Neuro.ZERO: A Zero-Energy Neural Network Accelerator for Embedded Sensing and Inference Systems

Seulki Lee
University of North Carolina at Chapel Hill
Chapel Hill, North Carolina
seulki@cs.unc.edu

Shahriar Nirjon
University of North Carolina at Chapel Hill
Chapel Hill, North Carolina
nirjon@cs.unc.edu

ABSTRACT

We introduce *Neuro.ZERO*—a co-processor architecture consisting of a main microcontroller (MCU) that executes scaled-down versions of a deep neural network[1] (DNN) inference task, and an accelerator microcontroller that is powered by harvested energy and follows the intermittent computing paradigm [76]. The goal of the accelerator is to enhance the inference performance of the DNN that is running on the main microcontroller. Neuro.ZERO opportunistically accelerates the run-time performance of a DNN via one of its four acceleration modes: *extended inference*, *expedited inference*, *ensemble inference*, and *latent training*. To enable these modes, we propose two sets of algorithms: (1) *energy and intermittence-aware DNN inference and training algorithms*, and (2) *a fast and high-precision adaptive fixed-point arithmetic* that beats existing floating-point and fixed-point arithmetic in terms of speed and precision, respectively, and achieves the best of both. To evaluate Neuro.ZERO, we implement low-power image and audio recognition applications and demonstrate that their inference speedup increases by 1.6× and 1.7×, respectively, and the inference accuracy increases by 10% and 16%, respectively, when compared to battery-powered single-MCU systems.

CCS CONCEPTS

• **Computer systems organization** → **Neural networks**; • **Hardware** → **Hardware accelerators**; **Power and energy**.

KEYWORDS

Accelerator, Deep neural networks, Zero energy, Batteryless

ACM Reference Format:
Seulki Lee and Shahriar Nirjon. 2019. Neuro.ZERO: A Zero-Energy Neural Network Accelerator for Embedded Sensing and Inference Systems. In *SenSys '19: Conference on Embedded Networked Sensor Systems, November 10–13, 2019, New York, NY, USA.* ACM, New York, NY, USA, 15 pages. https://doi.org/10.1145/3356250.3360030

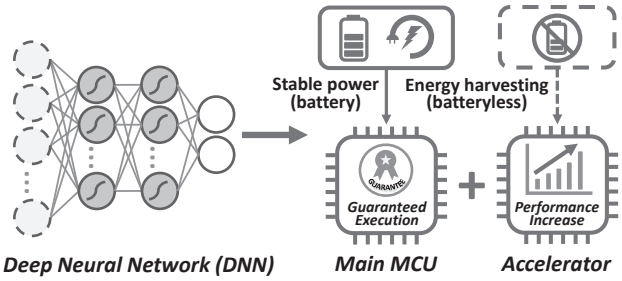


Figure 1: Neuro.ZERO: The batteryless accelerator, powered by harvested energy, opportunistically enhances the run-time performance of DNN execution without consuming power from the main system. The main MCU (microcontroller unit) guarantees seamless execution of DNN by using a stable power source such as a battery.

1 INTRODUCTION

In recent years, deep neural networks (DNN) [69, 101] have shown stellar performance in solving problems in machine learning and related fields [13, 26, 27, 38, 49, 64, 102, 121]. Following the trend, embedded systems have started to implement lightweight versions of DNNs [33, 114, 120], primarily focused on inference or generalization tasks [66]. The de facto approach to enable deep inference on resource-constrained systems is to obtain a pre-trained model from some other sources and then to compress and/or prune the network until it fits the memory and computing capacity of the embedded platform [37, 42, 43, 80, 123]. Needless to say, such compression and pruning hacks inevitably degrade the performance, and many large-sized DNNs are quite challenging to port on resource-constrained embedded platforms even after compression and pruning.

The performance of DNNs running on an embedded system [10, 15, 33] is limited by the platform's CPU, memory, and battery-size; and their scope is limited to inference tasks only. To overcome this, special-purpose co-processors, called *DNN accelerators*, have been proposed and productized [4, 94], primarily targeted to smartphone-grade mobile systems. Although especially architected hardware in these accelerators enables faster execution of DNNs, they have some major practical limitations. First, DNN computations are power-hungry. The power consumption of these accelerators remains as a fundamental bottleneck—prohibiting them to be used in battery-powered systems. Second, existing accelerators primarily focus on speeding up the execution of an offline-trained DNN inference task. In general, there is a lack of research on how to facilitate run-time adaptation so that the inference accuracy increases over time as resources become available or newly sampled sensor data can be used to fine-tune the performance. Third, while application-specific

[1]The DNN, by definition, refers to neural networks having more than one hidden layers [23, 44, 55, 75]. Thus, a wide variety of networks qualify as a DNN in the existing literature. DNNs considered in this paper have up to 10^5 neurons and weights combined. They fit into 256KB memory of an MCU; have convolutional, ReLU, pooling, and fully-connected structures as regular DNNs; and perform on-device inference [34, 35].

SenSys '19, November 10–13, 2019, New York, NY, USA

ACM ISBN 978-1-4503-6950-3/19/11…$15.00
https://doi.org/10.1145/3356250.3360030

hardware accelerators of different types such as FPGAs and ASICs are effective, their lack of standardization, unavailability to system developers, and excessive price are slowing down the development of engineered systems that could leverage DNN acceleration in their embedded sensing and inference applications.

In this paper, we introduce *Neuro.ZERO*—a novel co-processor architecture consisting of two microcontroller units (MCUs): (1) a battery-powered main MCU that executes a scaled-down[1] DNN inference task, and 2) a batteryless (energy-harvesting) accelerator MCU that enhances the performance of DNN inference that runs on the main MCU. A high-level architectural diagram of Neuro.ZERO is shown in Figure 1. Unlike existing DNN accelerators that primarily focus on improving the inference speed [36, 112], the accelerator in Neuro.ZERO improves the run-time performance of the DNN on the main MCU by *increasing inference accuracy* or by *enabling on-device training*. Since the accelerator does not draw power from the main system, we call it a *zero-energy* accelerator. By having two MCUs, one powered by a battery and one powered by harvested energy, Neuro.ZERO guarantees sensing and inference for all sensor data while enjoying opportunistic run-time performance gain without spending system's energy. Neuro.ZERO is implemented on off-the-shelf, low-power, low-cost MCUs [107], and its source code is open [52]–which helps developers build low-power, intelligent sensing, and inference systems faster and at a lower cost.

The architecture of Neuro.ZERO falls into the general category of energy-aware heterogeneous multi-core systems such as ARM's big.LITTLE [5, 59] and application-specific systems [74, 85]. However, Neuro.ZERO takes this to an extremity where one of the cores runs completely on harvested energy. It flips a common practice of energy-aware heterogeneous multi-core systems where typically a lower-power core remains active, and it controls the sleep/wake cycles of a higher-power core based on the computational demand. Instead, a new execution paradigm is introduced in Neuro.ZERO, where the main MCU executes sensing and basic inference tasks as programmed by a developer to meet its timing and energy constraints, and when the batteryless MCU harvests enough energy to execute a task by itself, it uses up that energy to improve the main MCU's performance in executing its accelerated inference task.

The proposed zero-energy accelerator follows standard practices of intermittently-powered systems. Its core framework is built upon existing work on intermittent computing that address important problems such as atomicity [19, 79], consistency [19, 77, 79], programmability [51], timeliness [51], and energy-efficiency [12, 20, 50] to enable efficient code execution of general-purpose tasks. Neuro.ZERO complements existing literature and solves new and higher-level system challenges resulting from the heterogeneous execution pattern of Neuro.ZERO cores as well as fundamental challenges in executing accelerated inference and training on resource-constrained and intermittently-powered systems.

Neuro.ZERO opportunistically accelerates the run-time performance of a DNN via one of its four acceleration modes: *extended inference*, *expedited inference*, *ensemble inference*, and *latent training* which facilitates execution of larger sized networks, splits the given DNN for parallel execution, improves confidence of inference via ensembling [65], and updates the DNN weights via online training, respectively. To enable these modes, two sets of algorithms have been developed. First, energy and intermittence-aware algorithms have been developed that *steps-up* the DNN inference by scaling up the size of DNN based on the current energy level and *skips-out* back-propagation [117] of some weights during online training as the amount of harvested energy fluctuates at run-time. Second, a fast and high-precision adaptive fixed-point arithmetic has been proposed that beats existing floating-point and fixed-point arithmetic in terms of speedup and precision, respectively, and achieves the best of both. To demonstrate the efficacy of Neuro.ZERO, we implement two applications that use camera and microphone to recognize certain images (i.e., traffic signs) and audio events (i.e., voice commands). These systems have been tested extensively using both standard datasets, i.e., MNIST [70], CIFAR-10 [63], SVHN [86], and Fashion MNIST [118], as well as in real-world experiments.

2 OVERVIEW OF NEURO.ZERO

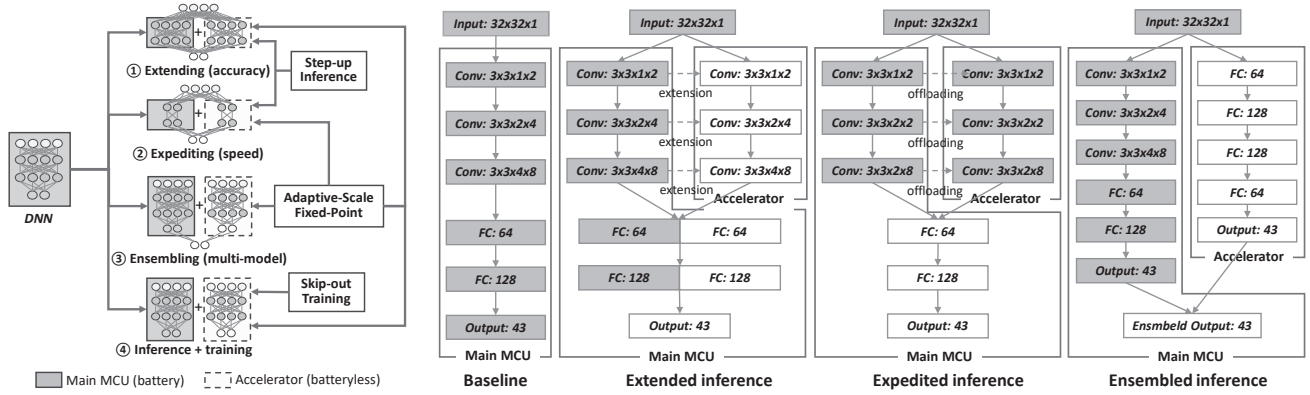
2.1 System Design

The goal of Neuro.ZERO is to increase the run-time performance of DNN on resource-constrained, MCU-based systems by having a low-power energy-harvesting MCU as an *accelerator*, that opportunistically improves the accuracy or speed of DNN inference, without drawing any power from the battery. Figure 2a shows an architectural diagram of Neuro.ZERO, which depicts how a DNN is converted to one of four different architectures at compile time. At run-time, the shaded part of the generated network run on the main MCU, while the rest run on the accelerator only when it is active. The algorithms enabling these modes are shown on the right.

Basic Working Principle. Neuro.ZERO comes with a compile-time tool and a run-time system. The compile-time tool takes a baseline DNN architecture, a training dataset, and an acceleration mode as an input. Depending on the chosen acceleration mode, Neuro.ZERO creates two network architectures, trains them using the given training dataset, and generates two DNNs as the output (one for each MCU)—which are ready to be executed on the two-MCU hardware platform designed for Neuro.ZERO. The DNN for the main MCU is generated based on the baseline DNN by appending the necessary architecture for acceleration without changing the baseline DNN. It ensures that the standalone execution of the main MCU is self-sufficient in satisfying the desired application-level performance goals (e.g., achieving the same accuracy and speed of the original baseline DNN). When the accelerating DNN is executed on the accelerator, the two networks combinedly are expected to achieve a better inference accuracy and/or speed.

The run-time system of Neuro.ZERO is responsible for executing the two DNNs by managing the coordination between the two MCUs. The run-time system also keeps track of the status of the two MCUs and provides APIs to know whether the accelerator is active or involved in the inference as well as APIs to turn ON/OFF the accelerator (e.g., for debugging or experiment purposes). To ensure the consistent execution of DNN, the accelerator executes the DNN only when the energy harvester accrues enough energy to complete a full pass of feed-forward (from input to output layer).

