Eciton: Very Low-power Recurrent Neural Network Accelerator for Real-time Inference at the Edge

JEFFREY CHEN, SANG-WOO JUN, and SEHWAN HONG, University of California, Irvine, USA
WARRICK HE, Diamond Bar High School, USA
JINYEONG MOON, Florida State University, USA

This article presents Eciton, a very low-power recurrent neural network accelerator for time series data within low-power edge sensor nodes, achieving real-time inference with a power consumption of 17 mW under load. Eciton reduces memory and chip resource requirements via 8-bit quantization and hard sigmoid activation, allowing the accelerator as well as the recurrent neural network model parameters to fit in a low-cost, low-power Lattice iCE40 UP5K FPGA.

We evaluate Eciton on multiple, established time-series classification applications including predictive maintenance of mechanical systems, sound classification, and intrusion detection for IoT nodes. Binary and multi-class classification edge models are explored, demonstrating that Eciton can adapt to a variety of deployable environments and remote use cases. Eciton demonstrates real-time processing at a very low power consumption with minimal loss of accuracy on multiple inference scenarios with differing characteristics, while achieving competitive power efficiency against the state-of-the-art of similar scale. We show that the addition of this accelerator actually reduces the power budget of the sensor node by reducing power-hungry wireless transmission. The resulting power budget of the sensor node is small enough to be powered by a power harvester, potentially allowing it to run indefinitely without a battery or periodic maintenance.

CCS Concepts: • **Hardware → Hardware accelerators**; *Chip-level power issues*; • **Computer systems organization → Embedded systems**; **Neural networks;**

Additional Key Words and Phrases: LSTM, RNN, predictive maintenance, binary classification, multi-class classification, quantization, low power, edge IoT, CPS, neural networks, FPGA, iCE40

ACM Reference format:
Jeffrey Chen, Sang-Woo Jun, Sehwan Hong, Warrick He, and Jinyeong Moon. 2024. Eciton: Very Low-power Recurrent Neural Network Accelerator for Real-time Inference at the Edge. *ACM Trans. Reconfig. Technol. Syst.* 17, 1, Article 16 (February 2024), 25 pages.
https://doi.org/10.1145/3629979

1 INTRODUCTION

Cyber-Physical Systems (CPS), coupled with the **Internet-of-Things (IoT)**, enable deeper, data-driven analytics and insight into our physical world by employing swarms of small, low-power

Authors' addresses: J. Chen, S.-W. Jun, and S. Hong, University of California, Irvine, 6210 Donald Bren Hall, Irvine, CA, 92697-3425; e-mails: {jeffrc2, swjun, sehwanh}@uci.edu; W. He, Diamond Bar High School, 21400 Pathfinder Rd, Diamond Bar, CA, 91765; J. Moon, Florida State University, 2525 Pottsdamer St, Tallahassee, FL, 32310; e-mail: jmoon@eng.famu.fsu.edu.

1936-7406/2024/02-ART16 $15.00
https://doi.org/10.1145/3629979

sensor nodes communicating over a wireless network. A massive amount of data can be collected from a large cluster of nodes, and their deep analysis can generate previously unimaginable quality of information and insight. As a result, the real-world impact of CPS/IoT is rapidly growing and is improving productivity in a wide range of areas including manufacturing [49], agriculture [30], and healthcare [69, 70].

Low power and energy consumption is one of the key goals in the design of a CPS deployment, as deployment scale or non-intrusive sensing requirements can limit access to power infrastructure [30, 48]. As a result, nodes are often expected to function on a scale of years powered by small batteries [52] or power-harvesting modules [2, 43, 48]. These constraints limit not only the overall energy consumption, but also the power consumption at any given time. This limits the amount of computation available on a node, as well as the communication bandwidth over power-hungry wireless communication. In fact, the primary limitation of data collection capacities for low-power nodes is often the power consumption of wireless data transmission [6, 12, 24, 39, 50].

Neural networks are a common machine learning paradigm used for mining data collected via CPS. Specifically, ***Recurrent* Neural Networks (RNN)** are a class of neural networks where each neuron retains some memory of past inputs and are very well-suited to processing time-series data. There are many sub-classes of recurrent neural networks with **Long Short Term Memory (LSTM)** recurrent neural networks have proven especially effective [5, 10, 73]. Due to the high computational intensity of inference with neural networks, data collected by IoT or CPS nodes are typically wireless transmitted to more capable processing units either in the cloud or the edge [41, 45, 61, 68, 84].

Edge Computing attempts to reduce the pressure on the wireless network by moving some computation to the nodes to distill data to smaller sizes. On the one hand, many real-world applications have shown that filtering at the edge can reduce the network data transmission requirement by over 95% [21]. On the other hand, provisioning enough computation performance on end nodes can be costly in terms of power consumption and hardware, defeating the purpose of edge computing.

Hardware accelerators can often remedy this problem of cost and power by achieving high performance on a low power budget [34, 36, 58, 83]. But while existing FPGA-based LSTM accelerators achieve high performance and power efficiency, even they regularly require hundreds of mW to multiple Watts of power, straining the limits of the stringent power budgets of edge nodes [7, 11, 26, 42, 44].

In this article, we present **Eciton**, an extremely power-efficient FPGA-based accelerator for time-series mining using recurrent neural networks at the edge. We show that Eciton can perform real-time inference of simple RNNs and LSTM neural network models of realistic size, at a power budget of 17 mW. Eciton demonstrates competitive performance compared to more power-hungry FPGA implementations, as well as competitive power efficiency against the state-of-the-art. To the best of our knowledge, no prior FPGA recurrent neural network accelerator has demonstrated lower power consumption numbers, especially while maintaining performance on models of similar scale.

To achieve very low power consumption, Eciton employs 8-bit fixed-point quantization of weights, hard sigmoid activation functions, as well as a carefully optimized microarchitecture to reduce the chip resource and memory requirements. As a result, the Eciton inference accelerator can fit in a low-power, low-cost, but resource-constrained Lattice iCE40 UP5K FPGA. The UP5K was uniquely positioned for this application, thanks not only to its low cost (~$5) and μW-scale static power consumption, but also relatively large on-chip SRAM resources (128 KiB), which is large enough to host real-world recurrent neural network models after 8-bit quantization. The UP5K is also equipped with four DSP blocks for fast arithmetic, both of which are unavailable in similarly positioned low-power FPGAs such as the Microsemi IGLOO or other Lattice iCE40 chips.

We evaluate Eciton on four different recurrent neural network models and measure their performance and accuracy. The tested neural networks include two LSTM models for predictive maintenance, another LSTM model for urban sound classification, and one RNN model for network intrusion detection on IoT nodes. We measure that quantization approaches result in a modest accuracy difference of ~5% when evaluated on real-world predictive maintenance LSTM models with 3- to 4-layers, ~10% on real-world recurrent neural network models including network intrusion detection.

This article claims the following contributions:

— Demonstrates an LSTM and RNN accelerator design that fits in a low-power FPGA and a 17 mW power budget.
— Demonstrates real-world recurrent neural network models can be compressed to fit within such a low-power accelerator at a modest loss of accuracy.
— Demonstrates real-time performance on realistic applications including predictive maintenance and multi-classification models such as intrusion detection.
— Demonstrates that a low-power accelerator at the extreme edge can actually reduce the power consumption of a CPS/IoT node by reducing network transmission requirements.

The rest of this article is organized as follows: Background and related works are explored in Section 2. We present the architecture of Eciton and our neural network compression methods in Section 3. We provide in-depth evaluation of Eciton in the context of a CPS deployment of compressed neural networks in Section 4. We conclude with future work in Section 5.

2 BACKGROUND AND RELATED WORKS

2.1 Predictive Maintenance

Predictive maintenance, or condition-based maintenance, aims to monitor mechanical systems for signs of imminent failure to preemptively perform maintenance operations. Compared to conventional, periodic maintenance, accurate predictive maintenance can significantly reduce downtime and cost [29].

Modern predictive maintenance approaches integrate pervasive data collection of CPS and IoT into target mechanical devices to achieve more accurate prediction compared to manual inspection [29]. Depending on the mechanical properties of the target, information including vibration, humidity, temperature, pressure, sound, electrical current, and inductance can be collected from various locations in the device, which are analyzed using various statistical and machine learning-based approaches [29, 33, 48, 64, 78].

