

# EVE: Environmental Adaptive Neural Network Models for Low-power Energy Harvesting System

\*Sahidul Islam<sup>1</sup>, \*Shanglin Zhou<sup>2</sup>, Ran Ran<sup>3</sup>, Yu-Fang Jin<sup>1</sup>, Wujie Wen<sup>3</sup>, Caiwen Ding<sup>2</sup>, Mimi Xie<sup>1</sup>

<sup>1</sup>Department of Computer Science, The University of Texas at San Antonio

<sup>2</sup>Department of Computer Science, University of Connecticut

<sup>3</sup>Department of Electrical and Computer Engineering, Lehigh University

## ABSTRACT

IoT devices are increasingly being implemented with neural network models to enable smart applications. Energy harvesting (EH) technology that harvests energy from ambient environment is a promising alternative to batteries for powering those devices due to the low maintenance cost and wide availability of the energy sources. However, the power provided by the energy harvester is low and has an intrinsic drawback of instability since it varies with the ambient environment. This paper proposes EVE, an automated machine learning (autoML) co-exploration framework to search for desired multi-models with shared weights for energy harvesting IoT devices. Those shared models incur significantly reduced memory footprint with different levels of model sparsity, latency, and accuracy to adapt to the environmental changes. An efficient on-device implementation architecture is further developed to efficiently execute each model on device. A run-time model extraction algorithm is proposed that retrieves individual model with negligible overhead when a specific model mode is triggered. Experimental results show that the neural networks models generated by EVE is on average 2.5X times faster than the baseline models without pruning and shared weights.

## KEYWORDS

Energy Harvesting, DNN, Pruning, IoT, Memory Footprint

### ACM Reference Format:

\*Sahidul Islam<sup>1</sup>, \*Shanglin Zhou<sup>2</sup>, Ran Ran<sup>3</sup>, Yu-Fang Jin<sup>1</sup>, Wujie Wen<sup>3</sup>, Caiwen Ding<sup>2</sup>, Mimi Xie<sup>1</sup>. 2022. EVE: Environmental Adaptive Neural Network Models for Low-power Energy Harvesting System. In *Proceedings of ACM Conference (ICCAD '2022)*. ACM, New York, NY, USA, 9 pages. <https://doi.org/XXXXXXX.XXXXXXX>

## 1 INTRODUCTION

In the wake of democratization of artificial intelligence, there are increasing demands on executing deep neural network (DNN) models on sensing devices for better accuracy and more efficient prediction [1, 2] for various IoT applications [3, 4]. However, when DNN

models come to on-board, there is a grand challenge to accommodate the giant models to sensing devices with limited memory and computing resources [5, 6], and how to satisfy the real-time processing requirement. Particularly, first, embedded IoT devices have limited computational units and low CPU frequency (e.g., 1-16MHz). Since DNNs are computationally expensive, DNN algorithm takes long on-board execution time. Second, embedded IoT devices are equipped with small memory (e.g., hundreds of KBs) which can not even afford tiny DNN models (e.g., Tens of MBs). In addition, these battery-powered devices naturally have a limited battery stand-by time.

Energy harvesting technology that harvests energy from ambient environment is a promising alternative due to the low maintenance cost and wide availability of the energy sources. However, the power provided by the energy harvester is low and has an intrinsic drawback of instability since it varies with the ambient environment. For example, solar cells can generate power of different densities ranging from 0 to  $15mW/cm^2$  depending on the varying light intensity. With an unstable low power supply, the DNN execution will be interrupted frequently, resulting in significantly increased execution time. As a result, a deployed model that has 1 second real-time performance when there is high power intensity may execute for 10s when the power intensity is low, resulting in dramatically degraded quality-of-service (QoS).

This paper proposes EVE, a novel pattern pruning based framework that generates and implements multiple hardware-friendly models with different sparsity but shared weights to adapt to the varying environment of the energy harvesting devices. The multiple shared weights models (SWM) can successfully fit within the on-chip memory budget with slightly larger size than the sum of multiple independent models. For example, if we consider three energy levels as High, Medium, and Low, then SWM should consist of three shared-weight models and holds almost the size of only one model. Since our final objective is performing environment adaptive inference, after we deploy the SWM, depending on the availability of energy, a particular fit model inside the SWM needs *extraction*. Hence, we further introduce *bit-matrix*, which expresses the applied pattern with small overhead. With the help of the *bit-matrix*, a novel model extraction algorithm is proposed that can successfully reconstruct the required individual model from the SWM during run-time.

To the best of our knowledge, this is the first hardware/software co-design attempt to adaptively configure DNN models on resource limited energy harvesting devices. The major contributions are summarized as follows:

\*Both authors contributed equally to the paper

Permission to make digital or hard copies of all or part of this work for personal or classroom use is granted without fee provided that copies are not made or distributed for profit or commercial advantage and that copies bear this notice and the full citation on the first page. Copyrights for components of this work owned by others than ACM must be honored. Abstracting with credit is permitted. To copy otherwise, or republish, to post on servers or to redistribute to lists, requires prior specific permission and/or a fee. Request permissions from [permissions@acm.org](mailto:permissions@acm.org).

ICCAD '2022, 30 October - 3 November 2022, San Diego, California, USA

© 2022 Association for Computing Machinery.

ACM ISBN 978-1-4503-XXXX-X/18/06...\$15.00

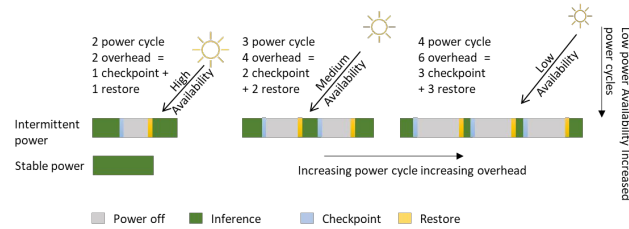
<https://doi.org/XXXXXXX.XXXXXXX>

- To satisfy real-time constraint under the varying harvesting power, we propose a shared weight training method by generating three models, which maximally mitigates the writing and reading energy consumption.
- We propose a AutoML-based framework to search for multiple compressed models with shared weights and best possible accuracies. A hardware performance predictor is developed to estimate the inference latency of models when applying different pruning patterns with different sparsity.
- We propose a run-time model extraction algorithm to reconstruct a particular model without any information loss or any significant overhead. Our algorithm can instinctively selects the candidate model for extraction, based on the energy condition of the environment.