Four Modes of Acceleration. To improve the run-time performance of DNNs, Neuro.ZERO supports four modes of acceleration. Each mode takes advantage of the intermittently-powered accelerator in a unique manner. The *extended inference* mode improves the inference accuracy by extending the DNN's structure and running



(a) Neuro.ZERO system design (b) Examples of the three zero-energy inference acceleration: extended, expedited, and ensembled inference

Figure 2: Neuro.ZERO's four modes accelerate a DNN in terms of accuracy, speed, multi-model, and training by extending, expediting, ensembling, and training the DNN on the accelerator. They are enabled by energy-aware acceleration (step-up inference and skip-out training) and numerical acceleration (adaptive-scale fixed-point).

the extended part on the accelerator. The *expedited inference* mode increases the inference speed by offloading some part of the original DNN to the accelerator. The *ensembled inference* mode runs a different DNN model on the accelerator as a second DNN and combines the output of the two independent DNNs to increase the inference accuracy. The *latent training* mode enables an intermittent on-device training of the baseline DNN for unseen data on the accelerator while allowing the main MCU to keep executing the inference task. The details of these modes are discussed in Section 3.

Algorithms Enabling Acceleration. The four acceleration modes of Neuro.ZERO are enabled by a set of algorithms that accelerate the DNN inference and training on an intermittently-powered system and expedite floating-point arithmetic. Since the accelerator runs on sporadically harvested energy, tasks running on it execute intermittently. Such an intermittent execution pattern makes both the inference and the training of a DNN challenging. To address this, we propose two novel algorithms, namely the *step-up inference* and the *skip-out training*, which accelerate the inference and training of DNNs in proportion to the harvested energy (Section 4).

Despite these accelerations, we observe that the execution of DNN, in general, is extremely slow on low-power, low-cost MCUs that do not have hardware support for floating-point operations [2]. To address this well-known issue, most low-power embedded systems use *fixed-point* arithmetic [88], which is computationally efficient but numerically inaccurate than floating-point. In Neuro.ZERO, we rethink the implementation of fixed-point operations and propose *adaptive-scale fixed-point* number representation that provides both the numerical correctness of floating-point arithmetic and the speed-up of fixed-point arithmetic. This is described in Section 5.

2.2 Design Rationale

We compare three alternative choices of processors for the accelerator in terms of their power consumption, CPU performance (measured in Dhrystone MIPS [116]), and cost in Table 1. Considering the low price and ultra-low power consumption, an MCU is the most suitable choice for an energy harvesting system like Neuro.ZERO as they can be run intermittently on harvested energy and wake-up more frequently due to shorter charge-discharge cycles, and enable large scale deployment due to low-cost. For example, when an RF harvester [91, 92] (generating 0.2mW–2.0mW) is used, an FPGA or an SoC would take several minutes to hours to harvest enough energy before they can execute any workload. Such a long delay is not suitable for Neuro.ZERO, as the accelerator is more likely to miss sensor data during its long charging time and the value of processing the data may be lost (e.g., in time-sensitive applications) after such long delay. Although large energy harvesters and huge capacitors as energy storage could be a makeshift solution, such systems will be bulky and expensive, and thus are not suitable for most embedded sensing systems.

Accelerator/Processor Type	Power	Performance	Cost
MCU – TI MSP430 [107]	3.8-6.2mW	13 DMIPS	$3-$5
FPGA – Xilinx Spartan 6 [103, 119]	24-109mW	166 DMIPS	$30-$33
SoC – Qualcomm Snapdragon [95]	2.1-4.8W	13,860 DMIPS	$70-$199

Table 1: Comparison of processor choices for the accelerator.

Having an MCU as the choice for the accelerator, Table 2 compares four co-processor designs for up to two MCUs. The first two rows show single-MCU systems, and the rest show two-MCU systems. We observe that only when the main MCU is battery-powered, and the accelerator is energy-harvesting, we achieve seamless execution of tasks (on the main MCU) and energy-savings and increased performance (due to the energy-harvesting accelerator).

Main MCU's Power Source	Accelerator MCU's Power Source	Timely/ Seamless Execution	Energy Harvesting (Accelerator)	Performance Increase by Accelerator
Battery	-	✔	-	✘
Harvesting	-	✘	-	✘
Battery	Battery	✔	✘	✔
Harvesting	Harvesting	✘	✔	✔
Battery	**Harvesting**	✔	✔	✔

Table 2: Comparison of MCU-based architectural choices.

Extensibility and Cost. Neuro.ZERO is developed as a two-MCU system. However, its design principles and algorithms are applicable to many-MCU systems where a subset of MCUs are battery-powered, and the rest are powered by harvested energy. Having additional MCUs in a system adds a one-time cost, but considering their small form-factor and the low cost (<$5 per unit), the benefit of increased accuracy and speedup clearly outweighs the cost.

2.3 Example Application Scenarios

We describe two example applications of Neuro.ZERO: (1) a traffic sign recognizer, and (2) a voice command recognizer, which classifies traffic sign images and voice audio data, respectively. In Section 8, we describe their implementation and evaluation results.

Wearables for Pedestrian and Biker's Safety. Pedestrians and bikers are often not fully aware of their surroundings, which is causing their lives [1]. To augment perception and cognition of pedestrians and bikers, wearable systems have been proposed that recognize imminent dangers on the road, alert the user on time, and help them avoid injury and death [16, 48, 104]. We propose to augment the ability of pedestrians and bikers to see and recognize traffic signs by enabling road-sign recognition on camera-based low-power wearable systems. These battery-powered systems need to process camera images in real-time and produce accurate classification results. Using Neuro.ZERO, we can improve the accuracy and confidence, and lower the execution time of the image recognition applications for such wearable systems. As these systems are expected to be used outdoors, solar energy can be harvested to power the accelerator. In this application, Neuro.ZERO can be operated (1) in the extended inference mode when the user enters an environment that requires higher-resolution images to detect objects, (2) in the expedited mode when the user is in a busy area, (3) in the ensemble mode when there are multiple cameras or a different sensor (e.g., microphone) to independently detect the same event, or (4) in the training mode when environment-specific parameter tuning is necessary to obtain better classification results.

Voice Commands for Smarter Things. Voice-based communication with everyday objects in natural languages is becoming a reality. Today, devices like Amazon Echo acts as a "middle-man" to enable voice communication with smart devices such as home appliances, remote controllers, thermostats, light bulbs, switches, speakers, clocks, and many more. We envision that, in a few years, voice-communication capability will be directly built into every smart object. In order to realize this vision, building low-power, low-cost, MCU-grade systems that recognize voice commands are essential. Neuro.ZERO enables the development of these next-generation smart objects that are able to sense and interpret voice commands on-device and in real-time, and opportunistically improve their inference performance at runtime by leveraging the harvested power from ambient RF energy at indoor environments. In this application, Neuro.ZERO can be operated (1) in the extended inference mode when the environmental noise level is high or the device is far, (2) in the expedited mode when the user interacts with the device more frequently or when there are many users issues commands to the device, (3) in the ensemble mode when there are multiple microphones and each can be specialized on detecting different subset of voice commands, or (4) in the training mode when person or environment-specific parameter tuning is necessary to achieve more accurate classification results.

3 ACCELERATION MODES

In this section, we describe the four modes of zero-energy acceleration in Neuro.ZERO, of which, only one mode is active at a time, as configured by the application developer.

3.1 Extended Inference

A larger network having more neurons, in general, is a better classifier [67, 68, 113]. Although there are studies showing that the accuracy of a DNN drops when its size grows beyond a certain limit [32, 53], for resource-constrained embedded systems like Neuro.ZERO, we safely assume that more neurons and connections are likely to improve its inference accuracy. The memory of an MCU being small, a DNN residing in the main MCU of Neuro.ZERO is benefited by additional neurons in the accelerator since some DNNs cannot be stored in a single MCU even after compression. For example, SqueezeNet (470KB) [58] is a compressed version of AlexNet [64], but it is still too large to fit in the main MCU (256KB for MSP430). In such cases, an accelerator becomes necessary for the system to achieve desirable performance.

Based on this assumption, given the baseline DNN, Neuro.ZERO generates an extended version of it by adding additional neurons to each layer. The newly added neurons are identical in numbers and types for each layer. Figure 2b shows an example of an extended DNN that has three extended *convolutional* (Conv) and two extended *fully-connected* (FC) layers having the same dimensions as in the baseline DNN. To avoid creating a dependency between the two MCUs which requires extensive communication between them at run-time, we intentionally regularize (remove) the connections between convolutional filters running on the two MCUs and execute all fully connected layers on the main MCU. The benefit of this are two-fold: first, the main MCU independently makes inferences, and second, execution of half of the convolutional filters, which account for 45% of the total energy consumption, are offloaded to the accelerator. For example, scaled-down versions of popular DNNs like ResNet [49] can be divided into two networks and accelerated by using parallel algorithms for DNNs such as [40].

The accelerator executes the convolutional filters in an energy-aware manner by selecting a subset of them for execution based on the current level of harvested energy. We call this *step-up inference* since the accelerator's effort toward increasing the inference accuracy increases proportionally with the harvested energy. The details of the algorithm are described in Section 4.1.

3.2 Expedited Inference

Exploiting the inherent parallelism in DNN architecture is a common technique to increase the speedup of DNN execution. We leverage this parallelism in Neuro.ZERO by executing a subset of the convolutional filters of the baseline DNN on the accelerator. Like the extended inference mode, the fully connected layers run on the main MCU to ensure that the main MCU makes inferences without requiring frequent communication with the accelerator. Since executing convolutional layers take as much as 90% of the total execution time, the expedited mode cuts down the inference time approximately by 45%. Figure 2b shows an example of expedited DNN having three convolutional layers offloaded from the baseline DNN. Although this mode looks similar to the extended inference, the main difference between the two is that unlike the extended mode, the expedited mode trades off accuracy for speedup.

3.3 Ensembled Inference

Unlike the above two modes, the ensembled inference mode executes an independent DNN on the accelerator, which is given as an

additional input to Neuro.ZERO. This mode enables execution of a different DNN that performs the same inference task and provides a second opinion on the inference result. It also allows execution of a different inference task that may complement inference results on the main MCU. Furthermore, the accelerator may choose to use a different sensor than the main MCU to perform the same or a different inference task than the main MCU. Thus, this mode offers the most flexibility, but it does not necessarily improve the speedup. However, by carefully choosing a suitable combination of sensors and inference tasks, novel multi-modal, multi-objective sensing and inference systems can be developed with this mode—which may effectively increase the accuracy and speedup of inference. Figure 2b shows an example of an ensembled DNN. Unlike the baseline DNN having a convolutional architecture, the accelerator runs a fully-connected DNN that learns non-spatial features. When the accelerator is available, the output of the accelerator is combined with that of the main MCU to generate the final inference result.

Although Neuro.ZERO is a minimalistic system that has only one main MCU and one accelerator, the design can be extended to support many-MCU systems that run more complicated tasks and ensembles of many networks. Outputs of these networks can be combined using existing techniques such as concatenation and averaging [45, 109].

3.4 Latent Training

Real-time training of machine learning classifiers is a desirable feature for many mobile and embedded systems [31]. In recent years, we see a growing trend of online training of embedded classifiers in commercial products such as iPhone's face recognition [4], Google Clip's image capturing [39], and Android smartphone's key-press learning features [46]. To future-proof Neuro.ZERO, we introduce a fourth acceleration mode that enables retraining of the DNN on the accelerator. We call this *latent training* since the training process gradually progresses over time as the accelerator harvests energy. Training happens separately on the accelerator while the main MCU independently executes inference tasks. The two MCUs asynchronously communicate with each other only when the DNN model has been updated via training. Then, the main MCU fetches the newly updated model and uses it from then on.

Unlike the DNNs that have several millions of parameters, requiring thousands of training examples to train, and are meant to run on high-end processors, the DNNs in Neuro.ZERO are much smaller in size and training happens online, i.e., only one example at a time to perform a back-propagation algorithm. However, even a single round of back-propagation is difficult on a small system that is powered intermittently. To solve this challenge, we propose an energy-aware back-propagation algorithm that updates the weight parameters of a DNN in proportion to the amount of harvested energy. The details of the algorithm are in Section 4.2.

One caveat of on-device training is that the system requires labeled data. To handle this, we propose several solutions: (1) applying semi-supervised learning principles that do not require labeled data [71, 90, 124], (2) relying on an external, high-accuracy inference system to obtain the labels at run-time, and (3) in a distributed sensor network or ensemble scenario, aggregating (e.g., voting) neighboring nodes' inference results and treat it as the label. In our demonstration, we use (2) for the simplicity of implementation.

