*, "Deep neural networks for acoustic modeling in speech recognition," *IEEE Signal processing magazine*, vol. 29, 2012.

[4] A. Vaswani, N. Shazeer, N. Parmar, J. Uszkoreit, L. Jones, A. N. Gomez, Ł. Kaiser, and I. Polosukhin, "Attention is all you need," *Advances in Neural Information Processing Systems*, vol. 30, 2017.

[5] D. Silver, A. Huang, C. J. Maddison, A. Guez, L. Sifre, G. Van Den Driessche, J. Schrittwieser, I. Antonoglou, V. Panneershelvam, M. Lanctot *et al.*, "Mastering the game of go with deep neural networks and tree search," *Nature*, vol. 529, no. 7587, p. 484, 2016.

[6] T. Insights, "Iot connected devices worldwide 2019-2030," *Statista*, 2022. [Online]. Available: https://www.statista.com/statistics/1183457/iot-connected-devices-worldwide/

[7] F. Samie, L. Bauer, and J. Henkel, "From cloud down to things: An overview of machine learning in internet of things," *IEEE Internet of Things Journal*, vol. 6, no. 3, pp. 4921–4934, 2019.

[8] G. Premsankar, M. Di Francesco, and T. Taleb, "Edge computing for the internet of things: A case study," *IEEE Internet of Things Journal*, vol. 5, no. 2, pp. 1275–1284, 2018.

[9] A. Zanella, N. Bui, A. Castellani, L. Vangelista, and M. Zorzi, "Internet of things for smart cities," *IEEE Internet of Things journal*, vol. 1, no. 1, pp. 22–32, 2014.

[10] G. Zazzaro, S. Cuomo, A. Martone, R. V. Montaquila, G. Toraldo, and L. Pavone, "Eeg signal analysis for epileptic seizures detection by applying data mining techniques," *IEEE Internet of Things Journal*, vol. 14, p. 100048, 2021.

[11] F. Liang, W. Yu, X. Liu, D. Griffith, and N. Golmie, "Toward edge-based deep learning in industrial internet of things," *IEEE Internet of Things Journal*, vol. 7, no. 5, pp. 4329–4341, 2020.

[12] P. Villalobos, J. Sevilla, T. Besiroglu, L. Heim, A. Ho, and M. Hobbhahn, "Machine learning model sizes and the parameter gap," *arXiv preprint arXiv:2207.02852*, 2022.

[13] J. Lin, W.-M. Chen, Y. Lin, C. Gan, S. Han *et al.*, "Mcunet: Tiny deep learning on iot devices," *Advances in Neural Information Processing Systems*, vol. 33, pp. 11 711–11 722, 2020.

[14] C. Banbury, C. Zhou, I. Fedorov, R. Matas, U. Thakker, D. Gope, V. Janapa Reddi, M. Mattina, and P. Whatmough, "Micronets: Neural network architectures for deploying tinyml applications on commodity microcontrollers," *Proceedings of Machine Learning and Systems*, vol. 3, 2021.

[15] A. Capotondi, M. Rusci, M. Fariselli, and L. Benini, "Cmix-nn: Mixed low-precision cnn library for memory-constrained edge devices," *IEEE Transactions on Circuits and Systems II: Express Briefs*, vol. 67, no. 5, pp. 871–875, 2020.

[16] M. Rusci, A. Capotondi, and L. Benini, "Memory-driven mixed low precision quantization for enabling deep network inference on microcontrollers," *Proceedings of Machine Learning and Systems*, vol. 2, pp. 326–335, 2020.

[17] J. Lin, W.-M. Chen, H. Cai, C. Gan, and S. Han, "Memory-efficient patch-based inference for tiny deep learning," *Advances in Neural Information Processing Systems*, vol. 34, pp. 2346–2358, 2021.

[18] S. Sadiq, J. Hare, P. Maji, S. Craske, and G. V. Merrett, "Tinyops: Imagenet scale deep learning on microcontrollers," in *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition*, 2022, pp. 2702–2706.

[19] M. Courbariaux, Y. Bengio, and J.-P. David, "Binaryconnect: Training deep neural networks with binary weights during propagations," *Advances in neural information processing systems*, vol. 28, 2015.

[20] C. Zhu, S. Han, H. Mao, and W. J. Dally, "Trained ternary quantization," *International Conference on Learning Representations*, 2017.

[21] I. Hubara, M. Courbariaux, D. Soudry, R. El-Yaniv, and Y. Bengio, "Quantized neural networks: Training neural networks with low precision weights and activations," *The Journal of Machine Learning Research*, vol. 18, no. 1, pp. 6869–6898, 2017.

[22] P. Stock, A. Joulin, R. Gribonval, B. Graham, and H. Jégou, "And the bit goes down: Revisiting the quantization of neural networks," *International Conference on Learning Representations*, 2020.

[23] A. Fan, P. Stock, B. Graham, E. Grave, R. Gribonval, H. Jegou, and A. Joulin, "Training with quantization noise for extreme model compression," *International Conference on Learning Representations*, 2021.

