Intermittent-Aware Neural Network Pruning

Chih-Chia Lin[1,2], Chia-Yin Liu[2], Chih-Hsuan Yen[1,2], Tei-Wei Kuo[1,4], Pi-Cheng Hsiu[1,2,3]

[1]National Taiwan University, [2]Academia Sinica,[3]National Chi Nan University, Taiwan
[4]Mohamed bin Zayed University of Artificial Intelligence, UAE

***Abstract*—Deep neural network inference on energy harvesting tiny devices has emerged as a solution for sustainable edge intelligence. However, compact models optimized for continuously-powered systems may become suboptimal when deployed on intermittently-powered systems. This paper presents the pruning criterion, pruning strategy, and prototype implementation of iPrune, the first framework which introduces intermittency into neural network pruning to produce compact models adaptable to intermittent systems. The pruned models are deployed and evaluated on a Texas Instruments device with various power strengths and TinyML applications. Compared to an energy-aware pruning framework, iPrune can speed up intermittent inference by 1.1 to 2 times while achieving comparable model accuracy.**

***Index Terms*—Neural network pruning, deep learning, intermittent computing, battery-less devices**

I. Introduction

Advancements in *model compression* techniques are pushing *deep neural network* (DNN) inference to tiny devices [16], to enhance application responsiveness and user privacy while mitigating communication bandwidth usage. To maintain environmental sustainability, energy harvesting is deemed a promising alternative to battery charging [9], thereby enabling maintenance-free tiny devices with an extended lifespan. However, battery-less devices powered by ambient energy, which is inherently weak and unstable, suffer from frequent power failure and resumption. Consequently, DNN inference on such tiny devices has to be executed *intermittently* [5], [10], and *intermittent-aware deep learning* has emerged as a critical research topic to allow DNN models to be deployed and efficiently executed on battery-less tiny devices.

Intermittent deep inference enables DNN models to be executed on tiny devices with ambient power [5], [7], [10], [19]. Generally, existing approaches perform *progress preservation* and *progress recovery*. During inference, progress preservation continually saves some form of *progress indicator* along with the computed accelerator outputs into nonvolatile memory (NVM), thereby reducing the amount of progress lost in volatile memory (VM) when power fails. Upon power resumption, progress recovery utilizes the progress indicator to correctly resume, rather than re-execute from scratch, the interrupted inference. The difference among intermittent inference approaches lies in the used progress indicator, which directly affects how much data is preserved during inference and how much progress is re-executed during recovery. For example, SONIC/TAILS [5] adopts multiple loop indices as the progress indicator, preserved along with numerous accelerator outputs when each *task* is finished. During progress recovery, the entire atomic task interrupted should be re-executed. By contrast, HAWAII [10] uses a *job counter* as the progress indicator to track every accelerator output, where an accelerator operation comprises multiple sub-operations, each of which is referred to as a *job* and produces an *accelerator output*. Upon power resumption, only the interrupted job needs to be re-executed. These approaches simply take a given (unpruned) DNN model and execute it intermittently.

Neural network pruning removes relatively unimportant weight parameters from a pre-trained DNN model to trade off model accuracy for less hardware requirements [4], [6]. Generally, pruning approaches are different in the used *pruning criterion* and *pruning strategy*. The pruning criterion is used to estimate the importance of weights given a specific objective. For example, *magnitude-based pruning* tends to remove weights with relatively small absolute values [6], while *energy-aware pruning* prefers to remove weights with relatively high energy consumption [18]. On the other hand, the pruning strategy determines which weights to remove and how. Particularly, *one-shot pruning* prunes a pre-trained model only once and then retrains the pruned model to recover the accuracy loss [4], while *iterative pruning* removes a small percentage of the weights in each iteration to avoid an unrecoverable drop in accuracy [14]. Moreover, the *pruning granularity* affects the regularity and sparsity of the pruned model [12]. *Fine-grained pruning* removes individual weights, causing the pruned model to lack regularity and rely on specialized hardware for acceleration [6]. In contrast, *coarse-grained pruning* maintains the model regularity for ease of hardware acceleration, but at the cost of lower sparsity under the same accuracy loss [15]. Most recently, network pruning is used to produce multiple *shared-weight* models with different amounts of energy consumption [8], allowing dynamic model switching according to the available power strength. However, how to compress a model to specially adapt to intermittent systems has received little attention.

This paper presents an early attempt to address *intermittent-aware* neural network pruning. We observe that DNN inference behaves differently in hardware usage on continuously-powered and intermittently-powered systems. For conventional DNN inference, as the accelerator outputs are normally kept in VM whenever possible to maximize *data reuse*, the inference latency is mostly contributed by NVM reads and accelerator computations. In contrast, intermittent DNN inference requires the accelerator outputs and the corresponding progress indicators to be instantly written back into NVM, resulting in numerous NVM writes that dominate the inference latency. Consequently, a pruned model optimized for continuously-powered systems may be suboptimal when deployed and executed on intermittently-powered systems.