4 ENERGY-AWARE ACCELERATION

In this section, we introduce energy-aware acceleration algorithms called *step-up inference* and *skip-out training*, which enable intermittent inference and training of DNN based on the energy level.

4.1 Step-Up Inference

The *step-up inference* enables flexible inference acceleration of DNNs on the unpredictable harvested-energy. It dynamically adjusts acceleration in proportion to the run-time energy level by stepping up and down the size of DNN executed on the accelerator with multiple steps. Since the network size grows along with steps, e.g., step four has a larger DNN than step three, etc., a higher step is expected to achieve better performance acceleration (i.e., higher accuracy) than a lower one. Every execution of inference, the highest step that can be executed with the current energy level is selected among total n steps and executed on the accelerator.

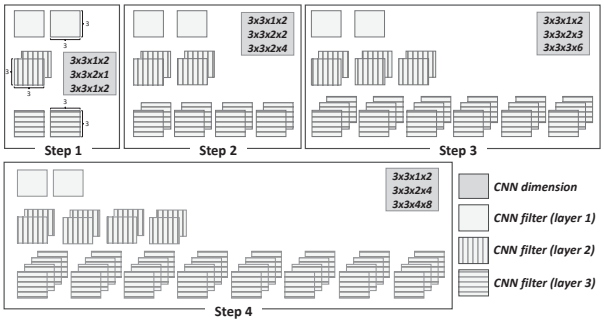


Figure 3: An example of step-up inference: The number of CNN filters running on the accelerator incrementally increase along with steps, and only one step is executed based on the energy level.

A set of n steps can be expressed as a set $S = \{S_1, S_2, ..., S_n\}$ and the amount of energy required to execute each step is given by another set $C = \{c_1, c_2, ..., c_n\}$, where S_i is the set of CNN filters of i-th step, and c_i is the energy consumption of the i-th step. Figure 3 depicts an example of four steps having different numbers of CNN filters that incrementally increase along with the steps. Starting from the baseline DNN, each step grows by adding a set of new filters to the previous step. Thus, $S_{i+1} = S_i \cup \{\text{new CNN filters}\}$ and $S_i \subset S_{i+1}$. The step S_{i+1} is obtained by training {new CNN filters} step-by-step until n while freezing S_i. In this way, the filters of the previous steps are reused without changes, and they are prevented from learning redundant features. While the total number of steps, n can be arbitrarily set at compile-time based on energy harvesting pattern, the total number of accelerating filters, $\sum_{i=1}^{n} |S_i|$ is limited to the same number of filters as the baseline DNN in the main MCU.

For every inference acceleration, Neuro.ZERO determines a step to be executed by the accelerator based on the currently-available energy at run-time. The step to be executed at the k-th inference execution, s_k is determined by $s_k = \text{argmax}_{i \le n}\, c_i$ subject to $c_i \le e_k$ where e_k is the current energy level at the k-th inference.

4.2 Skip-Out Training

The *skip-out training* enables intermittent training of DNN on the irregular energy harvesting pattern. It accelerates a train by ensuring the completion of one execution of back-propagation regardless of the amount of currently-available energy.

Skip-Out Back-Propagation. Unlike conventional training, the skip-out algorithm skips a back-propagation step for some of the weights with the skip-out rate, r_k at the k-th iteration of training:

$$r_k = 1 - \frac{1}{n} \min\left(\left\lfloor \frac{e_k}{e_f + e_b} \right\rfloor, n\right) \quad (1)$$

where n is a total number of weights in a DNN, e_k is the current energy level at the k-th iteration, e_f is the amount of energy needed for feed-forward of one weight, and e_b is the amount of energy needed for back-propagation of one weight. Given the skip-out rate, r_k and a total number of weights, n in a DNN, the number of weights to be trained at the k-th iteration, n_{r_k} is given by $n_{r_k} = \lfloor n(1 - r_k) \rfloor$.

For each k-th iteration, the skip-out rate, r_k is obtained from the current energy level, e_k; and only n_{r_k} weights are trained by back-propagation. Therefore, a different number of weights are trained every iteration, increasing the speed of training by guaranteeing the completion of one back-propagation regardless of the current energy level. Figure 4 shows back-propagation with skip-out.

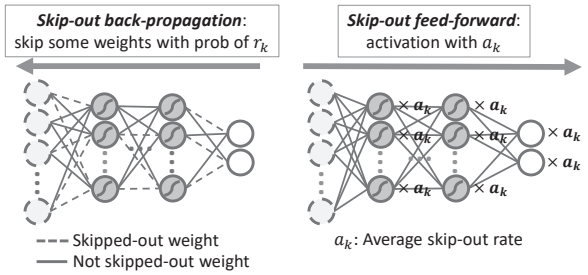


Figure 4: Skip-out back-propagation: Some weights are skipped with skip-out rate r_k for every k-th iteration. Skip-out feed-forward: All neurons are multiplied with the average skip-out rate a_k.

The weights to be skipped are selected using Bernoulli distribution [110] with probability, r_k with no other considerations such as their current values. This kind of skipping is effective and is known as *drop-out* [105] since it not only increases the training accuracy but also mitigates the overfitting problem [47]. Moreover, the Bernoulli distribution is one of the best choices for our system as any other selection algorithm, e.g., sorting or scoring, consumes more energy, which would not leave enough energy for back-propagation. The difference between skip-out and drop-out is that the skip-out rate changes at every iteration while the drop-out rate stays the same for the entire training (e.g., 0.5). Another difference is that the skip-out algorithm leaves the selected weights as they are without training, so they are used in back-propagation for the survived weights while drop-out completely removes them by setting their values to zero.

Skip-Out Feed-Forward. The skip-out-based feed-forward activation computation at the k-th iteration of training is given by:

$$o_j^{(l)} = \sum_i w_{i,j}^{(l)} \cdot \varphi(o_i^{(l-1)}) \cdot a_k \text{ and } a_k = \frac{1}{k} \sum_{i=1}^{k} r_i \quad (2)$$

where $o_j^{(l)}$ is the j-th neuron in the l-th layer, $w_{i,j}$ is the (i,j)-th weight in the l-th layer, $\varphi(\cdot)$ is an activation function, r_i is the skip-out rate at the i-th iteration, and a_k is the average skip-out rate until the k-th iteration. Unlike the skip-out back-propagation that skips some weights, the skip-out feed-forward does not skip any weights for activation computation. Instead, the average skip-out rate, a_k until the k-th iteration in Equation 2 is applied to the activation of all neurons. Figure 4 shows feed-forward with skip-out.

Skip-out trains a DNN with a different number and combination of weights for each iteration. Since weights are trained with the probability of $1 - r_k$ for k-th iteration in the back-propagation, it is averaged by a_k in the feed-forward. Hence, any weight trained for a specific DNN does not dominate feed-forward. In general, an averaged feed-forward of different DNNs results in a better performance than a feed-forward based on one particular DNN [105].

Convergence of a_k. The average skip-out rate a_k used for feed-forward converges after a number of training iterations. By using Equation 1, a_k in Equation 2 can be re-written as:

$$a_k \simeq \frac{1}{k} \sum_{i=1}^{k} \left(1 - \frac{e_i}{n(e_f + e_b)}\right) = \frac{1}{k} \sum_{i=1}^{k} 1 - \frac{\mu_{e_k}}{n(e_f + e_b)} \quad (3)$$

where $\mu_{e_k} = \frac{1}{k} \sum_{i=1}^{k} e_i$ is the mean of e_i. Since n, e_f, and e_b are constants and μ_{e_k} tends to converge to a constant as $k \to \infty$, a_k also converges. If we consider e_i as an independent random variable, its distribution tends toward a normal distribution as $e_i \sim \mathcal{N}(\mu_{e_k}, \sigma_{e_k})$ where μ_{e_k} is mean, and σ_{e_k} is variance regardless of its original distribution as $i \to \infty$ based on the *Central Limit theorem* [98].

5 NUMERICAL ACCELERATION

In this section, we present an underlying numerical acceleration of Neuro.ZERO called *Adaptive-Scale Fixed-Point (ASFP)* arithmetic that adjusts the scaling factor of fixed-point (FP) numbers during arithmetic operations. It produces more reliable numerical results than fixed-point while being faster than floating-point operations.

5.1 Fixed-Point Numbers

The standard 32-bit IEEE-754 floating-point numbers are either not supported or computationally very slow in embedded systems that do not have an on-board *Floating Point Unit* (FPU) [2]. For these reasons, most embedded systems that have no hardware support for floating-point operations, use *fixed-point (FP)* arithmetic [88] which is numerically less accurate than floating-point.

Given total n bit-width, a number x is represented with FP format using x_f number of fractional bits (Qx_f), i.e., $x = x_b 2^{-x_f}$ for $1 \leq x_f \leq n - 1$ where x_b is the integer base ranging from -2^{n-1} to $2^{n-1} - 1$ for a signed number, e.g., 1.625 is represented as $1664 \cdot 2^{-10}$ with $Q10$. Increasing the scaling factor increases the range and reduces precision. On the contrary, reducing it reduces the range and increases precision. Hence, FP is a number format having a unique shared fixed exponent with a trade-off between the range and precision. Since the scaling factor is fixed for every number, overflow and precision loss occurs in compute-intensive DNN tasks.

5.2 Adaptive-Scale Fixed-Point Arithmetic

To overcome the limitations of fixed-point (FP), we propose adaptive-scale fixed-point (ASFP) numbers that adjust the scaling factor when performing the four fundamental arithmetic operations. An ASFP-based DNN can be trained with significantly less error by mitigating the overflow and precision loss problem of FP. Here, we describe adaptive-scale Multiply-Accumulate (MAC) operation that

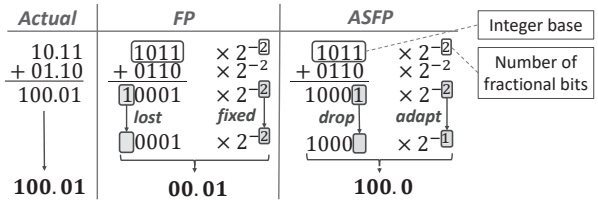


Figure 5: For addition of two binary number 10.11 ($Q2$) and 01.10 ($Q2$), FP produces 00.01 using the fixed scaling factor (-2). On the other hand, ASFP produces 100.0 by adapting the integer base (1000) and the scaling factor (-1) which is closer to the actual result 100.01.

is frequently performed in DNNs. To understand it, the two parts of MAC, i.e., addition and multiplication are first discussed.

ASFP Addition. Addition of two FP number $x = x_b 2^{-x_f}$ and $y = y_b 2^{-y_f}$ $(x_f \geq y_f)$ given total n bit-width is given by:

$$x + y = x_b 2^{-x_f} + y_b 2^{-y_f} = (x_b 2^{y_f - x_f} + y_b) 2^{-k} 2^{-(y_f - k)} \quad (4)$$

where $(x_b 2^{y_f - x_f} + y_b) 2^{-k}$ is new integer base and $(y_f - k)$ is new number of fractional bits for the addition result. If $(x_b 2^{y_f - x_f} + y_b) 2^{-k}$ does not fit into the maximum integer base range between -2^{n-1} to $2^{n-1} - 1$, the result will overflow and end up being inaccurate. On the other hand, if it is too small, it will not overflow but end up being too coarse with relatively small fractional bits, which scarifies its precision. Hence, to provide the most fine-grained precision without overflow, new integer base $|x_b 2^{y_f - x_f} + y_b| 2^{-k}$ needs to be maximized by finding minimum k such that:

$$k \geq \lceil \log_2 |x_b 2^{y_f - x_f} + y_b| \rceil - (n-1); \text{ and, } k \geq y_f - (n-1) \quad (5)$$

$\lceil \log_2(\cdot) \rceil$ is computed by finding the most significant bit (MSB). It is efficiently obtained either using bit-shifting operations, which is extremely fast or using a near constant-time algorithm such as De Bruijn sequence [25]. As an example, adaptive-scale addition of two binary FP number 10.11 ($Q2$) and 01.10 ($Q2$) with 4 bit-widths is given in Figure 5, which shows ASFP produces more accurate result than FP by preventing overflow.