## 2 MOTIVATION AND RELATED WORK

### 2.1 Motivation

**Why multiple adaptive models matter?** Energy harvesting (EH) that harvests energy from ambient environment has an intrinsic drawback: the harvesting power varies with the ambient environment. As a result, the same computation task may take different time to complete under different harvesting environment. Figure 1 illustrates three different DNN inference scenarios when the energy availability level is high, medium, and low respectively. During a power cycle, an energy harvesting device has 4 states including DNN inference, checkpoint, power off, and restore. Compared with high available energy environment, it takes more power cycles for the DNN inference to complete when there is low available energy due to frequent power failures, high checkpoint/restore overhead, and slow charging rate, resulting in degraded QoS and even failure in meeting the time requirement. To meet the time requirement, a promising solution is switching to a low latency model with slightly lower prediction accuracy when there is low available energy. If there are multiple models to choose from, then we can dynamically switch the DNN models with different latency and energy consumption according to the energy levels to accommodate the varying environment during run-time.



**Figure 1: Illustration of required power cycles under three energy harvesting scenarios**

To meet the time requirement, the accuracy can be slightly compromised to reduced the computation latency. As we know that the lowest latency and highest accuracy is hard to achieve simultaneously, a trade-off point is needed to achieve the best possible accuracy while satisfying the QoS for a specific environment.

**Memory is a major concern for multiple models.** Multiple models need to be deployed into the same resource limited device to adapt to the environment. This is impractical for most of the energy

harvesting devices because of scarce resource where a single model needs rigorous compression to deploy. Given aforementioned conditions, the question of how to afford multiple model at a time in a single EH devices arises. In this research, we explore pattern-based pruning for designing *SWM* architecture. This paper attempts to achieve maximum commonality of multiple pruned DNN models with different sparsities and the best possible accuracy.

### 2.2 Related work

For hardware-friendly weight pruning, PatDNN [7] proposed a real-time DNN execution on mobile devices, which applied architecture-aware optimizations to fine-grain pruning patterns for the neural network. SMOF [8] put more effort into reducing kernel size and the number of filter channels to overcome fixed-size width constraints in SIMD units. For AutoML framework on edge devices, [9] combined hardware and software reconfiguration through reinforcement learning to explore a hybrid structured pruning for Transformer. Similarly, [10] designed an algorithm-hardware closed-loop framework to efficiently find the best device to deploy the given transformer model.

With the advancement of CNN and IoT, several works have been proposed to implement CNN on IoT devices. SONIC is an intermittence-aware software system with specialized support for DNN inference [11]. ACE is the accelerator based fast intermittent DNN inference on EH device [6]. NeuroZERO introduces a co-processor architecture consisting of the main microcontroller that executes scaled-down versions of a (DNN) inference task [12]. A software/hardware co-design technique that builds an energy-efficient low bit trainable system is proposed in [13]. TF-Net pipeline efficiently deploys sub-byte CNNs on microcontrollers [14]. Different from the existing works, this paper considers environmental changes of harvesting power for the first time and configures multiple models at a time on energy harvesting (EH) devices. To the best of our knowledge, this is the first work which deploys multiple DNN models within a single-model budget while enabling adaptive inference on EH devices.

## 3 HARDWARE-AWARE SHARED-WEIGHT MODELS SEARCH FRAMEWORK

Fig. 2 shows the overview of the proposed co-design framework, to satisfy real-time constraint under varying harvesting power. Given the hardware resources and the latency ( $L$ ) and accuracy ( $A$ ) constraints as inputs, the proposed AutoML search algorithm uses a RNN-based reinforcement learning (RL) controller to guide searching the best set of shared-weight compressed models starting with a backbone model.

In the framework, the AutoML first generates  $N$  pruning pattern samples ( $P$ ) under different sparsity. We then abstract a hardware performance predictor to estimate the inference latency of the  $N$  models that applied the sampled pruning patterns on the given hardware. Then a hierarchical shared weight training architecture is designed to train these  $N$  compressed models. These  $N$  models are under different compression ratios, which can be deployed to the different energy levels of the low-power energy devices. The  $N$  models with shared weights are then fine-tuned for resource

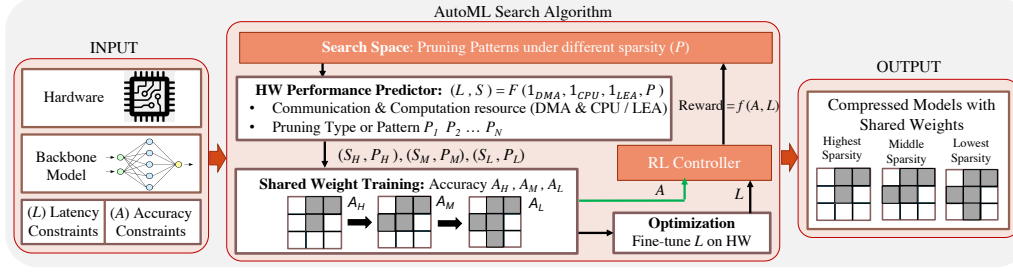


Figure 2: Overview of the Proposed Framework

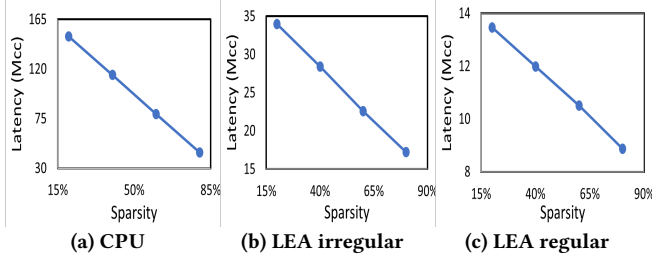


Figure 3: Latency Predictor Graph

allocation and are optimized for the latency under the given hardware constraints and restrictions. Different rewards are given under different constraints satisfaction situations. The controller is then updated based on the feedback (reward) and predicts a better pruning pattern set. In the following subsections, we will introduce each component in Fig. 2.