To realize intermittent-aware pruning, we develop *iPrune*, a simple yet effective framework inspired by the observation above to produce compact models that can reduce *intermittent* inference latency while maintaining model accuracy. Specifically, iPrune introduces a pruning criterion that reflects the impact of different weights on the intermittent inference latency despite varied power strengths. Furthermore, iPrune adopts a multi-step pruning strategy, with respective guidelines for network-, layer-, and block-level pruning, to remove more weights on layers with high intermittent inference latency but low sensitivity to model accuracy. The pruned models produced by iPrune are run by a HAWAII-extended intermittent inference engine [1] and deployed on a Texas Instruments MSP430 device. Experiments are conducted with various power strengths and representative TinyML applications [3]. Compared to a framework similar in concept to energy-aware pruning [18] intended for continuously-powered systems, iPrune can produce more compact models with 1.1 to 2 times faster intermittent inference and comparable accuracy.

The remainder of this paper is organized as follows. Section II provides background information and explains the motivation. In Section III, we present the pruning criterion, pruning strategy, and implementation issues behind iPrune. The experimental results are reported in Section IV. Section V contains some concluding remarks.

II. Background and Motivation

A. Tiny Deep Learning

A deep neural network (DNN) comprises an input layer and an output layer, as well as multiple hidden layers such as convolutional (CONV), fully connected (FC), and pooling (POOL) layers. Each hidden layer performs specific operations on input feature maps (IFMs) to generate output feature maps (OFMs) for the subsequent layers, while possessing a set of tunable parameters called *weights* for optimization [17]. To adapt a DNN model to resource-constrained tiny devices, neural network pruning is widely used to reduce its model size and computational complexity without significant accuracy loss, by removing redundant weights [6]. Accordingly, to determine which weights to be pruned, a *pruning criterion* is essential to estimate the importance of each weight in terms of a specific objective, and those weights with relatively less importance will be pruned according to a *pruning strategy* to achieve the objective while maintaining model accuracy. However, the pruned weights will be indicated with zeros, occupying storage resources despite having no significance for computation. Thus, the pruned model is typically stored in a compressed format, where an indexing scheme is used to record the positions of non-zero weights and thereby skip zero weights during inference.

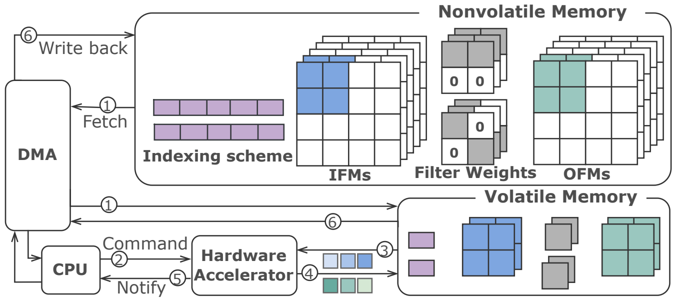
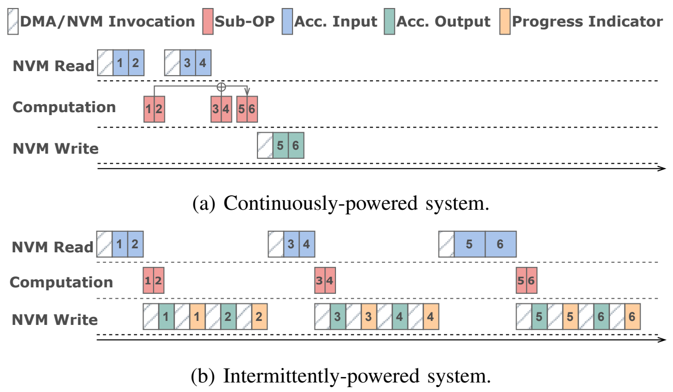


Fig. 1. Inference flow of a pruned DNN layer.

Figure 1 shows the inference flow of a pruned DNN layer on a tiny device. Due to the limited VM capacity, IFMs, OFMs, and model parameters (e.g., weights and biases), are stored in NVM and processed as logically partitioned *tiles* in VM for performance consideration [2]. To compute an OFM tile, the indexing scheme is referred to correctly fetch the corresponding IFM tile and weight tile, as well as biases and previously computed partial sums, from NVM to VM. Then, a hardware accelerator, increasingly available on off-the-shelf MCUs, is used to offload the expensive multiply and accumulate (MAC) operations to accelerate tile computation. Typically, completing an OFM tile requires several accelerator operations, each of which contains multiple atomic sub-operations. One single accelerator output represents the *minimum* intermediate output (e.g., a partially accumulated or fully completed output feature) produced by an accelerator sub-operation in a VM region shared with the CPU. To leverage *data reuse*, the produced accelerator outputs would be stored to accumulate in VM and written back to NVM only when the OFM tile is fully completed [2]. Data transfers between VM and NVM can be expedited via a direct memory access (DMA) controller. One data transfer command transfers the data stored in contiguous memory locations, and its latency includes the DMA invocation, the NVM invocation, and the NVM read/write latency, which depends on the amount of transferred data.