ASFP Multiplication. Multiplication of two FP number $x = x_b 2^{-x_f}$ and $y = y_b 2^{-y_f}$ given total n bit-width is given by:

$$xy = x_b 2^{-x_f} \cdot y_b 2^{-y_f} = x_b y_b 2^{-k} 2^{-(x_f + y_f - k)} \quad (6)$$

where $x_b y_b 2^{-k}$ is new integer base and $(x_f + y_f - k)$ is new number of fractional bits for the multiplication result. The integer base and scaling factor of the multiplication result are adjusted by finding minimum k such that:

$$k \geq \lceil \log_2 |x_b| \rceil + \lceil \log_2 |y_b| \rceil - (n-1); \text{ and, } k \geq x_f + y_f - (n-1) \quad (7)$$

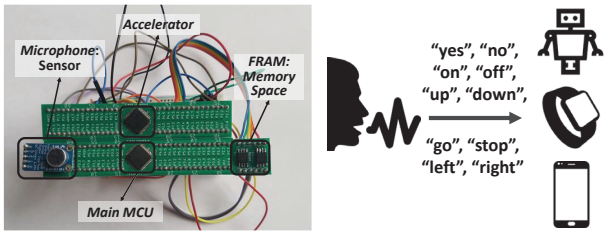
Same as the addition, the calculation of $\lceil \log_2(\cdot) \rceil$ and k can be efficiently performed. Once a multiplication is performed, the log value of the multiplication result does not need to be computed any more since it stays as $(n-1)$ from then on, which results in the even faster computation for future multiplications.

ASFP MAC. By combining the adaptive-scale addition and multiplication, the MAC computation is effectively performed with numerical accuracy similar to floating-point, which provides a more reliable result than FP. Also, it is computationally more efficient than floating-point since its basic format is based on FP.

Given two vectors, it first performs element-wise multiplication and updates the scaling factor and the integer base for each multiplication result using Equation 7. After all the results are added up, the final summation is obtained by finding the best integer base and the scaling factor based on Equation 5. By providing a unified MAC operation, its computational efficiency is further improved when compared to performing individual multiplication and addition.

6 IMPLEMENTATION

The Neuro.ZERO platform consists of two MCUs, memory space, sensors, and energy storage, which is shown in Figure 6a.



(a) Neuro.ZERO prototype platform **(b) Voice command recognition**

Figure 6: Neuro.ZERO hardware platform: a custom-built dual-MCU prototype for zero-energy acceleration.

The Main MCU and Accelerator. The Neuro.ZERO platform has two microcontrollers (MSP430FR5994 [107]), serving as the main MCU and the accelerator, which can operate with low power (118μA–1.8mA) supplied from an energy harvester. Two power connectors dedicated to each MCU allow separate power supplies, i.e., stable power (battery) and energy-harvesting power (batteryless).

Memory Space. A memory module is connected to both the main MCU and the accelerator. It works as a common data storage for the shared data, e.g., intermediary data or sensor readings. A FRAM [11] is chosen to be placed between two MCUs since it is a nonvolatile memory performing read/write operation in nanoseconds with high energy efficiency [24]. These attributes of FRAM minimize data sharing overhead between the main MCU and the accelerator.

Sensors. Sensors are connected to both the main MCU and the accelerator so that the data is accessible by both without any lag. They are powered from the battery for reliable and timely data collection. They are connected through the pin-headers on the below surface, e.g., we connect a camera and microphone for the traffic sign and voice command recognizer, respectively.

Energy Storage. A capacitor charged by an energy harvester works as the energy buffer for the accelerator. When the energy level of the capacitor exceeds the required energy level for acceleration, the system gets accelerated. The amount of energy needed for acceleration is statistically obtained from multiple energy measurements.

7 ALGORITHM EVALUATION

Prior to describing the performance of Neuro.ZERO in real-world scenarios (Section 8), we conduct dataset-driven experiments to evaluate the two core algorithms of Neuro.ZERO, i.e., energy-aware acceleration (step-up inference and skip-out training) and adaptive-scale fixed-point (ASFP). The evaluation of the step-up and skip-out algorithms is conducted on a GPU machine (GTX 1080 Ti) using four datasets, i.e., MNIST [70], CIFAR-10 [63], SVHN [86], and Fashion

MNIST [118]. We use a variation of LeNet architecture [70] as the baseline DNN, which is also used later in the traffic sign recognizer (Section 8.1). The performance of ASFP is evaluated on an MCU.

7.1 Energy-Aware Acceleration

Evaluation of Step-up Inference. To evaluate the effectiveness of the step-up inference, we measure the inference accuracy of the extended inference mode by varying the *step* from step 1 to step 5 for each of the four datasets. For training, we use a learning rate of 10^{-3} with Adam optimizer [62], L2 regularization parameter of 10^{-6}, and a mini-batch size of 96. A separate test dataset (different from the training data) is used to evaluate the accuracy of each network. The test accuracy, along with the size of the DNN and the number of CNN filters for each step are shown in Figure 7.

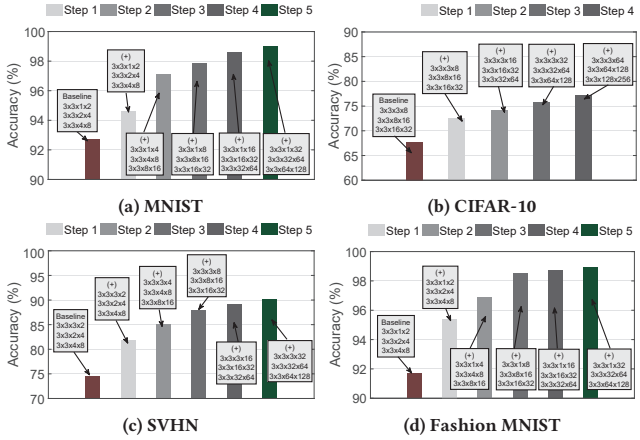


Figure 7: Performance of step-up inference for different *steps*.

For all datasets, the accuracy increases as steps are increased, i.e., from 92.7% to 99.0%, 67.7% to 77.1%, 74.5% to 90.2%, and 91.7% to 98.9% for MNIST, CIFAR-10, SVHN, and Fashion MNIST dataset, respectively. However, the increment in accuracy is relatively smaller as the network grows. For example, the delta in accuracy for SVHN dataset is initially 7% from step 1 to step 2, but later it drops to 1% from step 4 to step 5. Since the gain in accuracy tends to maximize during the first few steps of the step-up algorithm, executing extended inference for smaller *steps* is an effective strategy to improve the inference performance.

Evaluation of Skip-out Training. We compare the inference accuracy of the skip-out algorithm running at different skip-out rates against two baseline solutions: DNNs that do not implement skip-out (no skip-out) and thus it is expected to set the upper limit for skip-out; and DNNs that implement drop-out (sets 50% weights to zero). These DNNs are trained and tested on different, non-overlapping subsets of the dataset. Figure 8 shows how the accuracy (evaluated on the test dataset) varies as the number of training iteration (on the training dataset) is increased.

We observe that for every dataset, the accuracy of skip-out converges to no skip-out. For instance, a skip-out rate between 0.0–0.4 results in a similar accuracy to no skip-out with a negligible (0.4%-0.9%) loss in accuracy for any dataset. Furthermore, skip-out yields a similar or higher accuracy to drop-out given similar skip-out rates. In general, the performance of skip-out depends on its rate; as the rate gets closer to zero, its accuracy gets closer to no skip-out.

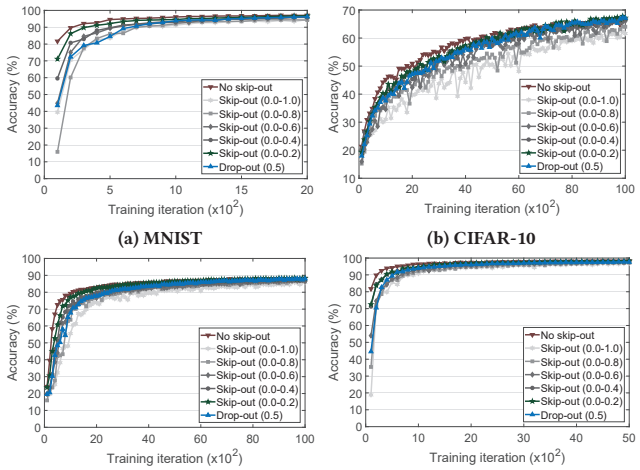


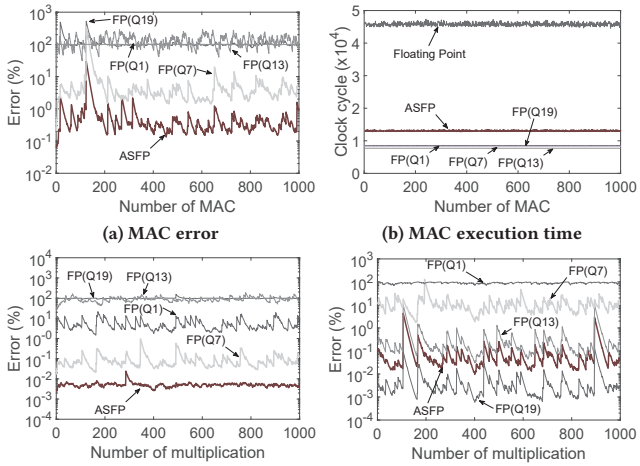
Figure 8: Performance of skip-out at different skip-out rates.

We also observe that skip-out requires a shorter training time to achieve comparable accuracy to no skip-out (not shown in the figure). For instance, skip-out (0.0-0.6) reaches 80% of accuracy about 100 iterations earlier than no skip-out. This is because the number of weights trained at each iteration in skip-out algorithm is flexibly changed based on the energy, which is usually much smaller than the total number of weights. Although the gain in accuracy after each training iteration in skip-out is generally smaller than no skip-out, larger training iterations, given the same time, compensates for the sluggish increase in accuracy, and sometimes it slightly increases the overall accuracy. Skip-out saves training time and energy consumption and guarantees the completion of an iteration regardless of the energy level. This incremental and quick pace of training is more suitable for intermittent online learning.

7.2 Numerical Acceleration

To evaluate the effectiveness of adaptive-scale fixed-point (ASFP), its MAC operation error and execution time are compared against fixed-point (FP). Figure 9a shows the MAC operation errors of ASFP, and four FP formats ($Q19$, $Q17$, $Q13$, $Q1$) for two randomly generated 64×1 vectors having 20 bit-widths. Here, Qx denotes that x number of bits are used to represent the fractional part. The error is calculated as $|(f - x)/f| \times 100$, where f is MAC result of floating-point (32bit) and x is the MAC result of ASFP or FP. As shown in the figure, ASFP provided average 0.71% error, which is ten times less than the best-performing FP ($Q7$, 7.32%). The errors of other FPs are numerically intolerable (more than 100%). Although the execution time (measured in clock cycles) of ASFP is 1.5 times slower than fixed-point, it is 3.4 times faster than floating-point with only 0.71% numerical difference, which is shown in Figure 9b.

To investigate the results further, we measure the overflow and precision errors separately. The overflow error is measured by multiplying two random numbers ranging from -128 to 128 and the precision error is measured by the multiplication of two random numbers between -2 and 2. Among FPs, Figure 9c and 9d show that $Q7$ yields the smallest error of 0.009% in the overflow test, while $Q19$ achieves the smallest error of 0.005% in the precision test. The overflow and precision errors of ASFP are 0.005% and 0.123%,



(a) MAC error (b) MAC execution time

(c) Overflow error (d) Precision error

Figure 9: The MAC error, MAC execution time, overflow error, and precision error of ASFP and four FPs (Q1, Q7, Q13, and Q19).

respectively. It demonstrates that unlike ASFP, a single fixed-point format can only provide either a small overflow error ($Q7$) or a small precision error ($Q19$), but not both. Hence, ASFP achieves better numerical correctness regarding both overflow and precision at the same time, which a fixed-point format cannot.

8 APPLICATION EVALUATION

8.1 Traffic Sign Recognizer

We implement a traffic sign recognizer, which uses a camera to capture and classifies 43 different types of traffic signs, as shown in Figure 10b. The system is powered by a 5V@40mA solar energy harvester. The camera first takes a 64×48 image with RGB565. The MCU converts the image into a grayscale image (32×32) and passes it to the DNN. The baseline DNN running on the main MCU consists of seven layers including the input and the output layers: 32×32×1 (input), 3×3×1×2 (Conv), 3×3×2×4 (Conv), 3×3×4×8 (Conv), 64 (FC), 128 (FC), 43 (output), which is a variant of the LeNet architecture [70] with an additional conv layer. Table 3 describes the network architecture during acceleration. We use ASFP with 16 bit-width for all numerical operations.

(a) Traffic sign photoshoot (b) Example images taken by the camera

Figure 10: Traffic sign recognizer: (a) Traffic signs are captured using a camera. (b) Examples of images taken.