2.2 Multi-class Classification

Multi-class classification applications present a different arrangement, making the assumption that each input is assignable to one and only one label. The applications can classify each input as one label among multiple labels (three or more) based on patterns trained from the feature properties of the data collected [8]. Such models are used when each input must be classified into multiple categories. For example, the UrbanSound8K dataset and model tries to determine what kind of sound each audio snippet is [59] and tries to assign each audio clip to one of 10 classes, spanning from air conditioner to gunshot. The network intrusion detection model [53] tries to classify a stream of packet patterns into five categories: a *safe* and normal behavior, and four different classes of attack patterns.

For the final layer of a multi-class classifier neural network, a softmax function is typically used over the array of activation values to determine which class has the highest probability. [19].

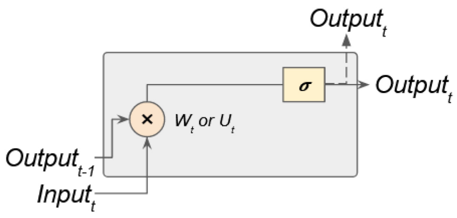


Fig. 1. Recurrent neural network cell architecture.

2.3 RNN Recurrent Neural Network

A recurrent neural network is a neural network class distinguished by their internal cell memory states, as the cells take information from prior timestep inputs to influence the current input and output [9]. The internal cell state is referred to as the *hidden state* and is updated every timestep, summarizing the unique necessary information from prior timesteps [60].

Figure 1 shows the internal architecture of a single RNN cell. The kernel weight W is multiplied against the input, while the recurrent weight U is multiplied against the hidden state, after which the aggregate result is processed through the activation function, commonly sigmoid (σ), and used as the the output or the next hidden state. The hidden state records past information across timesteps and is integrated with current input to make accurate inferences [60].

Because of RNNs' ability to process information sequentially in time, there is strong interest in executing RNNs on embedded devices, which can improve real-time response efficiency and improve system security compared to cloud computing [57]. However, the computational and memory requirements, mainly from the multiply-accumulate operations, create challenges in maintaining the real-time response goal, driving optimizations of RNNs and embedded systems to overcome these limitations.

2.4 LSTM Recurrent Neural Network

LSTM is an RNN architecture that can better handle long-term dependencies in the input data.

Classical RNNs suffer from the so-called *vanishing gradient* problem, wherein the effects of an earlier input quickly become too small to be useful [60].

LSTMs handle this problem by controlling gradient degradation using three *gates*. This adds three more trainable weights per cell, which can learn the best rate of gradient decay. As a result, LSTMs can learn relationships between events happening millions of timesteps apart [63], making them useful for many real-world time series data mining applications [74, 75].

Figure 2 shows the internal architecture of a single LSTM cell. Each weight (f, i, $\widetilde{c}$, and o) represents a multiplication and accumulation operation against both input and recurrent values that regulates the retainment of past information [22, 57].

LSTMs also use two classes of activation functions: kernel and recurrent activation functions. As seen in Figure 2, typically used activation functions are sigmoid (σ) and tanh, respectively[57].

LSTMs are typically designed with multiple layers, as well as a *dense* layer at the top layer. The dense layer is a conventional fully connected neural network layer, usually with the same number of units as the number of classes to detect.

2.5 Neural Network Compression

Compressing neural networks, either by reducing the number of weights (*pruning*) or the bit width of each weight value (*quantizing*), is an important tool for reducing the computation and memory overhead of neural networks, often with negligible loss of accuracy [27]. Pruning involves removing weights with less impact on the final result, and then re-training the model without these

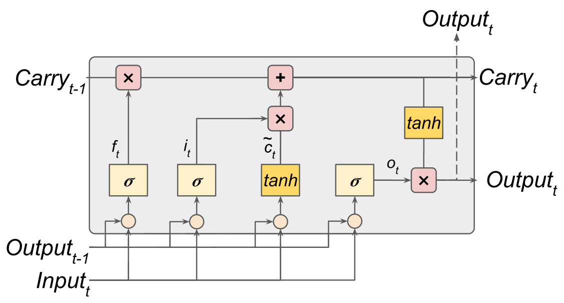


Fig. 2. Long-short term memory cell architecture.

weights [57]. Weight matrix sparsification, however, can reduce accuracy, along with negatively impacting weight memory organization, irregularity of which often results in increased computation time and hardware for data management.

Another compression technique, quantization, maps the range of original values to a smaller set of values using either a linear or non-linear mapping function. A variance-based cutoff may also be used to limit the range of the source set to account for outliers. A popular instance of quantization is to map 4 byte floating-point to 1 byte fixed-point [16, 31, 72, 82]. This not only reduces memory requirements, but also substitutes costly floating point operations with cheaper fixed-point arithmetic [35, 67, 79].

To best map a set of values from floating point to fixed point sets in a linear way, the range of float values R_{float} is obtained from the floating point set, as well as the possible signed integer range of the quantization target R_{fixed} (-128~127 for 8 bits). To maintain accuracy even when outliers may exist in the float set, we can also calculate R_{float} using a user-defined standard deviation. These two ranges can form a scale: R_{fixed}/R_{float}, which can be used to map each value. If there is a difference in range medians, then a zero point adjustment can be made using a bias term.

Quantization-aware training is available when direct model quantization results in significantly reduced accuracy.

2.6 Accelerators on CPS End Nodes

Augmenting CPS/IoT end nodes with reconfigurable hardware accelerators such as FPGAs can help reduce the power consumption of required computation, as well as power-efficiently implement edge computing on end nodes [17, 83]. Existing research has explored offloading end node operating system functions [54], encryption for security [66], and communication protocols [23], as well as application-specific accelerators for video processing [36, 83], person detection [58], and statistical time series mining [34] into FPGAs. Much research has gone into optimal management of FPGAs on end nodes as well, including dynamic partial reconfiguration and power gating [71]. Industry offerings also exist for low-power convolutional neural networks at the edge, including the Lattice SensAI platform [65].

Power-efficient neural network implementation on FPGAs for the edge has also been a popular research topic [28, 32, 81], including LSTMs [42, 44]. However, performance and model size requirements have prevented realistic LSTM models from fitting in the low-power class FPGAs such as Lattice iCE40 or Microsemi IGLOO series, limiting them to relatively performance-oriented FPGAs with 100s of mW of static power. This may limit their use in resource-constrained CPS/IoT scenarios with mW-level power budgets [2, 48].

Following our prior work [14], additional LSTM acceleration on FPGA projects have been developed. An optimized FPGA design focused on the LSTM cell modeled for traffic speed prediction

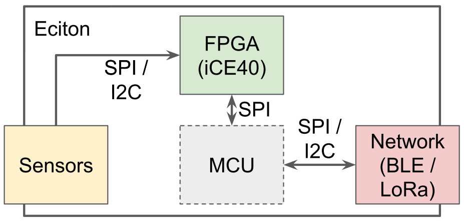


Fig. 3. Eciton end node architecture.

and performed on a Spartan-7 XC7S15 [55]. ASIC designs for LSTM acceleration have also been proposed, utilizing pruned bit-sparse representation instead of fixed-point representation [15].

2.7 Wireless Communication for CPS/IoT

There is a wide range of available wireless communication technologies for CPS and IoT systems [24, 46]. They typically represent a tradeoff between communication bandwidth, range, and cost/power consumption, spanning from LTE or GSM-based technologies such as LTE-M and MB-IoT with hundreds of kbps of performance as well as hundreds of mW of power consumption [39], to **Low-Power WAN (LPWAN)** technologies like LoRa with an order of magnitude bandwidth and power consumption [1, 12]. Reducing network transmission requirements with effective edge computing may allow use of low-power technologies while still maintaining effective data collection bandwidth.

3 ECITON NODE ARCHITECTURE

Eciton augments a CPS/IoT end node with a flexible, configurable accelerator that can process both RNN and LSTM models with minimal configuration. The RNN accelerator is implemented on a low-cost, low-power Lattice UP5K FPGA, adding minimal cost and power consumption to the overall system. In Eciton, collected sensor data is first evaluated by the accelerator, which runs either an RNN or an LSTM model, and the sensor data is sent wirelessly to a central server only when potential failure is detected locally. At the central server, a more complex software system can then make higher-level decisions based on data from multiple nodes.