B. Intermittent Deep Inference

Deep inference behaves differently between continuously-powered and intermittently-powered systems. Intermittent systems operate via ambient power, with the harvested energy buffered into an energy buffer (e.g., a capacitor). The system is powered on when the energy buffer is fully charged and powered off when the energy buffer is depleted [5], [10]. Typically, an end-to-end inference would require dozens to a few hundreds of power cycles to complete. Each power failure interrupts inference progress and causes data loss in VM. Therefore, storing accelerator outputs for accumulation in VM, like the inference flow presented in Section II-A, may lead to nontermination as the interrupted inference needs to be performed from scratch in every power cycle. To continue inference across power cycles, intermittent systems utilize NVM to perform *progress preservation*. Specifically, each accelerator output (or a batch of accelerator outputs) is paired with a progress indicator to track the inference progress, and they are immediately written from VM back to NVM in parallel to tile computation during inference. Upon power resumption, *progress recovery* is performed where, after system reboot, the progress indicators are used to identify the last preserved accelerator output and correctly resume the interrupted inference. The progress related to unpreserved accelerator outputs is simply re-executed. State-of-the-art intermittent inference approaches differ in the used progress indicators, such as loop indices [5], layer/job counters [10], or states [19], which directly affect the progress preservation and recovery costs.



(a) Continuously-powered system.

(b) Intermittently-powered system.

Fig. 2. Hardware usage on different inference systems.

This work is motivated by the difference between the hardware usage behavior on continuously-powered and intermittently-powered inference systems. As illustrated in Figure 2(a), the inference latency of a continuously-powered system is more contributed by NVM reads and accelerator computations than NVM writes, because accelerator outputs can be stored and accumulated in VM for better data reuse. In contrast, NVM writes dominate the inference latency of an intermittently-powered system, as shown in Figure 2(b), because progress preservation is indispensable to continued inference progress across power cycles, causing numerous NVM writes to immediately backup the accelerator outputs and their progress indicators back into

NVM. Existing pruning approaches are designed to improve inference latency on continuously-powered systems by reducing the numbers of NVM reads and accelerator computations. Although these approaches can greatly reduce the DNN model size, the pruned DNN model, when deployed on intermittent systems, may suffer from high latency, raising the need for a new network pruning approach that reduces the number of NVM writes to accelerate intermittent DNN inference.

III. Intermittent-Aware Neural Network Pruning

A. Design Overview

This section presents an intermittent-aware pruning framework called *iPrune*, which follows the *estimate-prune-retrain* principle of classic pruning frameworks [18], to reduce the intermittent inference latency while maintaining the model accuracy. iPrune adopts iterative pruning [14] to iteratively eliminate a small percentage of the model weights at a time, thereby achieving less pruning-induced accuracy loss than a threshold (ϵ), within which the loss remains possible to recover via fine-tuning. Since the accuracy typically fluctuates across iterations, to permit a brief rally in the early iterations, the model is pruned iteratively until the accuracy drop exceeds the threshold for twice (i.e., with a given second chance), and then the most compact model with accuracy recovered is adopted as the pruned model. As shown in Figure 3, for each iteration, iPrune first performs layer-wise criterion estimation to evaluate the intermittent inference latency of each layer, as well as analyze the *sensitivity* to model accuracy degradation once weights in the layer are pruned. Then, the estimated latency and sensitivity are used to determine an overall pruning ratio for the remaining network, the pruning ratios allocated among layers, and the to-be-pruned weight blocks within each layer. Finally, the network is actually pruned and fine-tuned to remediate accuracy degradation.

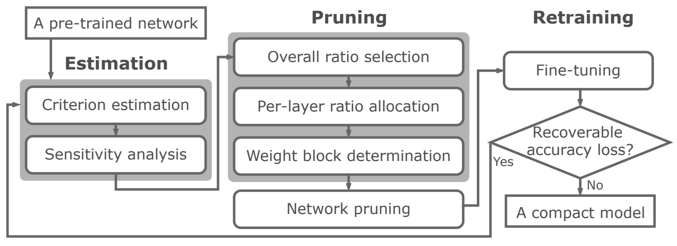


Fig. 3. The iPrune Framework.

However, realizing iPrune raises several design challenges. One major challenge is to determine an appropriate criterion that reflects the impact of different weights on intermittent inference latency. The total latency of NVM writes, which dominates the intermittent inference latency (as observed in Section II-B), cannot be directly adopted as a criterion because the number of incurred NVM writes may vary with intermittency and the amount of data transferred by each NVM write may also be different. Thus, the pruning criterion should be capable of estimating inference latency, despite the variety of intermittency. Another challenge is then to exploit the criterion to determine which weights to be pruned, so as to reduce the inference latency while maintaining model accuracy. For latency reduction, we would ideally remove the weights that contribute the most inference latency, but these weights may also significantly impact model accuracy. Thus, the pruning strategy should jointly consider latency and accuracy. Our pruning criterion and strategy are respectively presented in Sections III-B and III-C, while Section III-D discusses some implementation issues.