### 3.1 Hardware Performance Predictor

We develop a hardware performance predictor to estimate the inference latency of models when applying different pruning patterns with different sparsity. For each single DNN layer, we derive the sparsity vs latency curve by implementing different combinations of patterns that varies with sparsity and the hardware computation resources. Figure 3 shows an example curve of the relation between latency and sparsity when different computing units are utilized for the DNN inference.

Based on the obtained curves, we produce latency-profiler functions for regular pattern pruning and irregular pattern pruning respectively. Execution with low energy accelerator(LEA) is faster when the applied patterns are regular. However, in CPU based inference, execution speed does not depend on pattern regularity since CPU does not perform bulk computation as LEA. Regular patterns lower the data movement cost resulting in reduced latency compared with irregular patterns.

The reason we develop the latency-profiler function is because there are too many patterns with different sparsity and shape combination. Implementing all of these combination is extremely time expensive and pointless since a simple linear regression can closely predict the latency. Therefore, our performance predictor takes pruning patterns as input, analyzes layers in the given backbone model, and uses the latency-profiler functions to estimate latency of the entire compressed model.

### 3.2 Shared Weight Training

We can dynamically switch the DNN models (*with different sparsities but shared weights*) according to different energy levels. Different sparsities (e.g., high, medium, low) can always enable energy devices to operate even when energy level is low, while the shared weights can minimize the writing overhead when switching to a different model. Overall, we prolong the "working efficiency" of the self-powered devices on the intermediate power trace, shown in Fig. 4.

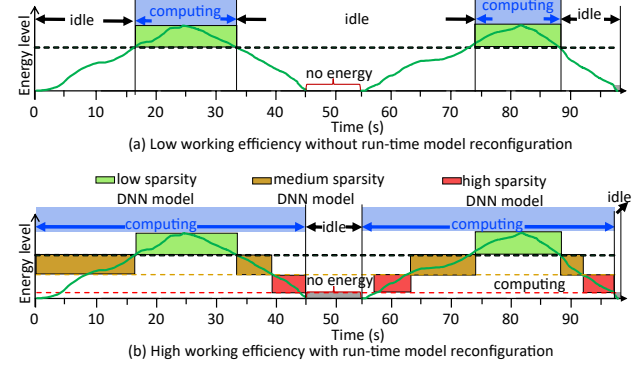


Figure 4: Run-time dynamic model switch under Intermittent power trace in energy harvesting devices.

**Shared weights to mitigate writing overhead.** An intuitive solution is to directly train several DNN models with different sparsities (e.g., high, medium, low). However, when we switch different DNN models according to different energy level, there will be a large amount overhead due to the weight re-written on memory. For example, writing 1.3MB weights consumes 1second and 1.51mJ energy. To address the overhead, we will generate different models with shared weights, as illustrated in Fig. 5. The white cells in the figure represent 0s or pruned weights. The grey cells represent unpruned weights. Orange, green and blue cells are remaining weights after pruning. The cells in the same green or red color stand for shared weights. When switching between models, we only need to update partial weights, and thus we reduce the writing overhead caused by dynamic model switch under the intermittent power trace.

In our shared-weights training (for demonstration, we use three models), the weights of the pruned layer are:

- *HighSparsity* :  $W_h = W_{Initial} \times M_h$

- *MediumSparsity* :  $W_m = W_{Initial} \times M_{hm} + W_h$
- *LowSparsity* :  $W_l = W_{Initial} \times M_{hm} + W_m$

In the equations shown above,  $W$  denotes the weight matrix,  $M$  represents the mask. The *footnote* stands for the percentage of sparsity.  $h$ ,  $m$ ,  $l$  and  $hm$  stand for *high*, *medium*, *low* and *slightly higher than medium* sparsity, respectively. It will be illustrated further in the Fig. 5, in which the shared-weights training workflow is illustrated.

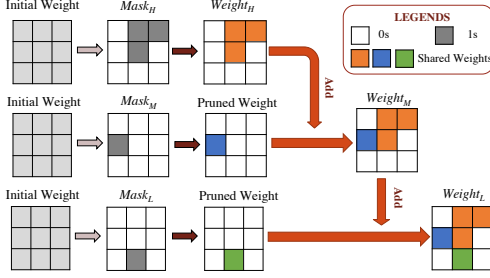


Figure 5: Shared-Weights Training Workflow

The shape of the pattern needs to be selected carefully. Since the central elements of a kernel weighs more importance [7], to guarantee accuracy, we try to keep the central elements as much as possible. Besides, the number of applied patterns also need to be selected carefully since it has to fit into small memory.

### 3.3 AutoML Search Algorithm

In our design, our search space includes three levels: sparsity level, pruning type level, and pruning pattern level. The search space is very big and it will contain many combinations if we simultaneously search for  $N$  different pruning patterns. In this case, we exploit reinforcement learning to guide the search. We use the RNN from [15] to implement our RL controller and leverage the idea from [10] to design our RL algorithm.