Figure 3 shows the overall architecture of an Eciton end node. Sensor data is first filtered by the FPGA, reducing the data traffic and workload of the **microcontroller unit (MCU)**. In the current version of Eciton, the MCU is only responsible for the network interface, transferring the accelerator output wirelessly to a remote host. This design decision is due to the complexities of managing the wireless network module, including executing the protocol.

3.0.1 Adaptable LSTM Accelerator. Internally, Eciton implements an LSTM accelerator, which can be easily configured to handle many classes of neural network models, such as the four recurrent models we evaluate in this manuscript. The accelerator module consists of two separate cores, each for LSTM and dense layers, as seen in Figure 4. The MCU is responsible for loading the LSTM and dense layer cores with weights, after which sensor data can stream through both cores for inference. This architecture can be adapted to handle different recurrent models. For example, the dense core can be bypassed for multi-classification models that do not need the dense layer, and parts of the LSTM accelerator can be disabled for the simpler RNN models.

3.1 FPGA Resource Restrictions

Our target platform is the Lattice iCE40 UP5K FPGA, which is a low-cost (~\$5), low-power (μW-scale static power) FPGA with stringent resource restrictions. This choice was driven by two

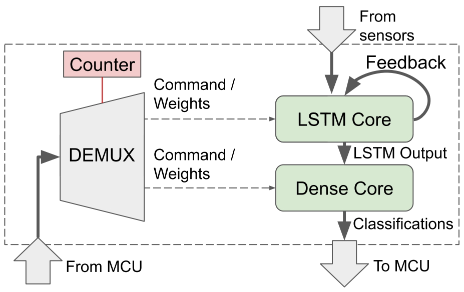


Fig. 4. The MCU provides weights to LSTM and Dense cores, after which the accelerator can process input from sensors.

factors: larger on-chip memory capacity and DSP block availability, compared to comparable low-cost, low-power offerings.

3.1.1 On-chip Memory. The UP5K includes 1,024 kbits of on-chip SRAM provided in the form of four **Single-Port RAM (SPRAM)** blocks, in addition to usual embedded Block RAM. Most other FPGA offerings of similar class, including the Microsemi IGLOO [47] or other Lattice iCE40 FPGAs [37], do not include such memory resources, limiting their on-chip memory to mere 32 to 128 kbits of block RAM. On-chip memory capacity is very important for low-power, high-performance neural network inference, since off-chip memory access is harmful for both performance and power efficiency. This is especially important for low-power FPGAs and MCUs, as they typically do not have high-performance memory interfaces such as DDR3.

However, after evaluating a wide range of published, real-world recurrent neural network models, we discovered that even the additional SPRAM is not enough for neural network models of realistic size. To remedy this, we explored various neural network compression approaches in an attempt to make the model fit entirely on-chip. We describe our approach in Section 3.2.

3.1.2 DSP Block Availability. In addition, the UP5K provides eight 16-bit **Digital Signal Processing (DSP)** blocks for fast arithmetic. While this is a larger number compared to similar offerings, it is nowhere enough to implement fast floating point operations, or even wide integer division. Our selection of quantization and hard sigmoid activation function parameters was the result of optimizing performance and accuracy within these constraints.

3.2 Quantizing RNN/LSTM Models

Eciton performs post-training linear quantization of weights from 4-byte floating point to 8-bit dynamic fixed point, reducing memory footprint by 1/4. We discovered 8-bit quantization resulted in the best size-accuracy tradeoff, which will be presented in more detail in Section 4.2.

To reduce chip resource requirements, Eciton uses the piece-wise linear *hard sigmoid* activation function for both kernel and recurrent activation functions of LSTM, instead of non-linear functions such as sigmoid or tanh. Figure 5 shows the three activation functions. While non-linear functions were too complex to fit on our target FPGA, we discovered that hard sigmoid can be implemented efficiently while providing a more accurate approximation of the non-linear functions, compared to simpler functions such as the **Rectified Linear Unit (ReLU)**. Quantizing models does require tuning the float range to consider both accuracy in calculations and in mapping, which will be discussed in Section 4.2.

One important consideration when designing an LSTM for quantization and hard sigmoid activation is the importance of bias weights. While omission of bias weights sometimes makes smaller but still accurate LSTM models possible, we have discovered that bias weights are crucial for

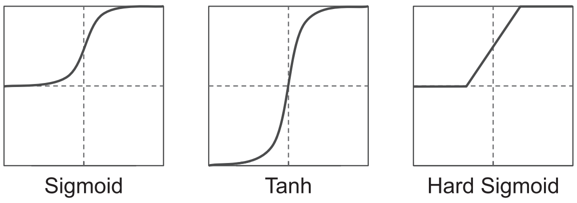


Fig. 5. Comparison between three activation functions.

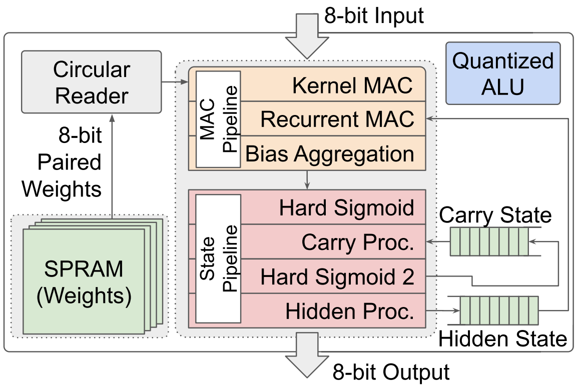


Fig. 6. Microarchitecture of the LSTM core.

accuracy when using hard sigmoid activation functions. This is because, in classical LSTMs, two different activations with different output ranges are used: sigmoid at $(0, 1)$ and tanh at $(-1, 1)$. A bias term can be trained to map the hard sigmoid output domain to the two different domains of sigmoid and tanh.

3.2.1 *Why Linear Quantization.* While non-linear, logarithmic quantization approaches can express a wider range of values, resulting in better accuracy, dynamically translating and performing quantized arithmetic on such non-linearly mapped values required more on-chip resources. Since our LSTM accelerator did not leave much room in the target FPGA, we decided on linear quantization for better performance.

3.2.2 *Why Not Prune.* While pruning neural networks often reduce model size significantly without much accuracy loss, Eciton did not use pruning for two reasons: First, pruning requires retraining, which adds a hurdle to deploying the system. Second, the sparsity resulting from pruning requires additional chip resources to handle.

3.3 LSTM Core Architecture

Figure 6 shows the LSTM core architecture. Input and output is done in units of 8-bit words. Eciton's batch size is one, to simplify the memory mapping and management as explained below. All weights are stored across the four available on-chip SPRAM blocks combined into a single address space, exposing a single 16-bit interface. The computation pipeline consists conceptually of two sub-pipelines: the **Multiply-Accumulator (MAC)** pipeline, and the State pipeline for calculating carry and hidden states. Both pipelines share a single quantized ALU Module.

3.3.1 *Weight Memory Map.* Weights are stored in a layer-major order, so for each input tuple to propagate through all layers, the whole weight memory is linearly scanned exactly once by a very simple circular reader module.

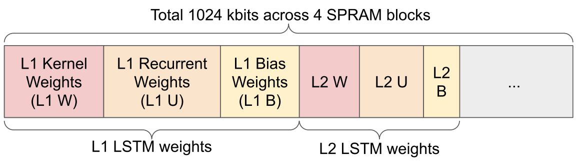


Fig. 7. LSTM weights are stored in layer-major sequence.

Figure 7 shows an example memory layout of weights for a model with two LSTM layers. The weights are stored in the order of kernel weights, recurrent weights, and bias weights, for each LSTM layer. Each weight is a four-byte tuple, corresponding to the four 1-byte weights of each LSTM cell. The microarchitecture orders computation in the same order, so weights can simply be linearly scanned for correct operation.

Within each class, the weights are ordered in an input-major order, meaning if the network has multiple columns fed into multiple cells, the weights for each column across all cells are grouped together. Each weight consists of a sequence of four-byte tuples, each tuple corresponding to the four weights of each LSTM cell. The tuples store four 1-byte weight values for the *input*, *cell*, *forget*, and *output* gates, in that order.