B. Pruning Criterion

The intermittent DNN inference latency depends significantly on both the progress preservation and recovery costs, as accelerator computation and progress preservation are pipelined and performed in parallel (Figure 2(b)). For progress preservation, although the total latency of NVM writes is difficult to estimate due to intermittency, the total amount of data written back to NVM is not significantly affected by intermittency. Since all accelerator outputs and corresponding progress indicators are written back instantly to accumulate inference progress, the preservation overhead decreases with the decrement of accelerator outputs and associated progress indicators. As a result, the preservation overhead correlates to the number of accelerator outputs, which can be calculated easily based on the DNN model structure and the inference engine configuration (e.g., the tile size and dataflow).

Apart from the latency caused by progress preservation, the criterion should also be capable of reflecting the latency of progress recovery. Under weaker power, the intermittent system will experience more frequent power failures, and the recovery cost incurred during an end-to-end inference will also increase as a consequence. The recovery cost depends on the number of power failures and the cost per power failure. Upon power resumption, the recovery cost per power failure is largely contributed by the data re-fetch latency, which can roughly be regarded as a constant cost when the data of a *fixed* tile size that fully utilizes the VM is always fetched for tile computation. Thus, reducing the number of power failures will improve the intermittent inference latency. Unlike continuously-powered inference, where the accelerator is often fully utilized, intermittently-powered inference typically incurs a high NVM utilization due to numerous NVM writes. Since those NVM writes account for a large portion of energy consumption, the number of power failures decreases with less accelerator outputs written back to NVM. As a result, both the progress preservation and recovery costs are correlated to the number of accelerator outputs, and iPrune can use it as the pruning criterion.

C. Pruning Strategy

Using the pruning criterion, our pruning strategy strives to reduce the number of accelerator outputs of a given model to improve intermittent inference latency while maintaining model accuracy. iPrune adopts iterative pruning, which allows pruning-induced accuracy loss to be recoverable by fine-tuning [14]. Firstly, for each iteration, iPrune determines an overall pruning ratio (Γ), so as to remove an appropriate percentage of the weights in the whole network. Secondly, given the overall ratio, iPrune allocates per-layer pruning ratios (γ_i) to all n layers in consideration of both latency and accuracy. Lastly, iPrune decides which weights to be pruned in each layer in accordance with the allocated pruning ratio and pruning granularity. The three-step pruning strategy is illustrated in Figure 4 and detailed below.

For an iteration, using a large pruning ratio may cause a considerable accuracy drop. In contrast, using a small ratio would take much time to complete the whole pruning process and sometimes even not make pruning progress. Therefore, an appropriate ratio (Γ) should be selected for each iteration. Ideally, a small pruning ratio is set if the weights selected to prune will result in a large drop in accuracy, and vice versa. It has been observed that the degree of redundancy differs between layers, leading to different degrees of sensitivity to model accuracy, where the sensitivity of a layer represents how significant the weights of the layer may affect accuracy [6]. When the layers with more accelerator outputs are sensitive to pruning, a larger pruning ratio will cause a more significant drop in accuracy because our strategy attempts to eliminate more accelerator outputs. Thus, we

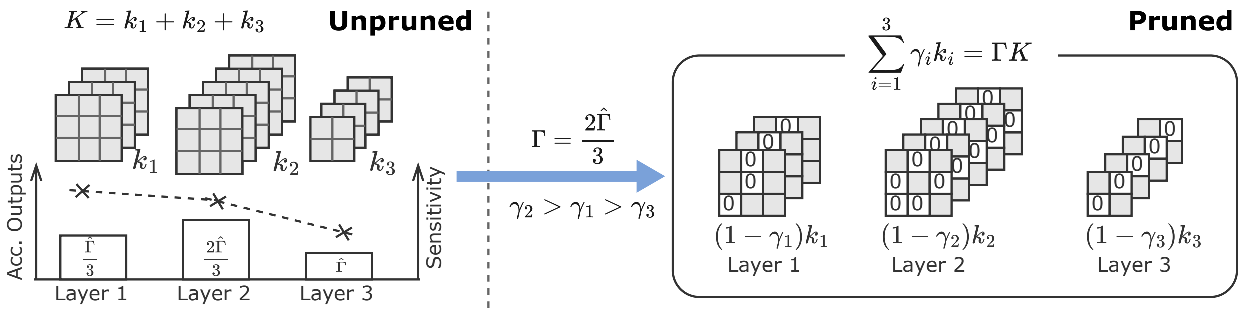


Fig. 4. A three-step strategy for network-, layer-, and block-level pruning based on per-layer latency and sensitivity.

propose our first guideline, wherein the overall pruning ratio should be set at a smaller value if the layer with the most accelerator outputs has a higher pruning sensitivity. According to the guideline, iPrune first performs sensitivity analysis on each layer to rank all n layers by sensitivity. Then, given an upper bound ($\hat{\Gamma}$) on the overall pruning ratio, iPrune maps n ratios within the upper bound evenly to the layers based on their ranks in decreasing order (i.e., the layer with rank i is mapped to $i \times \frac{\hat{\Gamma}}{n}$). Finally, the ratio mapped to the layer with the most accelerator outputs is selected as the overall pruning ratio (Γ) in this iteration.