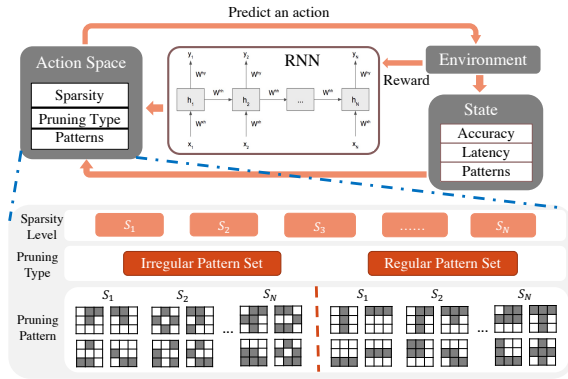


Figure 6: RL Workflow. Action space includes pruning patterns under different sparsity. The environment is composed of a HW performance predictor and a shared weight training as shown in Fig. 2

As shown in Fig. 6, in each episode, the controller firstly predicts  $N$  pruning patterns from the search space for  $N$  energy levels. The predicted patterns can be regarded as actions. Then the  $N$

pruning patterns are fed to the environment for evaluation. The environment mainly contains three modules. The HW performance predictor module takes the  $N$  patterns as input, analysis the sparsity of each pattern, estimates the latency for models when being compressed using the sampled patterns, and verifies whether the latency constraints can be satisfied. The shared weight training module takes the sparsity and patterns as input, train on the backbone model to obtain accuracy for all the  $N$  patterns on a hold-out dataset, and also estimate whether the accuracy constraints can be satisfied. *Reward* will be calculated based on the feedback from the environment. The parameters in the RNN will be updated using the Monte Carlo policy gradient algorithm [16] during this period, as follows Eq. 1.

$$\nabla J(\theta) = \frac{1}{K} \sum_{k=1}^K \sum_{s=1}^S \beta^{S-s} \nabla_{\theta} \log(a_s | a_{(s-1):1}, \theta) (R - r) \quad (1)$$

Here,  $\theta$  is the parameters in the RNN,  $K$  is the batch size,  $S$  is the total number of steps in each episode.  $\beta$  is the exponential factor to adjust the reward  $R$  at every step, and the baseline  $r$  is the average exponential moving of rewards. With the obtained information, we can formulate the reward function as Eq. 2.

$$R = \begin{cases} A + \frac{L_C - L}{L_C} & L < L_C, A > A_C \\ -\phi_P & P \text{ not satisfied} \\ -\phi_A \text{ or } -\phi_L & \text{otherwise} \end{cases} \quad (2)$$

Here,  $A$  is the lowest accuracy and  $L$  is the largest latency of the  $N$  models for the  $N$  energy levels,  $L_C$  and  $A_C$  are the given latency and accuracy constraints respectively,  $P$  is the predicted pruning pattern, and  $\phi$  are predefined numbers that represent constant penalty. Three cases will occur when calculating the reward  $R$ . (1) if  $L < L_C$  and  $A > A_C$ , it indicates that the performance of the models with the sampled patterns can satisfy the constraints, then we sum up the reward of hardware performance and accuracy; (2) if the predicted patterns for the  $N$  energy levels are the same or under the same sparsity, then the pattern constraint is not satisfied, so we return negative value  $-\phi_P$  to the controller; (3) in any other cases, which means either latency constraint or accuracy constraint is violated, we also return negative values to the controller. Different penalty are given to guide the search:  $-\phi_A$  is set for  $L < L_C$  &  $A < A_C$ ,  $-\phi_L$  is set for  $L > L_C$  &  $A > A_C$ .

## 4 ON-DEVICE ARCHITECTURE OF SHARED-WEIGHTS MODELS

In this section, we will discuss on-device implementation of the SWM. In pattern based pruning, the pruned information are unnecessary to keep in the weight matrix. Therefore, the condensed weight matrix, consisting only unpruned data can achieve significant memory cutback. Traditionally, sparse matrix and its condensation requires storing other information like (row, column, stride, offset etc) [7]. Storing such information introduces extra storage requirement except for the CNN weight matrix. In this work, since we are working on pattern-based pruned CNN, we only store the location-index of the applied pattern, which allow us to significantly reduce the memory overhead required in the process of reconstructing the patterns. As a result, the SWM found from the

AutoML search and shared weight training is further compressed and deployed on device. During inference, we perform the specific model extraction by extracting the condensed weights from the SWM. The condensed weights do not require reconstruction, since we extract the corresponding input by applying the pattern associated with the kernel and perform convolution operations as shown in figure 9. To accelerate the computation we used on-device LEA accelerator and DMA data transfer.

## 4.1 Multi-model Inference

Pattern based pruning opens up the opportunity to deploy multiple models which has uniform objective but differs only in sparsity. In DNN, low sparsity and high sparsity corresponds to the low and high pruning respectively. In Figure 7, green model has low sparsity while blue model has high sparsity. Conventionally, the DNN has the accuracy vs speed trade-off and which is why the sparsity differences among those models is the key to speed up the inference or improve the accuracy. For example, a low sparsity model provides higher latency (slow speed) and higher accuracy. However, when the sparsity increases, the latency (speedup) and accuracy decreases. This relationship is a key for the adaptive inference.

**4.1.1 Weight Sharing.** To deploy multiple models with different levels of sparsity, our framework ensures two things. First, the autoML search and training process ensures that all the models have shared weight, which is discussed in Section 3.2. Second, our framework attempt to maximize the commonality or shared weight among all the different sparsity models by approaching the idea that high sparsity models are the subset of low sparsity model. For instance, if we design three models (A,B,C) where the applied patterns are (P,Q,R) such that  $P > Q > R$  in terms of sparsity. Our aim is to maximize the commonality among P,Q and R by minimizing the symmetric difference such that  $minimize(P \oplus Q \oplus R)$ , so that  $A > B > C$  in terms of inference speed. Thus, in ideal scenarios, large sparsity patterns becomes the subset of the small sparsity patterns such that  $P \subset Q \subset R$ . The ideal scenario is preferable because it allow us more compression and low extraction complexity. However, not every kernel of the models can acquire such property since there is an accuracy-pattern trade-off. Once such models are achieved after the sequential shared-weight training, we compress and deploy the SWM into the device.