3.3.2 *MAC Pipeline.* The MAC pipeline is where the majority of the arithmetic for LSTM is done. While computation is conceptually organized as a pipeline, Eciton actually processes each step in non-overlapping sequence to efficiently reuse a small number of quantized arithmetic units. Computation is organized in the order of weight storage, i.e., *kernel*, *recurrent*, and *bias* weights.

For kernel weights, each 8-bit input can be independently multiply-accumulated against each of the four kernel weights of all LSTM cells in this layer. This results in $Unit$ number of four-byte tuples in a BRAM FIFO, where $Unit$ is the number of LSTM cells in this layer. By the time kernel MACs are done, the recurrent weights are already loaded and ready in the read queue. The same MAC process is performed, reusing the same ALU hardware, using the *hidden state* data of the previous timestep, if any.

MAC operations are done independently per gate (input, cell, forget, output), resulting in $Unit$ number of four value tuples. Again, thanks to the convenient memory mapping, by the time kernel MACs are done, the recurrent weights are also already loaded and ready in the read pipeline.

Once both stages are done, the results of both are forwarded to the bias aggregation stage, where the two streams are added with the bias weights conveniently ready in the read pipeline. At the end of this stage, we have a $Unit$ number of four value tuples stored in a BRAM FIFO.

3.3.3 *Carry and Hidden State Pipeline.* Once the MAC operations are done and the $Unit$ number of 4-byte tuples are generated, we can calculate the carry states for the next timestep and the hidden states for the next layer according to the LSTM algorithm. This process also re-uses the same quantized ALU module to perform hard sigmoid, quantized multiplication and addition.

Another notable feature of the layer-major weight map and computation organization is that the memory for carry state and hidden state results can each be stored in a single large FIFO implemented in BRAM, instead of a complex memory map for each layer or unit. Because the computation is done sequentially in a layer-major order, the hidden state FIFO will always hold the values for the next layer. By the time all layer processing is done and computation comes back to the first layer for the next timestep, the carry state for all of the layers will be waiting in order in the carry state FIFO.

3.3.4 *Quantized ALU Architecture.* Our quantized ALU provides three functions: quantized addition, multiplication, and hard sigmoid. While addition between two quantized values is straightforward, multiplication is slightly more complicated. Because linear quantization involves a scale

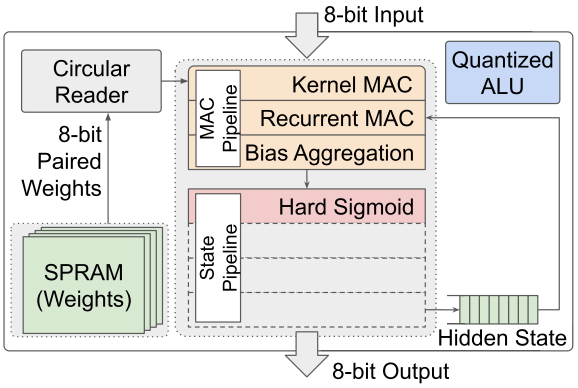


Fig. 8. Microarchitecture of the RNN Core.

factor mapping one set of values to another, multiplying two scaled values results in multiplication of the scale factor as well, and the multiplication results need to be divided with the scale factor again to maintain correctness.

While division in hardware is resource-intensive, fortunately, the scale factor is a static value determined during quantization. We pre-calculate a normalized division `(1≪16)/scale` during quantization. This value can simply be multiplied to the multiplication results using the UP5K DSP block, which supports 16-bit multiplication to a 32-bit value. The resulting value can be shifted down 16 bits to get the correct division results. Since the original multiplication results are 16 bits wide, and because weights and input are all 8 bits, we do not lose any correctness with this approach. As a result, quantized multiplication requires two DSP blocks, one for multiplication and one for division by the scale factor.

The hard sigmoid module also requires a multiplier, but, since our computation organization never requires multiplication and sigmoid calculation to happen simultaneously, our ALU is designed to share one internal multiplier.

Because the SPRAM memory interface is 16 bits wide, two weights are delivered per cycle. Since the four weights in a cell are processed independently from each other, our quantized ALU internally implements two separate modules to operate in parallel, consuming 4 of the 8 available DSP blocks. We note that although memory bandwidth could be easily increased by using SPRAM modules in parallel, our experiments showed that this would not lead to high performance, as it was difficult to fit more ALU modules in the limited chip space.

3.3.5 Adapting for Recurrent Neural Networks. For traditional recurrent neural networks that are not LSTMs, the four gates and their associated memory and computation requirements are removed. In this situation, parts of the LSTM accelerator can be disabled to handle such models, as seen in Figure 8. All of the gate-related calculation is bypassed, as well as most of the state pipeline except for the first hard sigmoid activation. This modification enables RNN models with more features and layer dimensionality to be utilized than would be limited by the required memory cell components of the LSTM, and also allows for non-sequential input models.

3.4 Dense Core Architecture

The dense core is a separate entity from the LSTM core, as depicted in Figure 9. Its architecture is similar to the architecture of the LSTM cell, but much simpler. Since the dense layer is a classical, feed-forward, fully connected neural network, there is no need for handling recurrent paths or weights. The dense core only has three computation stages: kernel MAC, bias aggregation, and a

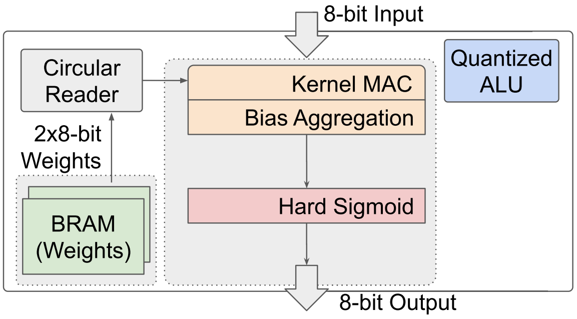


Fig. 9. Microarchitecture of the Dense Core.

Table 1. Models Evaluated on Eciton

Name	Class	Recurrent Layers	Dense Layers	Classification
Turbofan [62]	LSTM	2	1	Binary
Motor [48]	LSTM	3	1	Binary
UrbanSound [59]	LSTM	2	1	10 classes
Intrusion [53]	RNN	1	1	5 classes

final hard sigmoid. It also uses an on-chip block ram for weight storage instead of SPRAM, since the number of weights for the final dense layer is typically very small. The final output of the dense layer is a vector of quantized values representing each prediction class. This is the final inference result that is relayed to the software on the MCU. Since the amount of dense layer computation is nearly negligible compared to the LSTM layer, the quantized ALU of the dense block ALU is designed to use only one MAC unit, using only two DSP blocks and less supporting logic.

4 EVALUATION

The accelerators of Eciton were implemented using open-source toolchains Icestorm [76] and Bluespec [51], on a low-cost, low-power Lattice iCE40 UP5K FPGA, coupled with an Arduino Nano as the MCU [4].

The following sections present evaluation results of Eciton, including the neural network models we have trained and used for predictive maintenance datasets and multi-class classification datasets, as well as the performance and efficiency of our FPGA accelerator implementation.

4.1 Neural Network Models and Datasets

We demonstrate Eciton on four, real-world recurrent neural network models with various characteristics. The evaluated models are summarized in Table 1. The models span a wide range of characteristics, across LSTMs and RNNs of variable size, as well as binary and multi-class classification.

The models include two predictive maintenance binary classification scenarios: Turbofan engine maintenance dataset from NASA [62], as well as electrical motor maintenance using vibration and humidity [48]. We also demonstrate Eciton on two multi-class classification scenarios: the UrbanSound8K classification dataset [59] and the NSL-KDD dataset [53]. The recurrent neural network models were first trained using Tensorflow and Keras, quantized post-training, and finally loaded on the Eciton system. Table 2 lists the calculations for the requisite recurrent layer weight counts derived from the shapes of the models; $coeff_{layer}$ is 1 for RNN layers and 4 for LSTM layers, with the latter representing the 4 gate weights.

Table 2. Recurrent Layer Weight Count from Parameters

Weightset	Weightcount
Kernel	$Unit \times Seq.Width \times coeff_{layer}$
Recurrent	$Unit^2 \times coeff_{layer}$
Bias	$Unit \times coeff_{layer}$

Table 3. NASA Turbofan Predictive Maintenance Model

Layer	LSTM1	LSTM2	Dense	
Seq. Length	50	50	–	
Seq. Width	25	100	–	
Unit	100	50	–	
Kernel	10,000	20,000	50	
Recurrent	40,000	10,000	–	
Bias	400	200	1	
Total	50,400	30,200	51	**80,651**

We demonstrate two types of recurrent neural network models: LSTM and RNN. We present two different classes of models for two purposes: First, we demonstrate that the Eciton architecture can be conveniently adapted for different model classes. Second, as the RNN model is also the smallest model we evaluated, it serves as an important part of the suite of real-world applications across a wide range of scales.