Given the overall ratio (Γ), iPrune then allocates a pruning ratio to each layer. Intuitively, to eliminate more accelerator outputs, we should prune those weights which produce more accelerator outputs. Accordingly, our second guideline is to allocate a higher pruning ratio to a layer with a larger number of accelerator outputs. Note that the *layer-wise* criterion estimation is adopted because the numbers of accelerator outputs contributed by intra-layer weights are the same but may be different across inter-layer weights. However, determining an appropriate pruning ratio (γ_i) for each layer (i) by trial and error is extremely time-consuming. Thus, iPrune employs a search-based algorithm, with the objective of minimizing the number of accelerator outputs while maintaining the model accuracy, to find a set of pruning ratios such that $\sum_{i=1}^{n} \gamma_i k_i = \Gamma K$, where K denotes the total number of weights and k_i denotes the number of weights in each layer i.

Given the pruning ratio (γ_i) of a layer (i), iPrune determines which weights in the layer to prune eventually. Coarse-grained pruning may cause a higher accuracy drop. In contrast, fine-grained pruning can achieve a greater sparsity without a noticeable accuracy drop but may not be hardware-friendly. In other words, using a pruning granularity inconsistent with the inference engine and hardware platform may produce no significant reduction in latency [12]. In intermittent deep inference, if the weights computed by an accelerator operation are not removed altogether, even though the computational complexity required for each accelerator output can be reduced, the accelerator operation still needs to be invoked and may produce the original number of accelerator outputs. As a consequence, the number of NVM writes is not effectively reduced. Thus, our third guideline is that the pruning granularity should be a block of weights to be computed by one single accelerator operation. Accordingly, iPrune evaluates the impact of each *weight block* on model accuracy and sequentially removes weight blocks with the minimum impact until the pruning ratio (γ_i) of the layer is achieved.

D. Implementation Issues

Our implementation involves both server and device sides. On the server side, our iPrune implementation adopts *simulated annealing* [11] to search for per-layer pruning ratios ($\gamma_i, \forall i$), but any search algorithm could be used instead. By default, the upper bound ($\hat{\Gamma}$) on the overall pruning ratio in an iteration is empirically set at 40%, and the threshold (ϵ) of recoverable accuracy loss is set at 1%, according to some experimental results in related studies [14], [18]. To estimate the impact of a weight block on model accuracy, we use a commonly-used metric, namely the *root mean square* (RMS) of a set of values [20]. The pruning granularity in iPrune is configured as one accelerator operation that computes a weight block, with the same *loop tiling* and *ordering* used in [2] for matrix multiplication.

On the device side, the pruned models are run by HAWAII [10], which comprises an intermittent-aware inference engine and library developed for battery-less tiny devices. In HAWAII, each network layer is represented by a weight matrix, regardless of zero or nonzero weights. To store and compute *sparse matrices* more efficiently, we integrate a commonly-used indexing scheme, called *Block Compressed Sparse Row* (BSR), into HAWAII. BSR represents a matrix by three one-dimensional arrays, one for storing the nonzero weight blocks while the other two for jointly indexing each nonzero weight block in the original matrix. Thus, the inference progress is jointly indicated by the current indices of the three arrays. During progress recovery, the interrupted job in the last incomplete weight block can be derived by the *job counter* preserved in NVM immediately before power failure. Although two extra NVM reads are required to locate any nonzero weight block, thereby correctly fetching the related tile input into VM, the BSR format can avoid considerable computations with zeros and unnecessary NVM writes given sufficiently high sparsity. In addition, to further speed up intermittent inference, we enhance HAWAII with several implementation optimizations adopted in [19], including accelerated vector-matrix multiplication, tile input data transformation, and tile size selection to fully utilize the VM and maximize data reuse, as well as support for multiple path networks. The enhanced HAWAII is denoted as HAWAII$^+$.