**4.1.2 Weight Compression.** We compressed all the shared-weight models by totally eliminating the pruned information from the weight matrix as shown in Figure 7. We take the unified set of weights and store them in a 1D array instead of a 2D matrix. Finally, additional indexes (location-index) are also added with every kernel from the condensed weight referring to the address of the applied pattern.

Besides, the applied patterns are obtained from the pattern search space during autoML search discussed in section 3.3. The number of the applied patterns are small and constant to a given model (e.g., 3-6). It takes only few bytes of memory since patterns can be encoded with *bit-matrix*. A *bit-matrix* is the representation of the applied patterns by 0 and 1. Here, 0 represents the pruned information where 1 doesn't. The *bit-matrix* are deployed on device with the SWM. Thus, SWM can achieve approximately slightly larger than

single model memory space even though there are multiple models.

Figure 7 demonstrates the compression and extraction process with three different sparsity models. *Note that, this process also work when there are more than three models.* Here, green (C), orange (B) and blue (A) represents low (R), medium (Q) and high (P) sparsity model respectively. In this particular example, Q is a subset of R but P is not the subset of Q or R since they do not share the weight 6. Different kernel can also have different shape of sparsity patterns.

**4.1.3 Weight extraction.** To adapt to different environment, a particular model needs to be extracted from the SWM. Therefore, we perform model reconstruction by extracting the model specific weights from the SWM. Algorithm 1 and Figure 8 describes the model specific weight extraction process. The extracted weights are in condensed format ( $W_R$ ) as shown in Figure 7. The extraction process has three elements 1) SWM ( $W_S$ ) 2) Desired model pattern ( $P_D$ ) 3) Other shared model pattern ( $P_O$ ). Figure 8 shows that the algorithm includes three basic steps 1) Take 2) Skip and 3) Do nothing.

**Initial Setup:** The algorithm will operate those steps by iterating through the desired model pattern first. A source pointer points to the first element of the SWM.

**Take:** While iterating through the desired pattern  $P_D$ , a value of 1 means it will take the element that points to the SWM. Next, the source pointer will move to the adjacent element. In Algorithm 1, line 4-7 perform this step.

**Skip:** However, if the value of the desired pattern,  $P_D$  is 0, there are two other cases. In first case, the algorithm will look into other shared-model patterns,  $P_O$ . If any of the other patterns has a value of 1 within same spot, then the source pointer will move to the next element but nothing will be taken from the SWM. In Algorithm 1, line 9-12 perform this step.

**Do Nothing:** In last case, the value of the desired pattern,  $P_D$  is 0 and all the other shared-model pattern,  $P_O$  has a value of 0. Therefore, the source pointer will not be moved and nothing will be taken from the SWM.

Figure 8 shows an example where a high sparsity model is extracted from the SWM. In this particular example, the SWM is comprised of three different sparsity models. Similarly, low and medium sparsity model can also be extracted by selecting the low sparsity pattern and medium sparsity pattern as desired pattern ( $P_D$ ) respectively. The Algorithm 1 describe the whole process. Figure 7 shows the extracted condensed weights that represents the individual sparsity model.

**4.1.4 Convolution.** We perform convolution operation with the extracted condensed weight during inference. Since the weights are condensed, we extract the corresponding input from the input window by applying the same pattern associated to the condensed weight as shown in Figure 9. Finally, the multiplication results are accumulated to the output.

**4.1.5 Computation.** In on-device CPU based implementation, the multiply and accumulation operation take place with single element by checking the pattern value. However, the on-device low energy accelerator (LEA) can expedite the computation by bulk operation. For a particular CNN computation window, the extracted



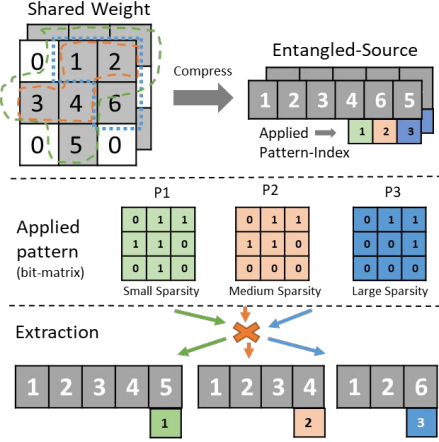


Figure 7: Shared weights Compression and Extraction

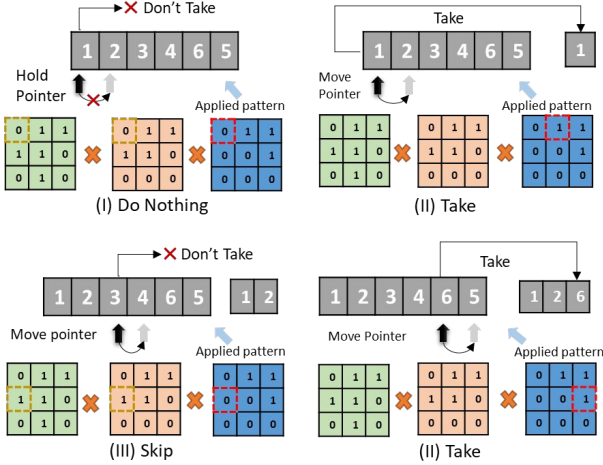


Figure 8: Weight Extraction Example

**Algorithm 1** Weight Extraction Algorithm

```

1: Input:  $W_S, P_D, [P_O]$ 
2: Output:  $W_R$ 
3:  $i, j \leftarrow 0$ ;
4: while  $p_d \in P_D$  do
5:   if  $p_d$  is 1 then
6:      $W_R[j] \leftarrow W_S[i]$ 
7:      $i++$ ;  $j++$ ;
8:   else
9:     while  $p_o \in [P_O] \mid p_o, p_d$  with same index, do
10:      if  $p_o$  is 1 then
11:         $i++$ ;
12:      break;
13:    end if
14:  end while
15: end if
16: end while
    
```

condensed weight and the corresponding input values are transferred for computation. Since direct memory access (DMA) is a reliable and faster data transfer method, we use DMA to transfer these data from non volatile memory (NVM) to volatile memory (VM). Finally, we invoke the MAC API from DSPLIB to perform the bulk computation. As the device is prone to frequent power failure, we implemented the inference with index based checkpointing [6] method so that the inference can resume when power restores.