Performance of the Accelerator. We evaluate the performance of four acceleration modes of Neuro.ZERO. We take photos of traffic signs from the GTSRB dataset [106] using the setup shown in Figure 10a. We capture 39,209 training images and 12,630 test images from 43 classes using a camera sensor connected to Neuro.ZERO as the images appear on the screen of a laptop. We also evaluate

	Main MCU	Accelerator
Extended	baseline[1]	step 1: 3×3×1×2, 3×3×2×1, 3×3×1×2 step 2: 3×3×1×2, 3×3×2×2, 3×3×2×4 step 3: 3×3×1×2, 3×3×2×3, 3×3×3×6 step 4: 3×3×1×2, 3×3×2×4, 3×3×4×8
Expedited	step1: baseline[2] step2: baseline[3]	step 1: 3×3×1×2, 3×3×2×1, 3×3×1×8 step 2: 3×3×1×2, 3×3×2×2, 3×3×2×8
Ensembled	baseline	32×32×1, 64, 128, 128, 64, 43 (FC DNN)
Latent Train	baseline	baseline with skip-out rate (0.0–0.4)

* Baseline[1]: 32×32×1, 3×3×1×2, 3×3×2×4, 3×3×4×8, 96, 192, 43
* Baseline[2]: 32×32×1, 3×3×1×2, 3×3×2×3, 3×3×3×8, 64, 128, 43
* Baseline[3]: 32×32×1, 3×3×1×2, 3×3×2×2, 3×3×2×8, 64, 128, 43

Table 3: The DNN architecture of the traffic sign recognizer

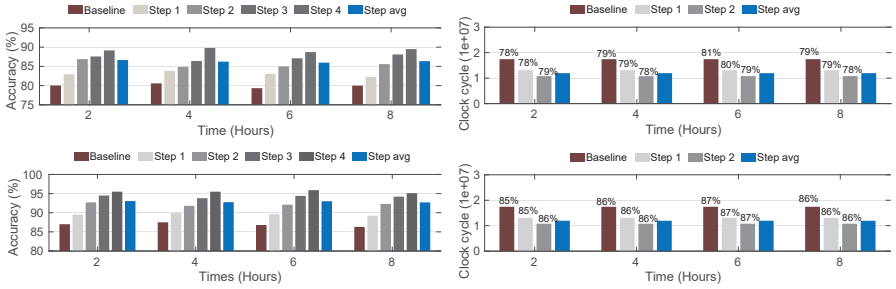
the performance of the traffic recognizer using the original traffic sign images from the GTSRB dataset as the input and compare it to the performance of the camera-taken images.

Figure 11a shows the recognition accuracy of the extended inference with four steps of incremental extension. Every two hours, a different set of 3,000 traffic signs (43 classes) is classified by the baseline DNN as well as the four steps to measure the accuracy. It shows that higher accuracy is achieved with further steps providing more extension of DNN. For instance, the baseline accuracy is improved from 80% to 83%, 86%, 87%, and 88% on average by each step with the camera-taken traffic sign images. Figure 11b shows the execution time of the expedited inference measured by clock cycle of the main MCU and the accuracy given two steps of incremental offloading. Compared to the baseline DNN, the execution time is decreased by 25% and 38%, accelerating the execution speed by 1.3× and 1.6× for step one and two, respectively. Both the camera-taken and original images experience only 1% of accuracy degradation by the maximum for the speed acceleration. Figure 11c shows the recognition accuracy of the ensembled inference with the second DNN consisting of six FC layers. The output of the two DNNs in the ensemble are combined using a fully-connected layer as done in [83]. By ensembling the FC DNN, the accuracy is improved from 80% to 85% and from 87% to 93% on average for the camera-taken and original GTSRB images, respectively.

Figure 11d shows the training accuracy over time for 20 classes of traffic signs performed by the latent training on the accelerator. Each single training example is trained online using SGD (Stochastic Gradient Descent [60, 97]) with the momentum algorithm [99]. As shown in the figure, the latent training keeps improving the accuracy over time up to 65% and 70% for the camera-taken and original GTSRB images, respectively. However, their accuracy is about 15% lower than the offline-trained DNNs (80% and 85%) on average since it uses SGD and ASFP instead of mini-batch and floating-point which usually provide better training performance.

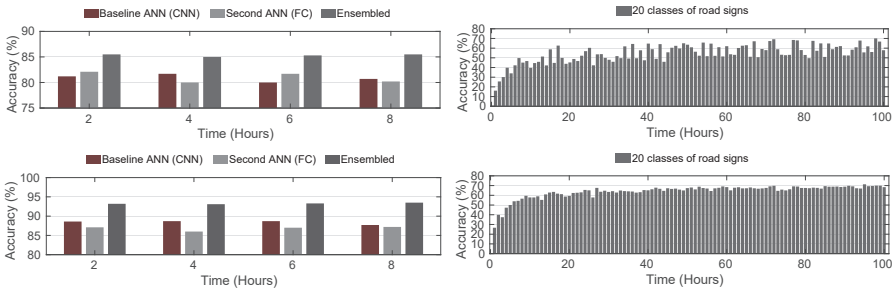
Execution Pattern of the Accelerator. We evaluate the execution pattern of four accelerations regarding the available energy of the accelerator. We measured the run-time energy level of the capacitor charged from a solar-harvesting panel (5V@40mA) for three hours (9 am - 12 pm) while executing each mode of acceleration every ten seconds which consume the energy in the capacitor.

Figure 12 shows the remaining energy level of the capacitor (harvesting minus consuming) over time and the amount of energy required by each step of four accelerations, i.e., the extended, expedited, ensembled inference, and the latent training. The horizontal lines on the figures indicate the minimum energy threshold required



(a) Extended inference with the camera-taken images (up) and the original GTSRB images (down)

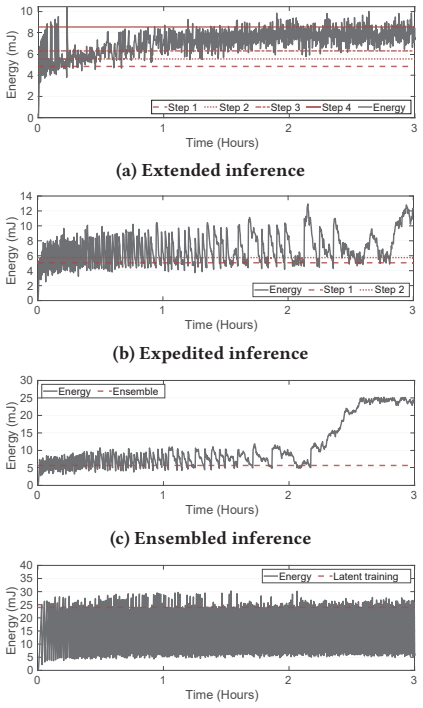
(b) Expedited inference with the camera-taken images (up) and the original GTSRB images (down)



(c) Ensembled inference with the camera-taken images (up) and the original GTSRB images (down)

(d) Latent training with the camera-taken images (up) and the original GTSRB images (down)

Figure 11: The inference accuracy of the traffic sign recognizer for all four modes of acceleration. Results are shown for both camera-taken images as well as the original GTSRB images.



(a) Extended inference

(b) Expedited inference

(c) Ensembled inference

(d) Latent training

Figure 12: Energy level of the capacitor during the execution of traffic sign recognizer.

for executing the acceleration with each step. When the current energy level is above one of the thresholds, the corresponding step is executed accordingly. For instance, as shown in Figure 12a, the extended inference with step 3 is executed at hour one since the energy level (7.35mJ) is larger than the step 3 threshold (6.29mJ) but smaller than the step 4 threshold (8.54mJ). Unlike the extended and expedited inference, the ensemble inference has only one energy threshold since the second DNN running on the accelerator is executed not in an energy-aware manner. Without the step-up inference algorithm, it is either executed or not. The latent train also has one energy threshold (train or not), but it runs with the skip-out that provides better utilization of energy. When the energy level is above the training threshold, it spends all the energy by training a portion of DNN proportionate to the current energy level.

The execution pattern of accelerations depends on the available energy (harvesting) and the energy required for acceleration (consuming). Table 4 shows the execution pattern of accelerations with each step out of total 1,080 executions. The baseline column indicates the standalone execution of main MCU (no acceleration) whereas the rests indicate the step-up accelerations on the accelerator. For the ensembled inference and latent training, we put their accelerations (ensembling/training) in the step 1 column. The execution patterns are different from each other since each acceleration with a different step consumes a different amount of energy.

Mode	Baseline	Step 1	Step 2	Step 3	Step 4
Extended	42 (3.8%)	56 (5.1%)	140 (12.9%)	729 (67.5%)	113 (10.4%)
Expedited	195 (18.0%)	169 (15.6%)	716 (66.2%)	-	-
Ensembled	172 (15.9%)	908 (84.0%)	-	-	-
Latent Training	862 (79.8%)	218 (20.1%)	-	-	-

Table 4: The execution pattern of accelerations (traffic sign)

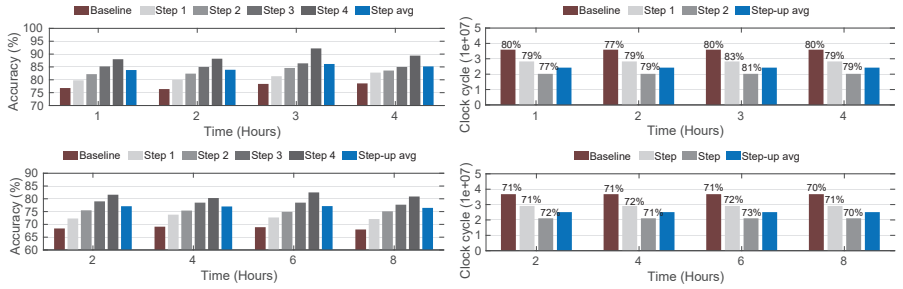
8.2 Voice Command Recognizer

We implement a limited-vocabulary speech recognition system that recognizes ten voice commands sensed through a microphone (Figure 6): {yes, no, on, off, up, down, go, stop, left, right} by using an RF energy harvester [91, 92]. To generate input for the DNN, the microphone first samples voice data at 8kHz. Then, it is divided into small frames consisting of 256 samples having an overlap of 128 samples between two frames. Frequency information is obtained for each frame using FFT with the help of the DSP module [107] in the MCU. Finally, MFCCs are generated as input data for the DNN by filling Mel-filter banks. The baseline DNN consists of total six layers including input and output: 61×13×1 (input), 12×6×1×4 (Conv), 6×3×4×8 (Conv), 64 (FC), 96 (FC), 10 (output), which is based on the small-footprint keyword spotting architecture [100]. Table 5 describes the detailed network architecture for acceleration. We apply the proposed adaptive-scale fixed point (ASFP) with 16 bit-width for all numerical operations.

Mode	Main MCU	Accelerator
Extended	baseline[1]	step 1: 12×6×1×1, 6×3×1×2 step 2: 12×6×1×2, 6×3×2×4 step 3: 12×6×1×3, 6×3×3×6 step 4: 12×6×1×4, 6×3×4×8
Expedited	step1: baseline[2] step2: baseline[3]	step 1: 12×6×1×1, 6×3×1×8 step 2: 12×6×1×2, 6×3×2×8
Ensembled	baseline	61×13×1, 64, 128, 128, 64, 10 (FC DNN)
Latent Training	baseline	baseline with skip-out rate (0.0–0.4)

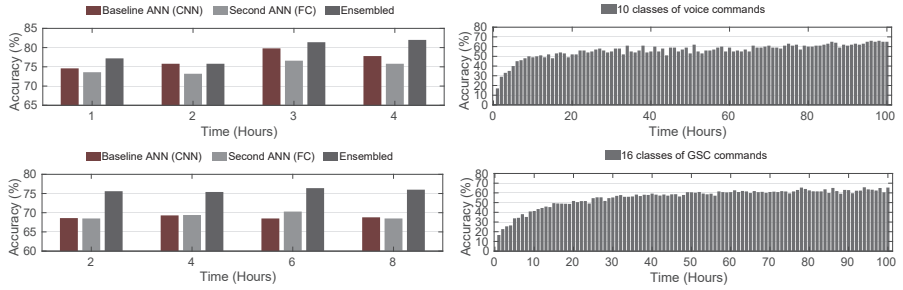
* Baseline[1]: 61×13×1, 12×6×1×4, 6×3×4×8, 96, 144, 10
* Baseline[2]: 61×13×1, 12×6×1×3, 6×3×3×8, 64, 96, 10
* Baseline[3]: 61×13×1, 12×6×1×2, 6×3×2×8, 64, 96, 10

Table 5: The DNN architecture of the voice command recognizer



(a) Extended inference with the microphone-captured words (up) and the GSC dataset (down)

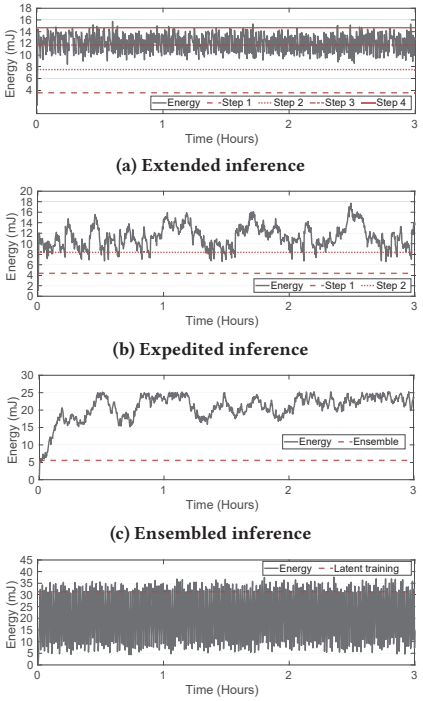
(b) Expedited inference with the microphone-captured words (up) and the GSC dataset (down)



(c) Ensembled inference with the microphone-captured words (up) and the GSC dataset (down)

(d) Latent training with the microphone-captured words (up) and the GSC dataset (down)

Figure 13: The inference accuracy of the voice command recognizer for all four modes of acceleration. Results are shown for both microphone-captured utterances as well as the GSC dataset.