IV. Evaluation

A. Experimental Setup

We run the model pruned by iPrune with HAWAII$^+$. The pruned model, together with the inference engine, is stored in a 512 KB external FRAM (NVM) module and executed on a Texas Instruments (TI) MSP430 device, equipped with a Low Energy Accelerator (LEA) and 8 KB internal SRAM (VM). The device is powered by a TI BQ25504 energy management unit, which comprises a 100 µF capacitor to buffer energy and a power switch to respectively turn on or off the device when the capacitor is fully charged or depleted. To emulate representatives of solar energy under different conditions, we use a B&K Precision programmable power supply to generate different power strengths. Under *continuous power* (1.65 W), the device can operate continuously. In contrast, both *strong power* (8 mW) and *weak power* (4 mW) are insufficient to operate the device continuously, resulting in repeated yet unpredictable power failures. Note that HAWAII$^+$ still performs immediate progress preservation under continuous power, as it assumes no prior knowledge of the

TABLE II
TINYML APPLICATIONS USED FOR EVALUATION

Applications	Layers	Model Size	MACs	Acc. Outputs	Diversity
SQN: Image Recognition Dataset: CIFAR-10	CONV x 11 POOL x 2	147 KB	4442 K	1483 K	Low
HAR: Human Activity Detection Dataset: Accelerometer sensor data	CONV x 3 POOL x 3 FC x 1	28 KB	321 K	77 K	Medium
CKS: Speech Keyword Spotting Dataset: Speech commands	CONV x 2 FC x 3	131 KB	2811 K	1582 K	High

TABLE I
SPECIFICATIONS OF THE EXPERIMENTAL ENVIRONMENT

Hardware	
MCU	TI MSP430FR5994
Volatile memory	8KB SRAM
Non-volatile memory	Cypress CY15B104Q 512KB FRAM
Accelerator	TI Low-Energy Accelerator
Energy	
Power supply	B&K Precision 9171B
Boost converter	TI BQ25504
Switch on/off voltage	2.8 V / 2.4 V
Capacitance	100 μF
Continuous power	1.65 W = 3.3 V $\times$ 0.5 A
Strong power	8 mW = 1 V $\times$ 8 mA
Weak power	4 mW = 1 V $\times$ 4 mA

ambient power strength at runtime. Table I details our experimental environment.

We select three DNN models typically used in TinyML applications [3] and fit within the 512 KB NVM. As shown in Table II, they are respectively used for image recognition (referred to as SQN), human activity detection (referred to as HAR), and speech keyword spotting (referred to as CKS). These models vary in their architectures, sizes, and number of MAC computations. In a model, layers can have a diverse number of accelerator outputs, which depends on not only the model architecture but also the tile size and the types of accelerator operations used by HAWAII$^+$ for each layer. SQN, HAR, and CKS are respectively representatives of models with low, medium, and high *diversity* among layers. To deploy these models on the MSP430 device, the model parameters are quantized from the 32-bit floating point representation used during pruning to a 16-bit fixed point representation (Q15.1 format), without a significant accuracy loss.

There is no intermittent-aware pruning framework that considers intermittency during pruning. Consequently, we choose *energy-aware pruning* [18] intended for continuously-powered systems as a relatively suitable comparison, since energy consumption also affects intermittent inference latency. Note that iPrune is not developed to compete with state-of-the-art pruning frameworks, but to evaluate the performance gains when network pruning is optimized for intermittency. Our iPrune is compared with *ePrune*, which attempts to prune more on layers with higher energy consumption[1] while maintaining accuracy. The original unpruned model (denoted as *Unpruned*) is also deemed a baseline. All the models are run with HAWAII$^+$. First, we investigate some characteristics of the models pruned by different frameworks. We then deploy the models on the MSP430 device and measure the inference latency under different power strengths.

B. Pruned Models

Table III shows some concerned characteristics of the respective models pruned by ePrune and iPrune, as well as the unpruned models. The model accuracy is evaluated by each respective dataset in

[1]The MSP430 device's energy model was profiled by a set of micro-benchmarks provided in [13].

TABLE III
CHARACTERISTICS OF THE PRUNED MODELS

Models		Accuracy	Model Size	MACs	Acc. Outputs
SQN	Unpruned	76.3%	147 KB	4442 K	1483 K
	ePrune	75.5%	56 KB	1617 K	561 K
	iPrune	75.5%	55 KB	1560 K	518 K
HAR	Unpruned	92.5%	28 KB	321 K	77 K
	ePrune	92.7%	14 KB	183 K	56 K
	iPrune	92.7%	9 KB	108 K	44 K
CKS	Unpruned	87.5%	131 KB	2811 K	1582 K
	ePrune	87.6%	75 KB	1047 K	987 K
	iPrune	87.7%	67 KB	1149 K	509 K

Table II. In comparison with the unpruned model, the models pruned by ePrune and iPrune both achieve comparable accuracy, because they use the same recoverable threshold (ϵ) to restrain accuracy loss. Specifically, iPrune shows an accuracy loss of 0.8% for the SQN model, but an accuracy gain of 0.2% for the HAR and CKS models although, by design, iPrune does not improve the model accuracy. Moreover, both ePrune and iPrune can substantially reduce the model size, which includes all model parameters and indexing structures in the BSR format. However, compared with ePrune, iPrune achieves a further reduction of 36% at most and 16% on average in the model size. The reason for the reduction is that iPrune usually selects a smaller overall ratio (Γ) than ePrune, as the first few layers in modern network architectures usually have more accelerator outputs and higher sensitivity to pruning. Consequently, iPrune can perform more pruning iterations than ePrune before the accuracy loss becomes hard to recover.