We also note that, since RNN models have fewer weights compared to LSTMs of the same dimensions, the RNN accelerator can support models with larger dimensions than the LSTM-configured accelerator. However, we could not find a convenient open dataset to demonstrate this in the Eciton context.

This section presents the original floating-point trained models and provides more detailed analysis of quantization efficiency in Section 4.2.

4.1.1 Turbofan Engine Maintenance. The NASA Turbofan Engine Maintenance Model is a predictive maintenance model that simulates the failure of a turbofan engine, with the dataset being an engine degradation simulation using C-MAPSS. The model dataset includes input from 25 different sensors. Our model uses two LSTM layers with 100 and 50 nodes, as well as a single-cell dense layer. The total number of weights adds up to 80,651, as shown in Table 3.

The model before quantization reached a competitive accuracy of 97%.

4.1.2 Electrical Motor Maintenance. The Electrical Motor Maintenance model is trained on four input data streams, three accelerometer axes and a humidity sensor, collected from an electrical motor in the process of spinning down. The accelerometers are sampled at 2 KHz, while the humidity sensor is sampled at 50 Hz. This scenario has interesting real-time and energy-budget requirements, because the system is powered by a power harvester generating energy from the magnetic field as the motor spins.

Because power harvesting becomes unavailable after the motor has shut down, the collected data must be either processed or transmitted after detecting motor shutdown within a budget of 0.6 Joules of harvested energy.

The model uses three LSTM layers each with 128, 32, and 16 nodes and a single-cell dense layer adding up to a total of 91,873 weight values, as seen in Table 4. The model before quantization has a competitive trained accuracy of 90%.

Table 4. Electrical Motor Predictive Maintenance Model

Layer	LSTM1	LSTM2	LSTM3	Dense	
Seq. Length	12,500	12,500	–	–	
Seq. Width	4	128	32	–	
Unit	128	32	16	–	
Kernel	2,048	16,384	2,048	32	
Recurrent	65,536	4,096	1,024	–	
Bias	512	128	64	1	
Total	68,096	20,608	3,136	33	**91,873**

Table 5. Urban Sound Multi-classification Model

Layer	LSTM1	LSTM2	Dense	
Seq. Length	173	173	–	
Seq. Width	128	128	–	
Unit	100	50	–	
Kernel	51,200	25,600	500	
Recurrent	40,000	10,000	–	
Bias	400	200	10	
Total	91,600	35,800	510	**127,910**

4.1.3 Urban Sound Classification. The Urban Sound multi-class classification model is trained on the UrbanSound 8K dataset divided into 10 sound classes, including gunshots and siren noises, with audio from the dataset pre-processed into frequency-based dimensional features according to mel spectrogram [40]. It has two LSTM layers and one Dense layer. There is a total of 181,642 weights.

The model before quantization has a competitive trained accuracy of 84.25%. The original model is variable length, however, since a great majority (~84%) of the data consisted of 173 timesteps (the near-maximum timestep allotment barring 7 cases of 174 timesteps), we fitted the model to 173 as the sequence length. Accuracy for the modified model trained this subset was retained at 84.23%.

Since this model is large, we iteratively reduced and tested LSTM1 unit from 128 to 100 and LSTM2 unit from 64 to 50 and found that accuracy was not impacted significantly from this change (84.16%). This results in 127,910 weights, shown in Table 5, though in practice 127,400 weights are kept on-chip, as the dense layer weights are moved off-chip.

4.1.4 Network Intrusion Detection. The Intrusion Detection RNN-IDS multi-class classification model is trained on the NSL-KDD dataset, widely used in intrusion detection experiments, classified into five categories (four attack categories and one normal behavior category) [80] from packet transmission patterns. Non-numeric features of the NSL-KDD dataset are converted into binary-coded vectors. The model utilizes a simple RNN layer and a Dense layer. A total of 16,645 weights were implemented, as outlined in Table 6. This model has a trained accuracy of 81.29% prior to quantization.

4.1.5 On-chip Weight Capacity. We compare the pre-quantized models with 32-bit weights versus quantized models with 8-bit weights, stored in concatenated pairs per 16-bit address, in Figure 10. The iCE40 SPRAM supports a total of 128 KB in weight storage, and the Turbofan, Electrical Motor, and Urban Sound Classification models were significantly above capacity, each exceeding the capacity by more than double. Quantized models all fit within the SPRAM capacity, including

Table 6. Intrusion Detection Multi-classification Model

Layer	RNN	Dense	
Seq. Length	1	–	
Seq. Width	122	–	
Unit	80	–	
Kernel	9,760	400	
Recurrent	6,400	–	
Bias	80	5	
Total	16,240	405	**16,645**

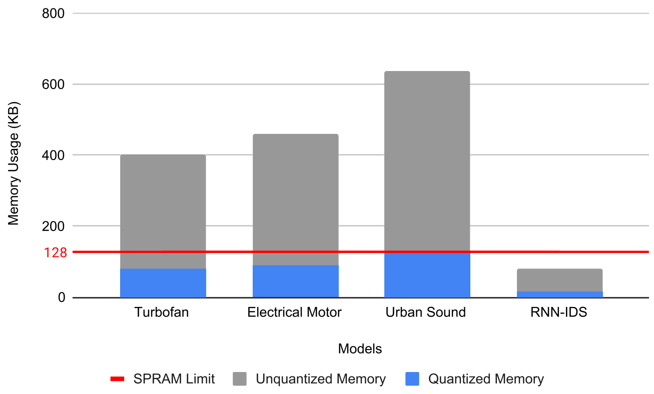


Fig. 10. Model weight comparisons, quantized and unquantized, against on-chip SPRAM memory capacity.

the Quantized Urban Sound Classification, which is just under the 128 KB limit at 127.4 KB. The smallest quantized model, RNN-IDS, uses 12.6% of available SPRAM memory.

4.2 Quantized Model Accuracy

Figure 11 shows the effect of various quantization approaches on accuracy on the turbofan model. While use of hard sigmoid, 16-bit and 8-bit fixed-point quantizations have gradual loss of accuracy, 4-bit quantization results in a sharp, 24% drop. While the 16-bit quantized model is sufficiently small to fit in the on-chip SPRAM, we decided on 8-bit quantization to make better use of the limited DSP blocks, performing more 8-bit operations per cycle. The same behavior was observed in other models as well, where a sharp loss in accuracy occurred after 8-bit quantization. We decide that 8-bit quantization is a good balance between performance, resource-efficiency, and accuracy; other industry and academic researchers appear to agree with our analysis, based on the offerings that exist. In fact, 8-bit quantization seems to have become an industry standard for edge machine learning [16, 31, 77, 82].

Post-training quantization on multi-classification models had significant accuracy drops, immediately falling to 60%. Therefore, quantization-aware training was conducted to preserve accuracy. However, training software did not support hard sigmoid activations during quantization-aware training. To circumvent this issue, we first trained the models using normal sigmoid activations. After the models were trained, its weights were extracted and additionally retrained in a standard model that did use hard sigmoid in the activation layers. 8-bit quantization was conducted after all of these training stages. The accuracy drops per step across all models are shown in Figure 12, and

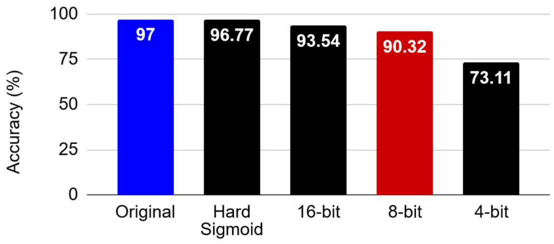


Fig. 11. Accuracy of the turbofan model drops sharply at 4-bit quantization.