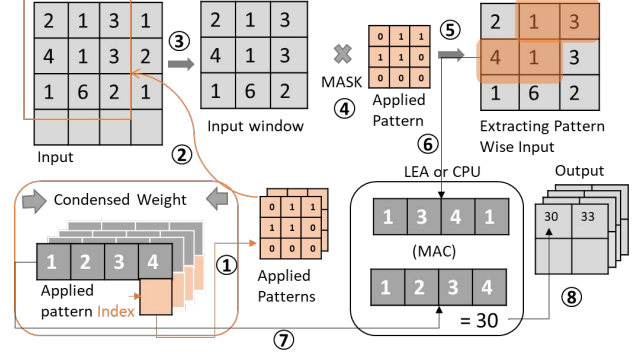


Figure 9: Convolution with Compressed Weight

**4.1.6 Fully Connected Layer.** In fully connected (FC) layer the weight-sharing and extraction process is almost similar to Convolutional layer except the fact that FC layer weights are usually large 2D matrix where convolutional weights are 3D. However, the same shapes of patterns are applied to the large 2D weight. Thus, we divide the large 2D weight matrix into several blocks and performed similar process on these blocks as discussed above. For instances, in a FC layer if a weight matrix shape is  $m \times n$  and applied pattern shape is  $x \times y$ , then there will be  $m/x \times n/y$  number of blocks and every block of matrix will follow the similar compression, extraction and computation process.

## 4.2 Adaptive Inference

Our framework is capable of switching *instantly* between different sparsity models on run-time. The model switching is performed instinctively by understanding the environment and executing the model that is designed for that specific environment. To understand the environment we designed *energyTraker*, a simple software solution to track the energy availability. This is implemented with a timer that recurrently checks the voltage level of the device in parallel with the computation. A very low voltage level is recognized as a possible power outage, thus, calculates the current power cycle time. Similarly, we detect the last three power cycle time and based on the pre-defined threshold value, we classify the current environment as high, medium, low and adapt with the suitable model for the next inference.

## 4.3 Generality

Although we demonstrated our work with three different sparsity models, our proposed framework can achieve an arbitrary number of shared-weights models. However, it is important to consider that increased shared-weight models will worsen the overhead since the escalating pattern information (Applied Pattern, Pattern-Index) will require more memory footprint and also exacerbate the model extraction complexity. Even though our experiment is conducted with convolutional neural network (CNN), this approach can be applied to other network such as RNN, transformer or any attention based network. Since these network consist of large 2D weight matrix, similar process can be followed as for FC layer.

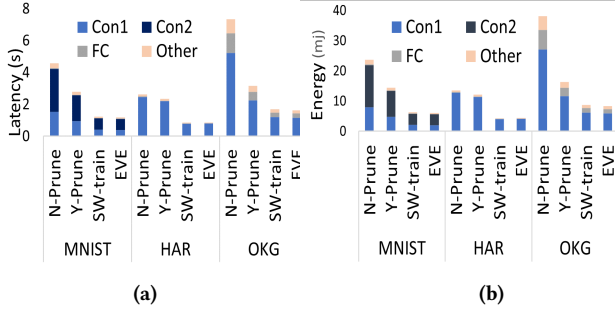


Figure 10: (a) Latency and (b) Energy under continuous power.

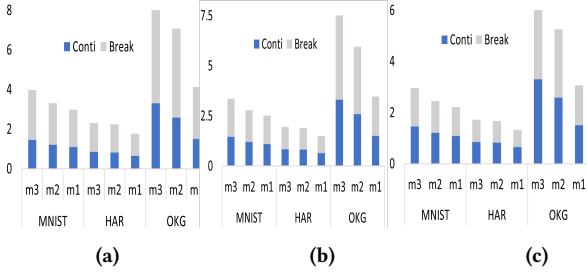


Figure 11: (a) Latency under 3mW (b) Latency under 4mW. (c) Latency under 5mW

## 5 EXPERIMENTAL EVALUATION

### 5.1 Hardware Setup

The experiments are conducted with TI's MSP430FR5994 ultra-low-power evaluation board, consisting of a 16 MHz MCU, a 8KB volatile SRAM, a 256KB nonvolatile FRAM memory, and a low-energy Accelerator (LEA) that supports independent vector operations such as FFT, IFFT, MAC, etc. The board is powered by a energy harvesting module composed of a function generator SIGLENT SDG1032X [17], a power regulator Bq25570, and an energy buffer (100μF capacitor). The function generator is used to simulate different energy harvesting power sources, which are insufficient for completing a single inference while causing frequent power failures. The power regulator provides a constant voltage of 3.3V for the normal operation of the board. TI EnergyTrace tool is used to measure energy consumption [18].

**DNN Models:** This paper considers four DNN models as shown in Table 1. They are Image Classification (MNIST [19] and ImageNet [20]), Human Activity Recognition (HAR [21]), and Google Keyword Recognition (OKG [22]) to represent image-based applications, wearable applications, and audio applications respectively.