A reduced model size is also beneficial to the decrement of MACs. Surprisingly, although 10% more MACs are necessary for the CKS model pruned by iPrune than by ePrune, iPrune shows a computation reduction of 0.4% for the SQN model and even 40% for the HAR model. As for the number of accelerator outputs, iPrune achieves a reduction of 48% at most and 26% on average, compared with ePrune. The reduction is more manifest with increased diversity among layers, because iPrune will allocate more diverse per-layer pruning ratios (γ) to different layers, with larger ratios allocated to layers having more accelerator outputs. As a result, the layers with the largest pruning ratios may vary with different pruning frameworks, and the pruned models can thus be very different.

C. Inference Latency

Figure 5 shows the intermittent inference latency, defined as the average time (in seconds) required to complete one single end-to-end inference, for different TinyML applications under various power strengths. The numbers above the bar plots indicate how many times a model pruned by iPrune can run faster than the counterpart pruned by ePrune and the unpruned model. As expected, both ePrune and iPrune can drastically reduce the inference latency for all evaluated models under all power strengths, showing the efficacy of neural network pruning even for TinyML applications. Furthermore, iPrune achieves larger inference latency reductions than ePrune across all

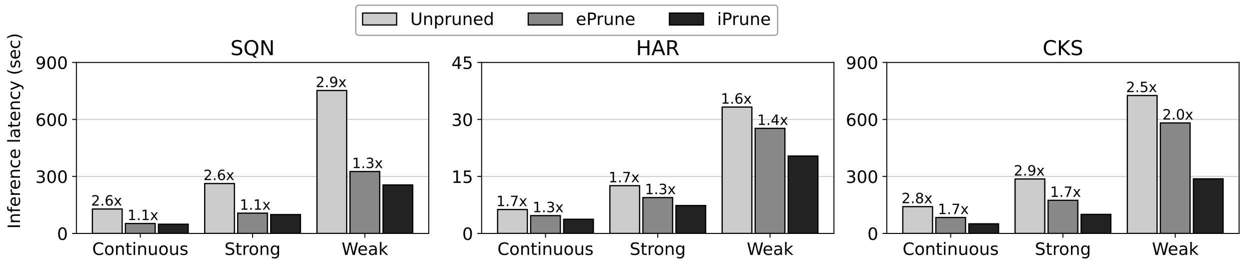


Fig. 5. Inference latency of various TinyML applications with pruned models under different power strengths.

evaluation conditions, which correlates well with the reduced model size, the decrement of accelerator outputs, and sometimes also the decrement of MACs, as discussed in Section IV-B.

We observe that the reduction in inference latency is more obvious when a model has greater diversity among layers, where iPrune and ePrune would prune more weights on different layers and generate significantly different models. Generally, the reduction remains consistent under various power strengths. Under weaker power, the device experiences more frequent power failures, resulting in a higher recovery cost during an end-to-end inference, and thus higher latency. As discussed in Section III-B, iPrune incurs less power failures by eliminating more accelerator outputs, while both incur similar data re-fetching latency during each progress recovery. iPrune shows a further improvement over ePrune when the power is weak. Overall, iPrune can speed up intermittent DNN inference of the unpruned models by 1.7 to 2.9 times, and by 1.1 to 2 times compared to ePrune, depending on the evaluated TinyML application.

V. Conclusion

This paper advocates that intermittent DNN inference should use custom models, rather than directly running existing compact models pruned for continuously-powered systems. To demonstrate the efficacy of this proposition, we develop iPrune, which explores the hardware usage behavior induced for enabling intermittency, to eliminate more accelerator outputs while balancing the inference latency and model accuracy. The models pruned by iPrune are run with a hardware accelerated intermittent inference engine, HAWAII$^+$, and deployed on a Texas Instruments MSP430 device under various power strengths to run representative TinyML applications [3]. Our experimental results show that an intermittent-aware pruning framework, even as simple as iPrune, can significantly improve intermittent inference latency, and the improvement remains consistent under various power strengths.

Our iPrune framework has been made open along with HAWAII$^+$ [1], facilitating the development of increasingly complicated intelligent applications on intermittent tiny devices[2]. We hope this work brings new thinking and motivates further research on the adaptation of various model compression techniques, like *matrix decomposition* and *weight sharing* [17], to intermittent systems.

VI. Acknowledgement

This work was supported in part by National Science and Technology Council, Taiwan, under Grant NSTC 110-2222-E-001-003-MY3 and by Academia Sinica under Grant AS-TP-110-M07.

References

[1] "The iPrune open project," https://github.com/EMCLab-Sinica/iPrune.

[2] A. Anderson, A. Vasudevan, C. Keane, and D. Gregg, "High-Performance Low-Memory Lowering: GEMM-based Algorithms for DNN Convolution," in *Proc. of IEEE SBAC-PAD*, 2020, pp. 99–106.