(a) Extended inference

(b) Expedited inference

(c) Ensembled inference

(d) Latent training

Figure 14: Energy level of the capacitor during voice command recognition.

Performance of the Accelerator. We evaluate the performance of four accelerations by collecting voice commands from four people. In total, 10,000 commands (8,000 for train and 2,000 for test) from ten-word classes were captured through the microphone on our voice command recognizer. We also evaluate the performance with GSC dataset (Google Speech Command) [115] (84,843 training words in 35 classes and 11,005 test words) with the same experimental setup and compare it with the microphone-captured commands.

Figure 13a shows the recognition accuracy of the extended inference with four steps of incremental extension. For the microphone-captured commands, a different set of 500 commands (10 classes) is classified by the baseline DNN as well as the four steps to measure the accuracy every hour. For GSC dataset, a different set of 2,500 commands (35 classes) is classified every two hours. For both datasets, the accuracy is improved along with the steps from 76% to 92% (microphone) and from 68% to 77% (GSC) by the maximum. Figure 13b shows the execution time of the expedited inference measured by clock cycle of the main MCU and the accuracy given two steps of incremental offloading. Compared to the baseline DNN, the execution time is decreased by 21% and 43%, accelerating the execution speed by 1.2× and 1.7× for step one and two, respectively. Both the microphone-captured and GSC commands experience only 1.3% of accuracy degradation for the speed acceleration. Figure 13c shows the recognition accuracy of the ensembled inference with the second DNN consisting of six FC layers. By ensembling, the accuracy is improved from 77% to 80% and from 68% to 75% on average for the microphone-captured and GSC commands, respectively.

Figure 13d shows the training accuracy over time performed by the latent training. Based on SGD and momentum algorithm, 10 and 16 classes of voice command are trained for the microphone-captured and GSC commands system, respectively. The accuracy keeps improving over time and converges to 65% and 60% for the microphone-captured and GSC commands, which are about 11% and 8% lower than the offline-trained DNNs (76% and 68%).

Execution Pattern of the Accelerator. We evaluate the execution pattern of four accelerations regarding the available energy of the accelerator. We measured the run-time energy level of the capacitor charged from an RF energy harvester [91, 92] for three hours while executing each mode of acceleration every ten seconds which consume the energy in the capacitor. Figure 14 show the remaining energy level of the capacitor (harvesting minus consuming) over time and the amount of energy required by each step of four accelerations. For instance, as shown in Figure 14b, the expedited inference with step 2 is executed at hour two since the energy level (10.27mJ) is larger than the step 2 threshold (8.34mJ). Table 6 shows the execution pattern out of total 1,080 executions.

Mode	Baseline	Step 1	Step 2	Step 3	Step 4
Extended	0 (0%)	0 (0%)	448 (41.4%)	615 (56.9%)	17 (1.5%)
Expedited	0 (0%)	81 (7.5%)	999 (92.5%)	-	-
Ensembled	5 (0.4%)	1075 (99.5%)	-	-	-
Latent Training	902 (83.5%)	178 (16.4%)	-	-	-

Table 6: The execution pattern of accelerations (voice command)

8.3 Overhead of Acceleration

Although the accelerator runs only on harvested energy, the main MCU needs to process the data from the accelerator, which causes an overhead on the main MCU. We evaluate this overhead by measuring the additional power consumption and clock cycles required

for acceleration, compared to standalone execution of the main MCU. Table 7 shows the overhead of the main MCU for four modes of acceleration for the traffic sign recognizer. The percentages indicate their relative amount compared to standalone execution of the main MCU without the accelerator. For latent training, the overhead refers to the cost of fetching a trained model from the accelerator to the main MCU. We observe that the three inferences require less than 1% extra energy and clock cycles for acceleration, while the overhead of latent training is relatively higher. This is because the amount of data moved between two MCUs are different for inference and latent training. During inference, the two MCUs exchange relatively small chunks of data for input/output and intermediate results, whereas latent training requires movement of relatively larger sized classifier models. However, the frequency of fetching models is much lower than the frequency of interaction between the two MCUs in other three modes since a model is fetched occasionally, only when it has been improved by completing a predefined number of training iterations on the accelerator.

Mode	Energy Overhead	Clock Cycle Overhead
Extended Inference	0.065 mJ (0.7%)	256×10^3 (0.9%)
Expedited Inference	0.058 mJ (0.9%)	240×10^3 (1.4%)
Ensembled Inference	0.055 mJ (0.6%)	240×10^3 (0.9%)
Latent Train (fetching)	3.4 mJ (-%)	$15,344 \times 10^3$ (-%)

Table 7: The overhead of the main MCU due to acceleration.

9 RELATED WORK

Embedded DNN Accelerator. DNN inference accelerators using FPGAs such as [30, 93, 112, 122] have been widely studied due to their high performance and reconfigurability. Also, there are some accelerators based on different platforms, e.g., embedded GPU [14, 42] or ARM microprocessors [36]. However, they only focus on speeding up DNN inference given a pre-trained model, unlike the proposed accelerator that enables accuracy improvement as well as training. Although some work introduced trainable accelerators, they depend on a specific platform such as NVIDIA GPU [29] or specific hardware [17, 61, 82]. None of these works utilize energy harvesters as their power source in an energy-aware manner.

DNN with Fixed-Point. Fixed-point arithmetic for DNNs has been explored in earlier works ranging from the theoretical analysis [28, 54] to implementations [21, 41]. Recently, [73] showed that DNNs could be effectively trained using only fixed-point arithmetic and many approaches have been proposed to increase accuracy and efficiency. Most popular approaches are based on compression of a pre-trained model, including quantization of weights [3, 37, 56, 111] and extremely low-precision (1-3 bits) [22, 57]. However, the proposed adaptive-scale fixed-point is applicable to general DNNs without requiring any compression. Although a pre-trained DNN model can be effectively reduced for fixed-point by compressing, [73, 96] showed that training DNN models with fixed-point results in better accuracy. In accordance with these results, we train DNNs from scratch using adaptive-scale arithmetic, i.e., MAC, multiplication, and addition. Similar to our work, [41] trained DNNs by taking a maximum integer base using stochastic and near-rounding. However, unlike ours, their scaling factor is fixed for the entire training.

Intermittent Computing. Existing work on intermittent computing address some important system-level problems, such as atomicity [19, 79], consistency [19, 77, 79], programmability [51], timeliness [51], and energy efficiency [12, 20, 50]. Although they enable efficient code execution of general-purpose tasks on batteryless systems, none of them considers the learning aspects of a DNN such as their accuracy or training. Recently, [33] implemented intermittent inference on harvested energy using a microcontroller. However, its execution of inference is not guaranteed unlike the proposed system since it entirely depends on harvested energy. For DNN training, [87] proposed layer-by-layer training approach with the concept of lifelong learning, which repeatedly trains a fixed number of weights without skipping out.

10 DISCUSSION

Unpredictable Harvested Energy. Due to the unpredictable nature of harvested energy, the accelerator may not be available at desired instants. To enable timely wake-up, specially designed energy management unit, along with scheduling algorithms for energy harvesting systems [7, 8, 18, 78, 84] should be implemented alongside Neuro.ZERO. To meet the varying energy demands of the running application, reconfiguration energy storage [20] and/or multi-capacitor systems [50] should be implemented to scale and/or partition harvested energy for efficient and timely use.

Caveats to On-Device Training. There are certain caveats to on-device online training on Neuro.ZERO. First, a mini-batch size of one is used in our implementation of the latent training mode of Neuro.ZERO, which might increase noise and cause abrupt changes in the training process [9, 72, 81]. To mitigate this, multiple examples should be stored and trained together as a batch instead of training only one. Second, a large learning rate may never converge to an optimal solution but to a sub-optimal one [6, 38]. To handle this, the learning rate should be decayed after a number of training iterations. Alternatively, transfer learning techniques [89, 108] such as retraining only the last few layers as opposed to training the whole network could be employed.

Using Battery as a Backup. Besides the energy harvester, the accelerator could use a battery of its own as a backup source. Although such a design allows waking-up the accelerator in times of need, eventually, the battery will die, and the design will fall back to our current implementation of Neuro.ZERO. Another alternative design is to design a single-MCU system that has both a battery and a harvester. Although such a design increases the battery-life of the MCU, the performance-gain in DNN acceleration on a single-MCU system is never going to be as high as a multi-MCU system like Neuro.ZERO, which has more computational capacity.

11 CONCLUSION

We introduce Neuro.ZERO, an intermittently-powered accelerator that draws no system energy and opportunistically accelerates the performance of a DNN based on the four modes of acceleration. To enable zero-energy acceleration, energy-aware acceleration algorithms, and adaptive-scale fixed-point are proposed. A traffic sign and a voice command recognizer are implemented, and they have been demonstrated that the inference accuracy and speed increase.

ACKNOWLEDGMENT

This paper was supported, in part, by NSF grants CNS-1816213 and CNS-1704469.

REFERENCES

[1] National Highway Traffic Safety Administration. 2013. Traffic Safety Facts. https://crashstats.nhtsa.dot.gov/Api/Public/ViewPublication/812124. (2013).

[2] SF Anderson, JG Earle, R Et Goldschmidt, and DM Powers. 1967. The IBM system/360 model 91: floating-point execution unit. *IBM Journal of research and development* 11, 1 (1967), 34–53.

[3] Sajid Anwar, Kyuyeon Hwang, and Wonyong Sung. 2015. Fixed point optimization of deep convolutional neural networks for object recognition. In *Acoustics, Speech and Signal Processing (ICASSP), 2015 IEEE International Conference on*. IEEE, 1131–1135.

[4] Apple. 2017. Neural Engine. https://www.apple.com/iphone-xs/a12-bionic/. (2017).

[5] Arm. 2013. big.LITTLE technology. https://www.arm.com/files/pdf/bigLITTLETechnologyTheFutureofMobile.pdf. (2013).

[6] Nii O Attoh-Okine. 1999. Analysis of learning rate and momentum term in backpropagation neural network algorithm trained to predict pavement performance. *Advances in Engineering Software* 30, 4 (1999), 291–302.

[7] David Audet, Leandro Collares De Oliveira, Neil MacMillan, Dimitri Marinakis, and Kui Wu. 2011. Scheduling recurring tasks in energy harvesting sensors. In *2011 IEEE Conference on Computer Communications Workshops (INFOCOM WKSHPS)*. IEEE, 277–282.

[8] Abdulrahman Baknina and Sennur Ulukus. 2017. Online scheduling for energy harvesting channels with processing costs. *IEEE Transactions on Green Communications and Networking* 1, 3 (2017), 281–293.

[9] Yoshua Bengio. 2012. Practical recommendations for gradient-based training of deep architectures. In *Neural networks: Tricks of the trade*. Springer, 437–478.

[10] Sourav Bhattacharya and Nicholas D Lane. 2016. Sparsification and separation of deep learning layers for constrained resource inference on wearables. In *Proceedings of the 14th ACM Conference on Embedded Network Sensor Systems CD-ROM*. ACM, 176–189.