### 5.2 Experimental Results

#### 5.2.1 Comparison of Shared Weight Training and AutoML Search:

In Table 1, the row SW-Train shows the accuracy of different sparsities after shared-weights training. We can tell from the table that shared-weights training can help retain weights among different sparsities to have them shared with a comparable accuracy. The patterns applied on the weight matrix are selected manually from the pattern searching space according to the rule proposed in the PCONV paper[25]. As for LeNet5, the high sparsity is 60.24% and the corresponding accuracy is 98.94%. The medium Sparsity is 20.093% and its accuracy is 98.96%. The low sparsity is 8.47% and its accuracy

is 99.02%. In order to ensure "shared weight" as illustrated in Figure 5, we try to ensure that there is no overlap of rows or columns between patterns for models under different sparsity level when manually selecting those patterns. Apart from image classification tasks, our shared-weights training shows satisfying accuracy on recognition tasks as well. For example, as for the HAR-Net model, the high, medium and low sparsities are 24.13%, 5.31% and 2.69%, respectively, and their corresponding accuracy are 90.19%, 89.78% and 87.21%. Comparably, the accuracy and sparsity of the OKG-Net show the same trend as HAR-Net. Thus, with sacrificing little accuracy drop, the shared-weights training algorithm could keep weights unchanged, and further benefit the inference process on energy harvesting devices.

The second row (EVE) in Table 1 shows the accuracy and model latency under three different sparsities after shared-weights training, with the three pruning patterns are automatically selected by the AutoML search algorithm. For LeNet-5 and OKG-Net, as there are totally 44 patterns in the pattern space, the total dimension of the search space for the three models is  $44 \times 44 \times 44 = 85184$ . For HAR-Net, as there are totally 21 patterns in the pattern space, the total dimension of the search space for the three models is  $21 \times 21 \times 21 = 9261$ . For SqNxt-23, as there are 36 patterns in the pattern space, the total dimension and the total of the search space is  $36 \times 36 \times 36 = 46659$ .

These action spaces are too large for human selection. That is why we use a Reinforcement Learning algorithm to design the AutoML search. Our RNN-based agent has one layer with 35 hidden units and a fully-connected layer with dimension-width size, then finally after a softmax function. The agent can choose the corresponding pattern with the highest probability. The batch size for updating the RNN-agent is 1. Thus, after the agent gives an output action, the RNN agent would be updated by the policy gradient algorithm. When the models with selected three patterns satisfy the constraints of both accuracy and latency, the searching algorithm will return the models and the selected three pruning patterns. If not, the agent will continue exploring until 300 episodes.

**5.2.2 Inference Time under Continuous Power Supply:** To fairly evaluate the performance of the proposed methods demonstrate the performance of having 3 models with shared weights on the device. Since to the best of our knowledge, shared weight training is a new idea that has not been implemented before on energy harvesting (EH) device, we first compared the inference latency of executing 3 generated models with shared weights of the two proposed methods (SW-train and EVE) in a sequence with two baselines for each benchmark. Baseline 1 is named as N-prune, where each backbone model without pruning are stored on the off-chip memory and have to be loaded into the on-chip memory for execution. Baseline 2 is named Y-prune, where we can only fit one pruned backbone model into the on-chip memory. We can observe from the Figure 10 that without using any pruning method, executing a DNN backbone model costs extremely long latency of 4s, 2.5s, 8s respectively. Y-prune generates three pruned models but cannot fit them all into the on-chip memory, and thus require less time than SW-train but more time than SW-train and EVE. We can observe from the Figure 10(a) that SW-train and EVE run 2.5× and 2.2× faster than the two baselines.

Dataset/Task		MNIST (Image Classification)			HAR (Human Activity Recognition)			OKG (Speech Recognition)			ImageNet (Image Classification)		
Models		LeNet-5 [23]			HAR-Net [21]			OKG-Net [22]			SqNxt-23 [24]		
		M1	M2	M3	M1	M2	M3	M1	M2	M3	M1	M2	M3
SW-Train	Sparsity	60.24%	40.50%	20.70%	24.13%	5.31%	2.69%	33.33%	5.88%	5.02%	55.39%	54.30%	54.07%
	Accuracy	98.94%	98.96%	99.02%	87.21%	89.78%	90.19%	71.99%	73.82%	75.20%	73.80%	75.02%	77.36%
	Latency (s)	1.1	1.3	1.5	0.66	0.83	0.85	2.26	3.19	3.23	3.4	3.6	3.7
EVE	Sparsity	60.24%	48.00%	35.76%	24.19%	5.37%	2.69%	55.56%	25.85%	4.17%	57.15%	35.24%	1.41%
	Accuracy	99.04%	99.14%	99.11%	92.4%	92.39%	89.03%	72.34%	73.17%	74.73%	77.13%	77.87%	79.34%
	Latency (s)	1.1	1.23	1.45	0.65	0.81	0.85	1.51	2.59	3.31	3.4	3.5	3.6
Accuracy gap		0.1%	0.98%	0.09%	2.21%	2.61%	1.82%	0.52%	0.24%	0.6%	3.33%	2.85%	1.98%
Latency gap (s)		0.00	0.07	0.05	0.01	0.02	0.00	0.01	0.00	0.00	0.00	0.10	0.10

Table 1: Results and Comparison between Shared Weight Training and EVE

**5.2.3 Energy Consumption under Continuous Power Supply:** In terms of energy consumption evaluation, SW-train and EVE also outperform the two baselines. As shown in Figure 10(b) that, SW-train and EVE achieve 2.5× and 2.77× energy-saving on MNIST; 2.8× and 3.1× energy-saving on HAR. And finally, 3.2× and 3.3× energy is saved on OKG. Having shared models significantly reduced the energy for switching models from the off-chip memory.

**5.2.4 Inference Time under Intermittent Power Supply:** To evaluate the performance of 3 models with shared weights generated with the proposed SW-Train and EVE methods, we use the function generator SIGLENT SDG1032X to generate different levels of harvesting power including 5mW, 4mW, 3mW as the high, medium, and low energy levels. The three harvesting power are smaller than the working power of the energy harvesting device. Thus the device has to accumulate energy first before starting each working cycle of inference. Figure 11 shows the inference latency when the harvesting is 5mW, 4mW, 3mW respectively for each of the three models with shared weights. From the three figures, we observe that, when the harvesting power is low, the execution time is significantly increased. Therefore, in order to meet the QoS requirement, the energy harvesting device needs to automatically switch to the low-accuracy model which takes much less time.