[3] C. R. Banbury, V. J. Reddi, M. Lam, W. Fu, A. Fazel, J. Holleman, X. Huang, R. Hurtado, D. Kanter, A. Lokhmotov, D. Patterson, D. Pau, J. sun Seo, J. Sieracki, U. Thakker, M. Verhelst, and P. Yadav, "Benchmarking TinyML Systems: Challenges and Direction," in *Proc. of MLSys*, 2020.

[4] J. Frankle and M. Carbin, "The Lottery Ticket Hypothesis: Finding Sparse, Trainable Neural Networks," in *Proc. of ICLR*, 2019.

[5] G. Gobieski, B. Lucia, and N. Beckmann, "Intelligence Beyond the Edge: Inference on Intermittent Embedded Systems," in *Proc. of ACM ASPLOS*, 2019, pp. 199–213.

[6] S. Han, J. Pool, J. Tran, and W. J. Dally, "Learning Both Weights and Connections for Efficient Neural Networks," in *Proc. of NIPS*, 2015, pp. 1135–1143.

[7] S. Islam, J. Deng, S. Zhou, C. Pan, C. Ding, and M. Xie, "Enabling Fast Deep Learning on Tiny Energy-Harvesting IoT Devices," in *Proc. of IEEE/ACM DATE*, 2022, pp. 921–926.

[8] S. Islam, S. Zhou, R. Ran, Y.-F. Jin, W. Wen, C. Ding, and M. Xie, "EVE: Environmental Adaptive Neural Network Models for Low-power Energy Harvesting System," in *Proc. of ACM/IEEE ICCAD*, 2022, pp. 1–8.

[9] H. Jayakumar, K. Lee, W. S. Lee, A. Raha, Y. Kim, and V. Raghunathan, "Powering the Internet of Things," in *Proc. of ACM ISLPED*, 2014, pp. 375–380.

[10] C.-K. Kang, H. R. Mendis, C.-H. Lin, M.-S. Chen, and P.-C. Hsiu, "Everything Leaves Footprints: Hardware Accelerated Intermittent Deep Inference," *IEEE TCAD*, vol. 39, no. 11, pp. 3479–3491, 2020.

[11] N. Liu, X. Ma, Z. Xu, Y. Wang, J. Tang, and J. Ye, "AutoCompress: An Automatic DNN Structured Pruning Framework for Ultra-High Compression Rates," in *Proc. of AAAI*, 2020, pp. 4876–4883.

[12] H. Mao, S. Han, J. Pool, W. Li, X. Liu, Y. Wang, and W. J. Dally, "Exploring the Granularity of Sparsity in Convolutional Neural Networks," in *Proc. of IEEE CVPRW*, 2017, pp. 1927–1934.

[13] H. R. Mendis, C.-K. Kang, and P.-C. Hsiu, "Intermittent-Aware Neural Architecture Search," *ACM TECS*, vol. 20, no. 5s, pp. 1–27, 2021.

[14] J. T. C. Min and M. Motani, "DropNet: Reducing Neural Network Complexity via Iterative Pruning," in *Proc. of ICML*, 2020, pp. 9356–9366.

[15] W. Niu, X. Ma, S. Lin, S. Wang, X. Qian, X. Lin, Y. Wang, and B. Ren, "PatDNN: Achieving Real-Time DNN Execution on Mobile Devices with Pattern-Based Weight Pruning," in *Proc. of ASPLOS*, 2020, pp. 907–922.

[16] M. Shafique, T. Theocharides, V. J. Reddy, and B. Murmann, "TinyML: Current Progress, Research Challenges, and Future Roadmap," in *Proc. of ACM/IEEE DAC*, 2021, pp. 1303–1306.

[17] V. Sze, Y.-H. Chen, T.-J. Yang, and J. S. Emer, "Efficient Processing of Deep Neural Networks: A Tutorial and Survey," *Proc. of the IEEE*, vol. 105, no. 12, pp. 2295–2329, 2017.

[18] T.-J. Yang, Y.-H. Chen, and V. Sze, "Designing Energy-Efficient Convolutional Neural Networks Using Energy-Aware Pruning," in *Proc. of IEEE CVPR*, 2017, pp. 6071–6079.

[19] C.-H. Yen, H. R. Mendis, T.-W. Kuo, and P.-C. Hsiu, "Stateful Neural Networks for Intermittent Systems," *IEEE TCAD*, vol. 41, no. 11, pp. 4229–4240, 2022.

[20] J. Yu, A. Lukefahr, D. Palframan, G. Dasika, R. Das, and S. Mahlke, "Scalpel: Customizing DNN pruning to the underlying hardware parallelism," in *Proc. of ACM/IEEE ISCA*, 2017, pp. 548–560.

[2]Interested readers may refer to a solar-powered inference system developed with iPrune and HAWAII$^+$ at https://youtu.be/sJgtzkPylpA.