[11] Dudley Allen Buck. 1952. *Ferroelectrics for Digital Information Storage and Switching*. Technical Report. MASSACHUSETTS INST OF TECH CAMBRIDGE DIGITAL COMPUTER LAB.

[12] Michael Buettner, Ben Greenstein, and David Wetherall. 2011. Dewdrop: an energy-aware runtime for computational RFID. In *Proc. USENIX NSDI*. 197–210.

[13] Erik Cambria and Bebo White. 2014. Jumping NLP curves: A review of natural language processing research. *IEEE Computational intelligence magazine* 9, 2 (2014), 48–57.

[14] Lukas Cavigelli, Michele Magno, and Luca Benini. 2015. Accelerating real-time embedded scene labeling with convolutional networks. In *Proceedings of the 52nd Annual Design Automation Conference*. ACM, 108.

[15] Jagmohan Chauhan, Suranga Seneviratne, Yining Hu, Archan Misra, Aruna Seneviratne, and Youngki Lee. 2018. Breathing-Based Authentication on Resource-Constrained IoT Devices using Recurrent Neural Networks. *Computer* 51, 5 (2018), 60–67.

[16] Liang-Bi Chen, Wan-Jung Chang, Jian-Ping Su, Ji-Yi Ciou, Yi-Jhan Ciou, Cheng-Chin Kuo, and Katherine Shu-Min Li. 2016. A wearable-glasses-based drowsiness-fatigue-detection system for improving road safety. In *2016 IEEE 5th Global Conference on Consumer Electronics*. IEEE, 1–2.

[17] Tianshi Chen, Zidong Du, Ninghui Sun, Jia Wang, Chengyong Wu, Yunji Chen, and Olivier Temam. 2014. Diannao: A small-footprint high-throughput accelerator for ubiquitous machine-learning. *ACM Sigplan Notices* 49, 4 (2014), 269–284.

[18] Maryline Chetto and Hussein El Ghor. 2019. Scheduling and power management in energy harvesting computing systems with real-time constraints. *Journal of Systems Architecture* (2019).

[19] Alexei Colin and Brandon Lucia. 2016. Chain: tasks and channels for reliable intermittent programs. *ACM SIGPLAN Notices* 51, 10 (2016), 514–530.

[20] Alexei Colin, Emily Ruppel, and Brandon Lucia. 2018. A Reconfigurable Energy Storage Architecture for Energy-harvesting Devices. In *Proceedings of the Twenty-Third International Conference on Architectural Support for Programming Languages and Operating Systems*. ACM, 767–781.

[21] Matthieu Courbariaux, Yoshua Bengio, and Jean-Pierre David. 2014. Training deep neural networks with low precision multiplications. *arXiv preprint arXiv:1412.7024* (2014).

[22] Matthieu Courbariaux, Yoshua Bengio, and Jean-Pierre David. 2015. Binaryconnect: Training deep neural networks with binary weights during propagations. In *Advances in neural information processing systems*. 3123–3131.

[23] George Cybenko. 1989. Approximation by superpositions of a sigmoidal function. *Mathematics of control, signals and systems* 2, 4 (1989), 303–314.

[24] Cypress. 2017. CY15B104Q. http://www.cypress.com/file/209146/download. (2017).

[25] Nicolaas Govert de Bruijn. 1975. *Acknowledgement of priority to C. Flye Sainte-Marie on the counting of circular arrangements of 2n zeros and ones that show each n-letter word exactly once*. Department of Mathematics, Technological University.

[26] Li Deng. 2014. A tutorial survey of architectures, algorithms, and applications for deep learning. *APSIPA Transactions on Signal and Information Processing* 3 (2014).

[27] Li Deng, Geoffrey Hinton, and Brian Kingsbury. 2013. New types of deep neural network learning for speech recognition and related applications: An overview. In *Acoustics, Speech and Signal Processing (ICASSP), 2013 IEEE International Conference on*. IEEE, 8599–8603.

[28] Sorin Draghici. 2002. On the capabilities of neural networks using limited precision weights. *Neural networks* 15, 3 (2002), 395–414.

[29] Aysegul Dundar, Jonghoon Jin, Berin Martini, and Eugenio Culurciello. 2017. Embedded streaming deep neural networks accelerator with applications. *IEEE transactions on neural networks and learning systems* 28, 7 (2017), 1572–1583.

[30] Clément Farabet, Berin Martini, Benoit Corda, Polina Akselrod, Eugenio Culurciello, and Yann LeCun. 2011. Neuflow: A runtime reconfigurable dataflow processor for vision. In *Computer Vision and Pattern Recognition Workshops (CVPRW), 2011 IEEE Computer Society Conference on*. IEEE, 109–116.

[31] National Science Foundation. 2019. Real-Time Machine Learning (RTML). https://www.nsf.gov/pubs/2019/nsf19566/nsf19566.htm?WT.mc_id=USNSF_25&WT.mc_ev=click. (2019).

[32] Xavier Glorot and Yoshua Bengio. 2010. Understanding the difficulty of training deep feedforward neural networks. In *Proceedings of the thirteenth international conference on artificial intelligence and statistics*. 249–256.

[33] Graham Gobieski, Nathan Beckmann, and Brandon Lucia. 2018. Intelligence Beyond the Edge: Inference on Intermittent Embedded Systems. *arXiv preprint arXiv:1810.07751* (2018).

[34] Graham Gobieski, Nathan Beckmann, and Brandon Lucia. 2018. Intermittent Deep Neural Network Inference. *SysML* (2018).

[35] Graham Gobieski, Nathan Beckmann, and Brandon Lucia. 2019. Intelligence Beyond the Edge: Inference on Intermittent Embedded Systems. In *ACM International Conference on Architectural Support for Programming Languages and Operating Systems (ASPLOS)*.

[36] Vinayak Gokhale, Jonghoon Jin, Aysegul Dundar, Berin Martini, and Eugenio Culurciello. 2014. A 240 g-ops/s mobile coprocessor for deep neural networks. In *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition Workshops*. 682–687.

[37] Yunchao Gong, Liu Liu, Ming Yang, and Lubomir Bourdev. 2014. Compressing deep convolutional networks using vector quantization. *arXiv preprint arXiv:1412.6115* (2014).

[38] Ian Goodfellow, Yoshua Bengio, Aaron Courville, and Yoshua Bengio. 2016. *Deep learning*. Vol. 1. MIT press Cambridge.

[39] Google. 2018. Google Clips. https://store.google.com/us/product/google_clips?hl=en-US. (2018).

[40] Stefanie Günther, Lars Ruthotto, Jacob B Schroder, EC Cyr, and Nicolas R Gauger. 2018. Layer-parallel training of deep residual neural networks. *arXiv preprint arXiv:1812.04352* (2018).

[41] Suyog Gupta, Ankur Agrawal, Kailash Gopalakrishnan, and Pritish Narayanan. 2015. Deep learning with limited numerical precision. In *International Conference on Machine Learning*. 1737–1746.

[42] Song Han, Xingyu Liu, Huizi Mao, Jing Pu, Ardavan Pedram, Mark A Horowitz, and William J Dally. 2016. EIE: efficient inference engine on compressed deep neural network. In *Computer Architecture (ISCA), 2016 ACM/IEEE 43rd Annual International Symposium on*. IEEE, 243–254.

[43] Song Han, Huizi Mao, and William J Dally. 2015. Deep compression: Compressing deep neural networks with pruning, trained quantization and huffman coding. *arXiv preprint arXiv:1510.00149* (2015).

[44] Boris Hanin. 2017. Universal function approximation by deep neural nets with bounded width and relu activations. *arXiv preprint arXiv:1708.02691* (2017).

[45] Lars Kai Hansen and Peter Salamon. 1990. Neural network ensembles. *IEEE Transactions on Pattern Analysis & Machine Intelligence* 10 (1990), 993–1001.

[46] Andrew Hard, Kanishka Rao, Rajiv Mathews, Françoise Beaufays, Sean Augenstein, Hubert Eichner, Chloé Kiddon, and Daniel Ramage. 2018. Federated learning for mobile keyboard prediction. *arXiv preprint arXiv:1811.03604* (2018).

[47] Douglas M Hawkins. 2004. The problem of overfitting. *Journal of chemical information and computer sciences* 44, 1 (2004), 1–12.

[48] Jibo He, William Choi, Yan Yang, Junshi Lu, Xiaohui Wu, and Kaiping Peng. 2017. Detection of driver drowsiness using wearable devices: A feasibility study of the proximity sensor. *Applied ergonomics* 65 (2017), 473–480.

[49] Kaiming He, Xiangyu Zhang, Shaoqing Ren, and Jian Sun. 2016. Deep residual learning for image recognition. In *Proceedings of the IEEE conference on computer vision and pattern recognition*. 770–778.

[50] Josiah Hester, Lanny Sitanayah, and Jacob Sorber. 2015. Tragedy of the coulombs: Federating energy storage for tiny, intermittently-powered sensors. In *Proceedings of the 13th ACM Conference on Embedded Networked Sensor Systems*. ACM, 5–16.

[51] Josiah Hester, Kevin Storer, and Jacob Sorber. 2017. Timely Execution on Intermi ently Powered Ba eryless Sensors. (2017).

[52] Embedded Intelligence Lab (UNC Chapel Hill). 2019. Neuro.ZERO open source project. https://github.com/learning1234embed/Neuro.ZERO. (2019).

[53] Sepp Hochreiter, Yoshua Bengio, Paolo Frasconi, and Jürgen Schmidhuber. 2001. Gradient flow in recurrent nets: the difficulty of learning long-term dependencies. (2001).

[54] Jordan L Holi and J-N Hwang. 1993. Finite precision error analysis of neural network hardware implementations. *IEEE Trans. Comput.* 3 (1993), 281–290.

[55] Kurt Hornik. 1991. Approximation capabilities of multilayer feedforward networks. *Neural networks* 4, 2 (1991), 251–257.

[56] Itay Hubara, Matthieu Courbariaux, Daniel Soudry, Ran El-Yaniv, and Yoshua Bengio. 2017. Quantized neural networks: Training neural networks with low precision weights and activations. *The Journal of Machine Learning Research* 18, 1 (2017), 6869–6898.

[57] Kyuyeon Hwang and Wonyong Sung. 2014. Fixed-point feedforward deep neural network design using weights+ 1, 0, and- 1. In *Signal Processing Systems (SiPS), 2014 IEEE Workshop on*. IEEE, 1–6.

[58] Forrest N Iandola, Song Han, Matthew W Moskewicz, Khalid Ashraf, William J Dally, and Kurt Keutzer. 2016. SqueezeNet: AlexNet-level accuracy with 50x fewer parameters and< 0.5 MB model size. *arXiv preprint arXiv:1602.07360* (2016).

[59] Shubham Kamdar and Neha Kamdar. 2015. big. LITTLE architecture: Heterogeneous multicore processing. *International Journal of Computer Applications* 119, 1 (2015).

[60] Jack Kiefer and Jacob Wolfowitz. 1952. Stochastic estimation of the maximum of a regression function. *The Annals of Mathematical Statistics* 23, 3 (1952), 462–466.

[61] Jonghong Kim, Kyuyeon Hwang, and Wonyong Sung. 2014. X1000 real-time phoneme recognition VLSI using feed-forward deep neural networks. In *Acoustics, Speech and Signal Processing (ICASSP), 2014 IEEE International Conference on*. IEEE, 7510–7514.

[62] Diederik P Kingma and Jimmy Ba. 2014. Adam: A method for stochastic optimization. *arXiv preprint arXiv:1412.6980* (2014).

[63] Alex Krizhevsky and Geoffrey Hinton. 2009. *Learning multiple layers of features from tiny images*. Technical Report. Citeseer.

[64] Alex Krizhevsky, Ilya Sutskever, and Geoffrey E Hinton. 2012. Imagenet classification with deep convolutional neural networks. In *Advances in neural information processing systems*. 1097–1105.

[65] Anders Krogh and Jesper Vedelsby. 1995. Neural network ensembles, cross validation, and active learning. In *Advances in neural information processing systems*. 231–238.

[66] Nicholas D Lane, Sourav Bhattacharya, Akhil Mathur, Petko Georgiev, Claudio Forlivesi, and Fahim Kawsar. 2017. Squeezing deep learning into mobile and embedded devices. *IEEE Pervasive Computing* 3 (2017), 82–88.

[67] Steve Lawrence, C Lee Giles, and Ah Chung Tsoi. 1997. Lessons in neural network training: Overfitting may be harder than expected. In *AAAI/IAAI*. Citeseer, 540–545.