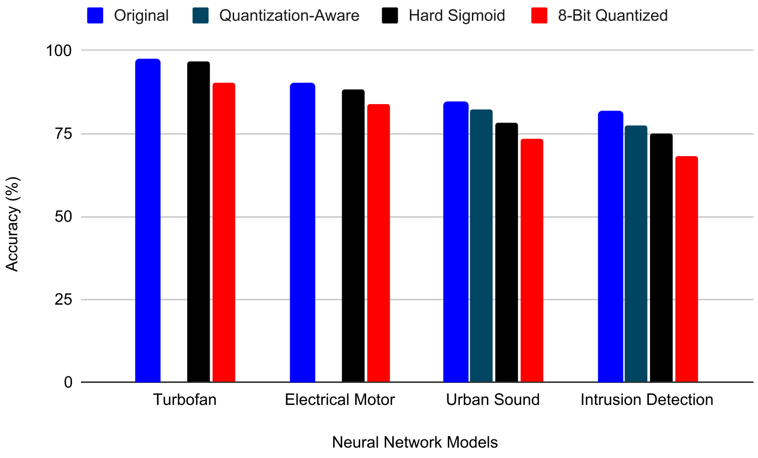


Fig. 12. Accuracy of all the models through training and quantization.

Table 7. Models' Accuracy through Processing

Model	Turbofan	Electrical Motor	Urban Sound	Intrusion Detection
Baseline	97	90	84.25	81.29
Quantization-aware	-	-	82.1	77.4
Hard Sigmoid	96.77	88.3	78.3	75.2
8-bit Quantized	90.32	84	73.4	68.2

the accuracy values are presented in Table 7. Both of the multi-classification models experienced greater accuracy drops than the binary classification models. There is greater accuracy loss with the Urban Sound model, compared to the Turbofan and Electrical Motor models. The intrusion detection model experiences the greatest accuracy loss, despite best efforts.

In the end, all evaluated models suffered an accuracy loss spanning 6 to 13 percentage points. This result is in part due to the fact recurrent neural networks are significantly more susceptible to accuracy loss from quantization [18]. However, existing work and industry offerings for edge machine learning appears to accept such accuracy degradations from quantization, demonstrated by many offerings that focus exclusively on quantized arithmetic while suffering similar accuracy losses [20, 44]. The benefits of the order of magnitude power and performance improvements from quantization often outweigh the accuracy losses, especially since edge deployment may simply not be feasible without it.

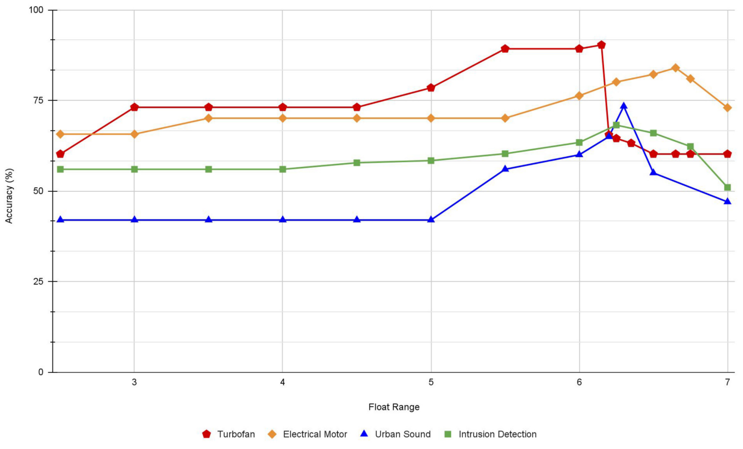


Fig. 13. Accuracy of quantized models according to assigned float range.

We also note that our uniform distribution quantization was done in a straightforward, linear manner to minimize resource utilization. If more accuracy is absolutely needed by the application, then it is possible to more aggressively minimize accuracy loss with more comprehensive and rigorous quantization-aware optimization techniques, such as asymmetrical or biased quantization [3, 56]. Such an approach will likely exceed the chip capacities of our target iCE40 chip, but we could solve this issue by provisioning more resources, e.g., using two iCE40 chips instead of one.

Since we used linear quantization, deciding the valid float ranges for quantization was an important step for accurate quantization. To conserve chip resources for the quantized arithmetic unit, we chose a single float range selection across all layers after evaluating a range of ranges. Figure 13 shows the quantized accuracy fluctuations as the float range was adjusted, for all models. We see that the quantized accuracy is affected by the defined float range, with the x-axis values depicted representing defined float range of (-x, x). Smaller quantized ranges gradually fall in accuracy, while the accuracy for larger ranges experience a more precipitous drop in accuracy. The minimum defined float range in the figure is 2.5 due to the hard sigmoid limits from activation functions.

The minimum float range of 2.5 occurs because otherwise comparisons to the hard sigmoid limits is not possible. In the lesser flatlining ranges, these are usually caused by the initially more-distributed unquantized values being truncated and grouped together in float ranges to quantized values such that quantized arithmetic in the nodes in the model become stagnant. Outside of these ranges, accuracy gradually climbs, as the float range expands to accommodate a range of values that fit unquantized values to more appropriate quantized values until a peak is reached and then falls; at this point, the float range has become overly large, with more space between them, causing less-distributed unquantized values to experience a greater relative range shift from their original position.

4.3 FPGA Resource Utilization

The resource utilization for Eciton is presented in Table 8. All four available SPRAM blocks, as well as six of the eight available DSP blocks, are used. All four SPRAM blocks are utilized for the potential to store as many weights as possible. Each MAC unit requires two DSP blocks, so four DSP blocks were needed for the LSTM cell, and two DSP blocks for the dense cell. Overall, the accelerator uses 94.5% of the board's logic cells, while achieving a clock frequency of 17.0 MHz.

The total on-chip capacity of the four SPRAM blocks is 256 KB, which can comfortably fit either of our trained models and potentially significantly larger ones as well.

Table 8. Eciton On-chip Resource Utilization

Resource	Resource Utilization	Percent Utilization (%)
LC	4,987/5,280	94.5
SPRAM	4/4	100
BRAM	22/30	73.3
DSP	6/8	75

4.4 Performance Evaluation

We evaluate the performance and power efficiency of Eciton against multiple system configurations, including Keras and our best-effort software implementation on an Intel core i7-8700K CPU, as well as a Raspberry Pi Zero, an Arduino Uno (ATmega328), and an Arduino Due (ARM Cortex-M0). We have implemented both quantized and non-quantized models in software and evaluated both on possible platforms.

The Raspberry Pi Zero was chosen, as it regularly demonstrated good power-efficiency among similar Linux-enabled embedded systems offerings. We emphasize that the both Arduino systems do not have enough on-chip SRAM to hold even the quantized model. To obtain an upper-limit estimate of performance on the Arduino boards, we measured the performance of the software modified to re-use a much smaller set of weights, resulting in extremely poor accuracy.

Figure 16 presents the performance and power efficiency evaluations of the four models we evaluate. The first three models, turbofan, motor, and urban sound, show very similar performance relations. Both keras and our hand-optimized software on the i7 vastly outperforms all embedded systems. But its high power consumption and low power efficiency make it a poor fit for the edge, and it is presented here just as a point of reference. Compared with embedded systems such as the Raspberry Pi and the Arduinos, Eciton vastly outperforms the comparison systems. We also note that on the i7 and RPi, the floating point implementations outperform quantized ones due to the added overhead of quantized arithmetic operations, specificially the added normalization requirement after each operation.

One exception to this pattern is the intrusion detection model on the RPi, where the overhead of more computation was offset by the reduced size of the model fitting better on the small on-chip cache. Figure 16(d) shows the power and performance evaluations of the intrusion detection model, which shows different performance and power efficiency relations. Note that this figure is presented in logarithmic scale, unlike the other three, due to the wide range of scales. In this example, the performance of the Raspberry Pi zero vastly exceeds the Eciton performance, resulting in better power efficiency compared to Eciton. According to our profiling, this is mainly due to the tiny size of the model, as shown in Figure 11. The small size of the model means that it can completely fit within the L1 cache of the CPU systems, resulting in very high performance, as well as high power efficiency.

4.5 Power-efficiency Evaluation

For power efficiency, we measured the total power consumption during execution of the turbofan neural network on all systems *except the i7 system*. For the iCE40 FPGA, we used the Lattice Diamond toolchain's built-in Power Calculator tool to measure the design's power consumption [38]. For the i7 system, instead of measuring total system power consumption, we only measured the difference in power consumption between idle and load states, added with the idle power consumption divided by the number of cores. This was an attempt to make a fair comparison in the favor of the i7 system, regarding economy of scale in the datacenter.