**5.2.5 Evaluation of Weight Reconstruction and Memory Overhead.** The weight reconstruction from the shared three models stored in on-chip memory brings 1 to 3% overhead to the overall inference for each benchmark which is negligible considering the significantly improved overall performance.

## 6 CONCLUSION

In this work, we propose EVE, a novel pattern pruning based framework that generate multiple hardware friendly models with shared weights, to dynamically satisfy the **real-time** requirement on the energy harvesting devices. We develop an AutoML-based co-exploration framework to search the desired multi models with shared weights meanwhile satisfying both the accuracy and latency constraints. The generated models can successfully fit within the on-chip memory budget of slightly larger than one model instead of **three** multiple models. We develop an efficient on-device implementation architecture to efficiently execute each model on device. Experimental results show that our design is on average 2.5× time faster than the baseline model without pruning and shared weights. **The** EVE can further achieve 1.2% higher accuracy with higher sparsity compared to human based shared weight search. It

is worth noting that, the proposed EVE model can be easily adapted to generate more than three shared models to adapt to more energy harvesting power levels.

## ACKNOWLEDGEMENT

The authors would like to acknowledge the USDOT Transportation Consortium of the South-Central States (TRAN-SET) (# 21-034 to YFJ and MX, and #21-049 to YFJ and MX), National Science Foundation (EEC-2051113 to YFJ, CCF-2011236 to WW, and CCF-2006748 to WW), USDA-NIFA Agriculture and Food Research Initiative (Award No.: 2022-67023-36399 to CD) for the funding and necessary support in completing the research. The funding sources had no role in the design of the study; collection, analysis, and interpretation of data; or in writing the manuscript.

## REFERENCES

- [1] C. Garvey, "A framework for evaluating barriers to the democratization of artificial intelligence," in *AAAI*, 2018.
- [2] E. Li and et.al., "Edge ai: On-demand accelerating deep neural network inference via edge computing," *IEEE-TWC*, vol. 19, no. 1, pp. 447–457, 2019.
- [3] K. He and et.al., "Deep residual learning for image recognition," in *CVPR*, pp. 770–778, 2016.
- [4] J. Devlin and et.al., "Bert: Pre-training of deep bidirectional transformers for language understanding," *arXiv preprint arXiv:1810.04805*, 2018.
- [5] J. Banerjee, S. Islam, W. Wei, C. Pan, D. Zhu, and M. Xie, "Memory-aware efficient deep learning mechanism for iot devices," in *2021 IEEE 32nd International Conference on Application-specific Systems, Architectures and Processors (ASAP)*, pp. 187–194, IEEE, 2021.
- [6] S. Islam, J. Deng, S. Zhou, C. Pan, C. Ding, and M. Xie, "Enabling fast deep learning on tiny energy-harvesting iot devices," in *2022 Design, Automation Test in Europe Conference Exhibition (DATE)*, pp. 921–926, 2022.
- [7] 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 *Proceedings of the Twenty-Fifth International Conference on Architectural Support for Programming Languages and Operating Systems (ASPLOS)*, pp. 907–922, 2020.
- [8] Y. Liu and et.al., "Smof: Squeezing more out of filters yields hardware-friendly cnn pruning," *arXiv preprint arXiv:2110.10842*, 2021.
- [9] Y. Song and et.al., "Dancing along battery: Enabling transformer with run-time reconfigurability on mobile devices," *arXiv*, 2021.
- [10] P. Qi and et.al., "Accelerating framework of transformer by hardware design and model compression co-optimization," *arXiv*, 2021.
- [11] G. Gobieski and et.al., "Intelligence beyond the edge: Inference on intermittent embedded systems," in *ASPLOS*, pp. 199–213, 2019.
- [12] S. Lee and S. Nirjon, "Neuro. zero: a zero-energy neural network accelerator for embedded sensing and inference systems," in *ENSS*, pp. 138–152, 2019.
- [13] S. Choi et al., "An optimized design technique of low-bit neural network training for personalization on iot devices," in *DAC*, pp. 1–6, 2019.
- [14] J. Yu and et.al., "Tf-net: Deploying sub-byte deep neural networks on microcontrollers," *ACM TECS*, vol. 18, no. 5s, pp. 1–21, 2019.
- [15] B. Zoph and Q. V. Le, "Neural architecture search with reinforcement learning," *arXiv preprint arXiv:1611.01578*, 2016.
- [16] L. Yang and et.al., "Co-exploration of neural architectures and heterogeneous asic accelerator designs targeting multiple tasks," in *DAC*, pp. 1–6, IEEE, 2020.
- [17] "Sdg1032x," <http://siglentna.com/product/sdg1032x/>, 2021.



- [18] TI, "Energy trace," 2021. <https://www.ti.com/tool/ENERGYTRACE>.
- [19] Y. LeCun and C. Cortes, "MNIST handwritten digit database," 2010.
- [20] J. Deng, W. Dong, R. Socher, L.-J. Li, K. Li, and L. Fei-Fei, "Imagenet: A large-scale hierarchical image database," in *2009 IEEE Conference on Computer Vision and Pattern Recognition*, pp. 248–255, 2009.
- [21] D. Anguita and et.al., "A public domain dataset for human activity recognition using smartphones,," in *Esann*, vol. 3, p. 3, 2013.
- [22] P. Warden, "Speech commands: A dataset for limited-vocabulary speech recognition," *ArXiv*, vol. abs/1804.03209, 2018.
- [23] Y. LeCun and et.al, "Gradient-based learning applied to document recognition," *Proceedings of the IEEE*, vol. 86, no. 11, pp. 2278–2324, 1998.
- [24] A. Gholami and et. al., "Squeezenext: Hardware-aware neural network design," in *CVPR*, pp. 1638–1647, 2018.
- [25] X. Ma and et.al., "Pconv: The missing but desirable sparsity in dnn weight pruning for real-time execution on mobile devices," *ArXiv*, 2020.