[68] Steve Lawrence, C Lee Giles, and Ah Chung Tsoi. 1998. *What size neural network gives optimal generalization? Convergence properties of backpropagation*. Technical Report.

[69] Yann LeCun, Yoshua Bengio, and Geoffrey Hinton. 2015. Deep learning. *nature* 521, 7553 (2015), 436.

[70] Yann LeCun, Léon Bottou, Yoshua Bengio, and Patrick Haffner. 1998. Gradient-based learning applied to document recognition. *Proc. IEEE* 86, 11 (1998), 2278–2324.

[71] Dong-Hyun Lee. 2013. Pseudo-label: The simple and efficient semi-supervised learning method for deep neural networks. In *Workshop on Challenges in Representation Learning, ICML*, Vol. 3. 2.

[72] Mu Li, Tong Zhang, Yuqiang Chen, and Alexander J Smola. 2014. Efficient mini-batch training for stochastic optimization. In *Proceedings of the 20th ACM SIGKDD international conference on Knowledge discovery and data mining*. ACM, 661–670.

[73] Zhouhan Lin, Matthieu Courbariaux, Roland Memisevic, and Yoshua Bengio. 2015. Neural networks with few multiplications. *arXiv preprint arXiv:1510.03009* (2015).

[74] Hong Lu, AJ Bernheim Brush, Bodhi Priyantha, Amy K Karlson, and Jie Liu. 2011. Speakersense: Energy efficient unobtrusive speaker identification on mobile phones. In *International conference on pervasive computing*. Springer, 188–205.

[75] Zhou Lu, Hongming Pu, Feicheng Wang, Zhiqiang Hu, and Liwei Wang. 2017. The expressive power of neural networks: A view from the width. In *Advances in Neural Information Processing Systems*. 6231–6239.

[76] Brandon Lucia, Vignesh Balaji, Alexei Colin, Kiwan Maeng, and Emily Ruppel. 2017. Intermittent Computing: Challenges and Opportunities. In *LIPIcs-Leibniz International Proceedings in Informatics*, Vol. 71. Schloss Dagstuhl-Leibniz-Zentrum fuer Informatik.

[77] Brandon Lucia and Benjamin Ransford. 2015. A simpler, safer programming and execution model for intermittent systems. *ACM SIGPLAN Notices* 50, 6 (2015), 575–585.

[78] Yubo Luo and Shahriar Nirjon. 2019. SpotON: Just-in-Time Active Event Detection on Energy Autonomous Sensing Systems. In *Proceedings of the 25th IEEE Real-Time and Embedded Technology and Applications Symposium (RTAS WIP Session)*. IEEE, Montreal, Canada.

[79] Kiwan Maeng, Alexei Colin, and Brandon Lucia. 2017. Alpaca: intermittent execution without checkpoints. *Proceedings of the ACM on Programming Languages* 1, OOPSLA (2017), 96.

[80] Franco Manessi, Alessandro Rozza, Simone Bianco, Paolo Napoletano, and Raimondo Schettini. 2018. Automated pruning for deep neural network compression. In *2018 24th International Conference on Pattern Recognition (ICPR)*. IEEE, 657–664.

[81] Dominic Masters and Carlo Luschi. 2018. Revisiting small batch training for deep neural networks. *arXiv preprint arXiv:1804.07612* (2018).

[82] Paul A Merolla, John V Arthur, Rodrigo Alvarez-Icaza, Andrew S Cassidy, Jun Sawada, Filipp Akopyan, Bryan L Jackson, Nabil Imam, Chen Guo, and Yutaka Nakamura. 2014. A million spiking-neuron integrated circuit with a scalable communication network and interface. *Science* 345, 6197 (2014), 668–673.

[83] Milad Mohammadi and Subhasis Das. 2016. SNN: stacked neural networks. *arXiv preprint arXiv:1605.08512* (2016).

[84] Clemens Moser, Davide Brunelli, Lothar Thiele, and Luca Benini. 2007. Real-time scheduling for energy harvesting sensor nodes. *Real-Time Systems* 37, 3 (2007), 233–260.

[85] Saman Naderiparizi, Pengyu Zhang, Matthai Philipose, Bodhi Priyantha, Jie Liu, and Deepak Ganesan. 2017. Glimpse: A programmable early-discard camera architecture for continuous mobile vision. In *Proceedings of the 15th Annual International Conference on Mobile Systems, Applications, and Services*. ACM, 292–305.

[86] Yuval Netzer, Tao Wang, Adam Coates, Alessandro Bissacco, Bo Wu, and Andrew Y Ng. 2011. Reading digits in natural images with unsupervised feature learning. (2011).

[87] Shahriar Nirjon. 2018. Lifelong Learning on Harvested Energy. In *Proceedings of the 16th Annual International Conference on Mobile Systems, Applications, and Services*. ACM, 500–501.

[88] Erick L Oberstar. 2007. Fixed-point representation & fractional math. *Oberstar Consulting* (2007), 9.

[89] Sinno Jialin Pan and Qiang Yang. 2009. A survey on transfer learning. *IEEE Transactions on knowledge and data engineering* 22, 10 (2009), 1345–1359.

[90] Mohammad Peikari, Sherine Salama, Sharon Nofech-Mozes, and Anne L Martel. 2018. A cluster-then-label semi-supervised learning approach for pathology image classification. *Scientific reports* 8, 1 (2018), 7193.

[91] Powercast. 2016. Powercast p2110b. http://www.powercastco.com/wp-content/uploads/2016/12/P2110B-Datasheet-Rev-3.pdf. (2016).

[92] Powercast. 2016. Powercaster transmitter. http://www.powercastco.com/wp-content/uploads/2016/11/User-Manual-TX-915-01-Rev-A-4.pdf. (2016).

[93] Jiantao Qiu, Jie Wang, Song Yao, Kaiyuan Guo, Boxun Li, Erjin Zhou, Jincheng Yu, Tianqi Tang, Ningyi Xu, and Sen Song. 2016. Going deeper with embedded fpga platform for convolutional neural network. In *Proceedings of the 2016 ACM/SIGDA International Symposium on Field-Programmable Gate Arrays*. ACM, 26–35.

[94] Qualcomm. 2017. Snapdragon 845 Mobile Platform. https://www.qualcomm.com/media/documents/files/snapdragon-845-mobile-platform-product-brief.pdf. (2017).

[95] Qualcomm. 2018. Qualcomm Snapdragon 820E Processor (APQ8096SGE). https://developer.qualcomm.com/download/sd820e/qualcomm-snapdragon-820e-processor-apq8096sge-device-specification.pdf. (2018).

[96] Mohammad Rastegari, Vicente Ordonez, Joseph Redmon, and Ali Farhadi. 2016. Xnor-net: Imagenet classification using binary convolutional neural networks. In *European Conference on Computer Vision*. Springer, 525–542.

[97] Herbert Robbins and Sutton Monro. 1985. A stochastic approximation method. In *Herbert Robbins Selected Papers*. Springer, 102–109.

[98] Mathieu ROUAUD. 2012. Probabilités, statistiques et analyses multicritères. (2012).

[99] David E Rumelhart, Geoffrey E Hinton, and Ronald J Williams. 1988. Learning representations by back-propagating errors. *Cognitive modeling* 5, 3 (1988), 1.

[100] Tara Sainath and Carolina Parada. 2015. Convolutional neural networks for small-footprint keyword spotting. (2015).

[101] Jürgen Schmidhuber. 2015. Deep learning in neural networks: An overview. *Neural networks* 61 (2015), 85–117.

[102] Florian Schroff, Dmitry Kalenichenko, and James Philbin. 2015. Facenet: A unified embedding for face recognition and clustering. In *Proceedings of the IEEE conference on computer vision and pattern recognition*. 815–823.

[103] Khurram Shahzad and Bengt Oelmann. 2014. Investigating Energy Consumption of an SRAM-based FPGA for Duty-Cycle Applications. In *International Conference on Parallel Computing-ParCo 2013, 10-13 Sept, Munich*. 548–559.

[104] Rajiv Ranjan Singh. 2007. Preventing Road Accidents with Wearable Biosensors and Innovative Architectural Design. In *2nd ISSS National Conference on MEMS, Pilani, India*. 1–8.

[105] Nitish Srivastava, Geoffrey Hinton, Alex Krizhevsky, Ilya Sutskever, and Ruslan Salakhutdinov. 2014. Dropout: a simple way to prevent neural networks from overfitting. *The Journal of Machine Learning Research* 15, 1 (2014), 1929–1958.

[106] Johannes Stallkamp, Marc Schlipsing, Jan Salmen, and Christian Igel. 2011. The German Traffic Sign Recognition Benchmark: A multi-class classification competition. In *IEEE International Joint Conference on Neural Networks*. 1453–1460.

[107] TexasInstruments. 2018. MSP430FR5994. http://www.ti.com/product/MSP430FR5994. (2018).

[108] Lisa Torrey and Jude Shavlik. 2010. Transfer learning. In *Handbook of research on machine learning applications and trends: algorithms, methods, and techniques*. IGI Global, 242–264.

[109] Varuna Tyagi and Anju Mishra. 2014. A survey on ensemble combination schemes of neural network. *International Journal of Computer Applications* 95, 16

(2014).
[110] James Victor Uspensky. 1937. Introduction to mathematical probability. (1937).
[111] Vincent Vanhoucke, Andrew Senior, and Mark Z Mao. 2011. Improving the speed of neural networks on CPUs. In *Proc. Deep Learning and Unsupervised Feature Learning NIPS Workshop*, Vol. 1. Citeseer, 4.
[112] Chao Wang, Qi Yu, Lei Gong, Xi Li, Yuan Xie, and Xuehai Zhou. 2016. DLAU: A scalable deep learning accelerator unit on FPGA. *arXiv preprint arXiv:1605.06894* (2016).
[113] Lipo Wang, Hou Chai Quek, Keng Hoe Tee, Nina Zhou, and Chunru Wan. 2005. Optimal size of a feedforward neural network: How much does it matter?. In *Joint International Conference on Autonomic and Autonomous Systems and International Conference on Networking and Services-(icas-isns' 05)*. IEEE, 69–69.
[114] Yue Wang, Tan Nguyen, Yang Zhao, Zhangyang Wang, Yingyan Lin, and Richard Baraniuk. 2018. EnergyNet: Energy-Efficient Dynamic Inference. (2018).
[115] P. Warden. 2018. Speech Commands: A Dataset for Limited-Vocabulary Speech Recognition. *ArXiv e-prints* (April 2018). arXiv:cs.CL/1804.03209 https://arxiv.org/abs/1804.03209
[116] Alan R Weiss. 2002. Dhrystone benchmark: History, analysis, scores and recommendations. (2002).
[117] Paul J Werbos. 1990. Backpropagation through time: what it does and how to do it. *Proc. IEEE* 78, 10 (1990), 1550–1560.
[118] Han Xiao, Kashif Rasul, and Roland Vollgraf. 2017. Fashion-MNIST: a Novel Image Dataset for Benchmarking Machine Learning Algorithms. (2017). arXiv:cs.LG/1708.07747
[119] Xilinx. 2011. Spartan-6 Family Overview. https://www.xilinx.com/support/documentation/data$_s$heets/ds160.pdf. (2011).
[120] Shuochao Yao, Yiran Zhao, Aston Zhang, Shaohan Hu, Huajie Shao, Chao Zhang, Lu Su, and Tarek Abdelzaher. 2018. Deep Learning for the Internet of Things. *Computer* 51, 5 (2018), 32–41.
[121] Tom Young, Devamanyu Hazarika, Soujanya Poria, and Erik Cambria. 2018. Recent trends in deep learning based natural language processing. *ieee Computational intelligenCe magazine* 13, 3 (2018), 55–75.
[122] Chen Zhang, Peng Li, Guangyu Sun, Yijin Guan, Bingjun Xiao, and Jason Cong. 2015. Optimizing fpga-based accelerator design for deep convolutional neural networks. In *Proceedings of the 2015 ACM/SIGDA International Symposium on Field-Programmable Gate Arrays*. ACM, 161–170.
[123] Aojun Zhou, Anbang Yao, Kuan Wang, and Yurong Chen. 2018. Explicit Loss-Error-Aware Quantization for Low-Bit Deep Neural Networks. In *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*. 9426–9435.
[124] Xiaojin Zhu and Andrew B Goldberg. 2009. Introduction to semi-supervised learning. *Synthesis lectures on artificial intelligence and machine learning* 3, 1 (2009), 1–130.