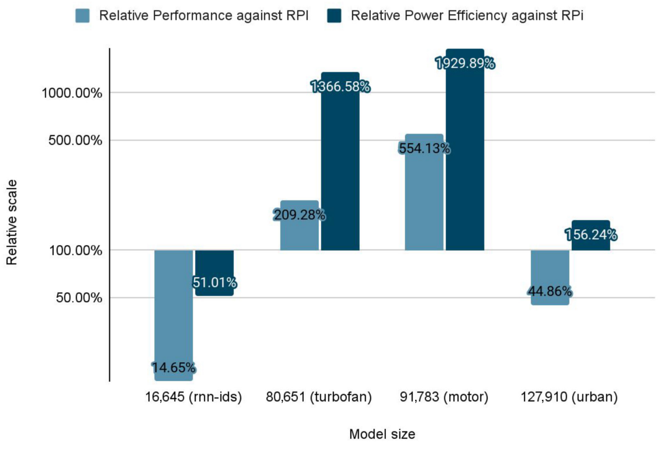


Fig. 14. Relative performance of Eciton against Raspberry Pi Zero.

Eciton's FPGA accelerator measured a power consumption around 17 mW under full load. Coupled with the Arduino Nano emulating sensor data input via USB serial, the average power consumption of the Eciton system under load measured around 290 mW. The power efficiency of Eciton is an order of magnitude better than other high performance and low power options. The systems with the next-best power efficiencies were the i7 and the Raspberry Pi running floating point, but even they were orders of magnitude lower. However, we also emphasize that the Raspberry Pi system cannot be the solution to the extreme edge scenarios Eciton targets, because the power consumption of the Raspberry Pi is 1,010 mW, 3.5 times greater than Eciton, as seen in Figure 15.

We also examine the effect of model size on the performance and power efficiency of the top two evaluated embedded platforms, Eciton and the Raspberry Pi Zero, in Figure 14. While Eciton performs relatively worse compared to the Raspberry Pi Zero for the very small model, we note that in terms of GOPS Eciton performs consistently high. The relative performance difference comes from the fact that the model is now small enough that it can fit on the cache of the Raspberry Pi. However, as we mentioned previously, the Raspberry Pi has a significantly higher power budget, which may be prohibitive for wide-scale IoT deployment. Selection of system platform must consider all of these points. For the mid-size models, Eciton receives 2–5× performance improvements and 10-fold improved power efficiency. With the urban classification model, at the SPRAM storage limit, the performance lags, but the power efficiency is still 50% better.

Eciton was also able to achieve the energy limit requirements of the electric motor scenario. It took Eciton 31.3 seconds to fully process the 25 seconds of 1/4 sampled data stream as per the model parameters, at a steady power consumption of 17 mW by the FPGA. Assuming the rest of the system can be put to sleep during this time, our Eciton's FPGA accelerator finishes the job within the 0.6 Joule energy limit imposed by the power harvester. No other system configuration was able to achieve this milestone.

4.6 Analysis of Performance and Power Evaluations

From our evaluations, we make two important observations:

— **Model size is very important for edge deployment:** Even for CPU-based systems, the same model compressed to smaller size demonstrates faster performance, even considering the added overhead of quantized arithmetic.

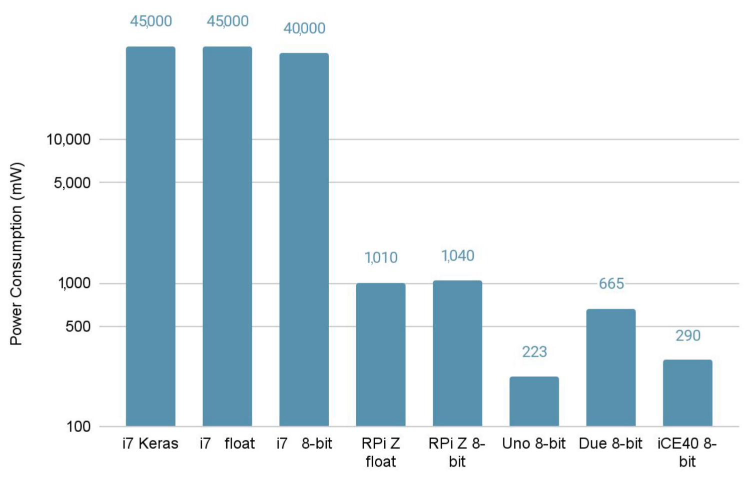
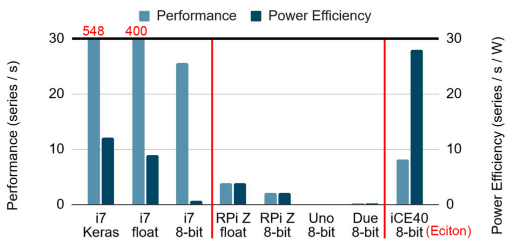
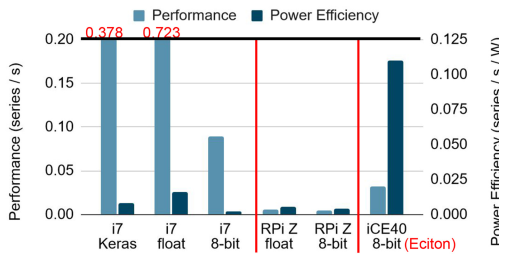


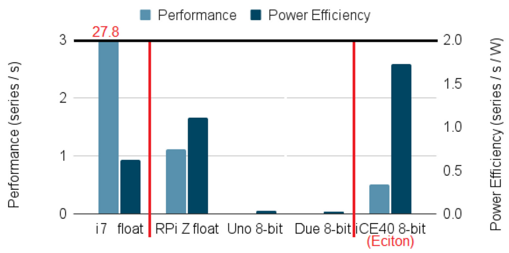
Fig. 15. Power consumption of evaluated hardware.



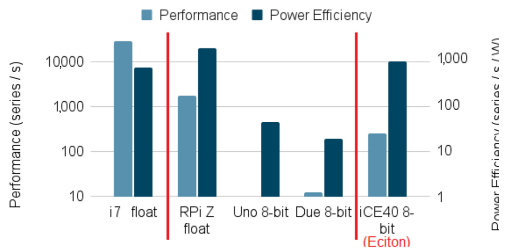
(a) Performance and power efficiency for turbofan



(b) Performance and power efficiency for motor



(c) Performance and power efficiency for urban sound



(d) Performance and power efficiency for rnn-ids

Fig. 16. Performance and power efficiency evaluations.

— **Eciton demonstrates best power characteristics among embedded systems:** While i7 or Raspberry Pi systems can sometimes achieve higher power efficiency compared to Eciton, the base power consumption requirements of such CPU-based systems may mean they cannot be deployed on extreme edge nodes, depending on the deployment scenario. Compared to low-power MCUs, Eciton demonstrates superior performance and power efficiency.

As we show in Section 4.8, to reduce the total node power consumption of an edge CPS, the total power consumption of the computation unit is an important constraint. Among the systems we evaluated, Eciton is the only one that can achieve reduction of total system power consumption without loss of performance.

Table 9. Comparisons against State-of-the-art

	Low-power			High-power				ASIC
	Eciton	[55]	[42]	[13]	[25]	[7]	[11]	[15]
Platform	**iCE40**	Spartan-7	Artix-7	Zynq-7000	Virtex-7	Zynq-7000	Arria 10	ASIC
mW	**17**	71	109	1,940	19,630	280	19,100	320
GOP/s	**0.067**	0.363	0.055	0.29	7.26	7.51	304.1	166.4
GOP/s/W	**3.9**	5.11	0.5	0.15	0.37	26.84	15.9	520

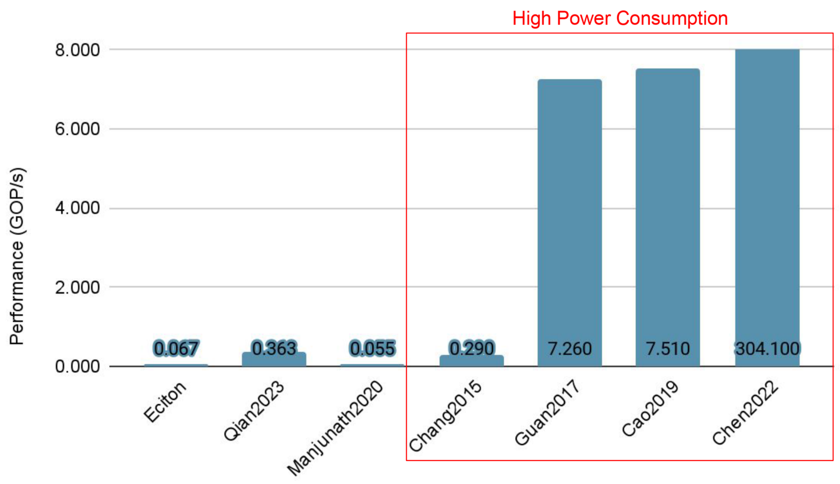


Fig. 17. Comparison of state-of-the-art performance.