[24] Y. LeCun, J. S. Denker, and S. A. Solla, "Optimal brain damage," in *Advances in Neural Information Processing Systems*, 1990, pp. 598–605.

[25] S. Han, H. Mao, and W. J. Dally, "Deep compression: Compressing deep neural networks with pruning, trained quantization and huffman coding," *International Conference on Learning Representations*, 2016.

[26] X. Dong and Y. Yang, "Network pruning via transformable architecture search," in *Advances in Neural Information Processing Systems*, 2019, pp. 760–771.

[27] I. Fedorov, R. Matas, H. Tann, C. Zhou, M. Mattina, and P. Whatmough, "Udc: Unified dnas for compressible tinyml models for neural processing units," *Advances in Neural Information Processing Systems*, vol. 35, pp. 18 456–18 471, 2022.

[28] A. G. Howard, M. Zhu, B. Chen, D. Kalenichenko, W. Wang, T. Weyand, M. Andreetto, and H. Adam, "Mobilenets: Efficient convolutional neural networks for mobile vision applications," *arXiv preprint arXiv:1704.04861*, 2017.

[29] M. Sandler, A. Howard, M. Zhu, A. Zhmoginov, and L.-C. Chen, "Mobilenetv2: Inverted residuals and linear bottlenecks," in *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition*, 2018, pp. 4510–4520.

[30] M. Tan and Q. Le, "Efficientnet: Rethinking model scaling for convolutional neural networks," in *International Conference on Machine Learning*. PMLR, 2019, pp. 6105–6114.

[31] B. Wu, X. Dai, P. Zhang, Y. Wang, F. Sun, Y. Wu, Y. Tian, P. Vajda, Y. Jia, and K. Keutzer, "Fbnet: Hardware-aware efficient convnet design via differentiable neural architecture search," in *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition*, 2019, pp. 10 734–10 742.

[32] M. Tan, B. Chen, R. Pang, V. Vasudevan, M. Sandler, A. Howard, and Q. V. Le, "Mnasnet: Platform-aware neural architecture search for mobile," in *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition*, 2019, pp. 2820–2828.

[33] S. Sadiq, P. Maji, J. Hare, and G. Merrett, "Deff-arts: Differentiable efficient architecture search," *Advances in Neural Information Processing Systems, ML for Systems Workshop*, 2020.

[34] E. Liberis, Ł. Dudziak, and N. D. Lane, "μnas: Constrained neural architecture search for microcontrollers," in *Proceedings of the 1st Workshop on Machine Learning and Systems*, 2021, pp. 70–79.

[35] R. David, J. Duke, A. Jain, V. Janapa Reddi, N. Jeffries, J. Li, N. Kreeger, I. Nappier, M. Natraj, T. Wang *et al.*, "Tensorflow lite micro: Embedded machine learning for tinyml systems," *Proceedings of Machine Learning and Systems*, vol. 3, 2021.

[36] uTensor, "utensor." [Online]. Available: https://utensor.github.io/website/

[37] J. Lin, W.-M. Chen, H. Cai, C. Gan, and S. Han, "Memory-efficient patch-based inference for tiny deep learning," *Advances in Neural Information Processing Systems*, vol. 34, pp. 2346–2358, 2021.

[38] A. Burrello, A. Garofalo, N. Bruschi, G. Tagliavini, D. Rossi, and F. Conti, "Dory: Automatic end-to-end deployment of real-world dnns on low-cost iot mcus," *IEEE Transactions on Computers*, vol. 70, no. 8, pp. 1253–1268, 2021.

[39] X. Wang, M. Magno, L. Cavigelli, and L. Benini, "Fann-on-mcu: An open-source toolkit for energy-efficient neural network inference at the edge of the internet of things," *IEEE Internet of Things Journal*, vol. 7, no. 5, pp. 4403–4417, 2020.

[40] L. Lai, N. Suda, and V. Chandra, "Cmsis-nn: Efficient neural network kernels for arm cortex-m cpus," *arXiv preprint arXiv:1801.06601*, 2018.

[41] H. Cai, L. Zhu, and S. Han, "Proxylessnas: Direct neural architecture search on target task and hardware," *International Conference on Learning Representations*, 2018.

[42] A. Howard, M. Sandler, G. Chu, L.-C. Chen, B. Chen, M. Tan, W. Wang, Y. Zhu, R. Pang, V. Vasudevan *et al.*, "Searching for mobilenetv3," in *Proceedings of the IEEE/CVF International Conference on Computer Vision*, 2019, pp. 1314–1324.

[43] S. S. Saha, S. S. Sandha, L. A. Garcia, and M. Srivastava, "Tinyodom: Hardware-aware efficient neural inertial navigation," *Proceedings of the ACM on Interactive, Mobile, Wearable and Ubiquitous Technologies*, vol. 6, no. 2, pp. 1–32, 2022.

[44] C. Banbury, V. J. Reddi, P. Torelli, J. Holleman, N. Jeffries, C. Kiraly, P. Montino, D. Kanter, S. Ahmed, D. Pau *et al.*, "Mlperf tiny benchmark," *arXiv preprint arXiv:2106.07597*, 2021.