4.7 Comparisons against State-of-the-art

Table 9 shows performance and power efficiency comparisons against state-of-the-art FPGA implementations of LSTMs across various hardware platforms with different scale and capabilities. We provide two recent works that postdate our prior work [14]: a framework inspired by Eciton utilizing a variety of optimization techniques for the LSTM [55] and a proposed ASIC design to accelerate LSTMs using bit-sparse quantized representation [15]. In Figure 17 and Figure 18, we look at the implementations on the various FPGAs, omitting the ASIC design proposal. Despite the resource constraints of the iCE40 UP5K platform, Eciton achieves competent performance and power efficiency compared to other low-power implementations such as those on Xilinx Artix 7 FPGAs. The ASIC systems display superior performance and power efficiency and may become the obvious choice if they become widely available. However, we note that the cited ASIC system has a higher power budget than Ecitonand may not be feasible for some scenarios if battery capacity or power harvesters cannot support it.

While some bigger chips can deliver better performance per watt, resource-constrained edge deployments may not be able to use them at all due to the power budget. Eciton has been optimized for the extremely low-power scenario, operating at such minimal power consumption that the next best state-of-the-art comparison we found is still four times as power-hungry.

We emphasize that, for the four applications we evaluated, we are able to achieve the application performance requirements even with the performance of Eciton. If Eciton can reach performance requirements, then its extremely low power budget becomes the primary strength that enables deployment in the most extreme scenarios.

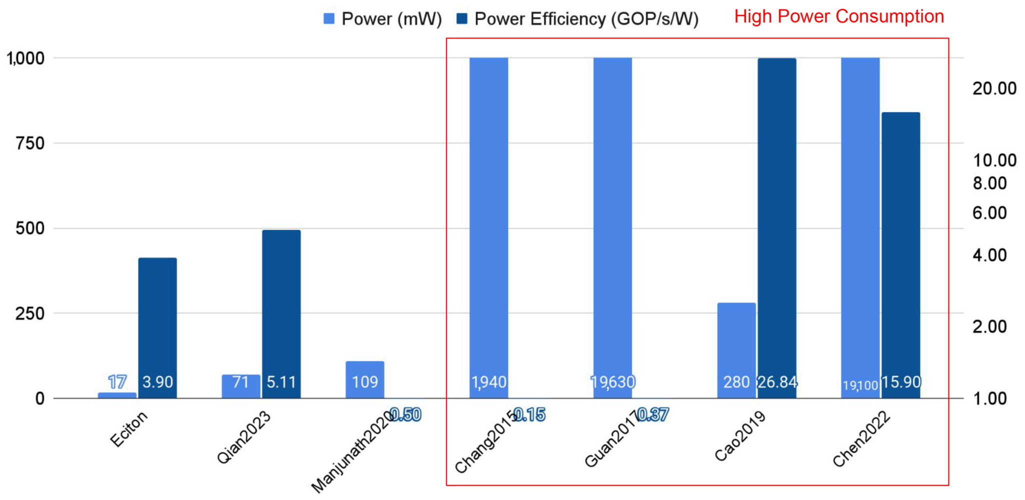


Fig. 18. Comparison of state-of-the-art power & power efficiency.

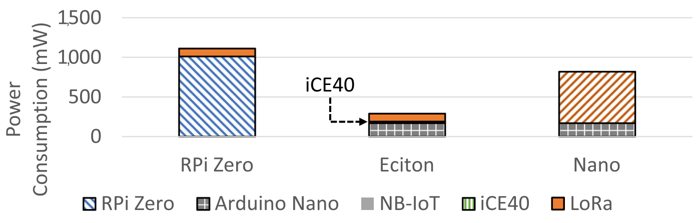


Fig. 19. Total system power consumption of nodes under load.

Eciton's selected commodity hardware platform is also cost-effective, costing approximately $5 despite its small resource availability, while other state-of-the-art FPGA implementations operate on platforms with more resources that cost tens to hundreds times more to purchase.

4.8 Total System Power with Networking

Figure 19 shows the sustained, under-load total system power consumption breakdown, including network transmission, for the top two most power-efficient embedded systems evaluated: the Raspberry Pi Zero and Eciton, as well as an Arduino Nano system transmitting all data collected with no edge computing. The Raspberry Pi and Eciton systems are configured with a low-power LoRa module, while the Arduino Nano is configured with a faster NB-IoT module due to the higher data rate required without edge filtering. For example, the electrical motor example collecting data from 3+ sensors at 2 KHz already exceeds typical LoRa bandwidth, whereas just sending a flag for every shutdown event is very low bandwidth. The power consumption numbers and performance of NB-IoT and LoRaWAN was taken from existing work [1, 12, 39].

We can see that the power consumption of Eciton is significantly lower than other systems despite the FPGA addition. Furthermore, the power consumption is low enough that it can be sustainably powered by many proposed non-intrusive power-harvesting units [2, 43, 48], meaning it can continuously operate indefinitely by harvesting power from the ambient environment.

5 CONCLUSION

We have presented the design and evaluation of Eciton, a low-power accelerator for real-time recurrent neural network inference at the edge, which uses post-training quantization and FPGA acceleration for high performance and power efficiency. We evaluated Eciton on a wide range of recurrent neural network models with various characteristics, all of which were from real-world CPS/IoT deployment scenarios. We demonstrate that, for all models tested, Eciton can efficiently handle the model sizes, maintain high accuracy, while demonstrating real-time inference performance, at competitive power efficiency compared even to cloud software with economy of scale. As a result, Eciton can reduce the total power consumption of the edge CPS/IoT node by reducing the wireless data transmission requirements at an almost negligible addition of FPGA power.

Furthermore, the wide range of models we tested demonstrates two important points: First, Eciton is capable of handling many problems of real-world scale while meeting the performance requirements. Second, once it satisfies the application performance requirements, the extreme low power of Eciton makes it uniquely able to satisfy extreme deployment scenarios using small batteries or power harvesters, where more powerful implementations with 100+ mW power budget cannot operate.

The reduced power and networking requirements enabled by Eciton will allow wider CPS/IoT deployments coupled with low-power wide-area networking and power-harvesting technologies. We also plan to explore many other CPS/IoT domains where such a platform can be beneficial, including those using convolution neural networks and other, more computationally intensive paradigms.

REFERENCES

[1] Ferran Adelantado, Xavier Vilajosana, Pere Tuset-Peiro, Borja Martinez, Joan Melia-Segui, and Thomas Watteyne. 2017. Understanding the limits of LoRaWAN. *IEEE Commun. Mag.* 55, 9 (2017), 34–40.

[2] Afghan Syeda Adila, Almusawi Husam, and Géza Husi. 2018. Towards the self-powered internet of things (IoT) by energy harvesting: Trends and technologies for green IoT. In *2nd International Symposium on Small-scale Intelligent Manufacturing Systems (SIMS'18)*. IEEE, 1–5.

[3] Md Zahangir Alom, Adam T. Moody, Naoya Maruyama, Brian C. Van Essen, and Tarek M. Taha. 2018. Effective quantization approaches for recurrent neural networks. In *International Joint Conference on Neural Networks (IJCNN'18)*. IEEE, 1–8.

[4] Arduino. 2022. Arduino Nano. Retrieved from https://store.arduino.cc/usa/arduino-nano

[5] Olgun Aydin and Seren Guldamlasioglu. 2017. Using LSTM networks to predict engine condition on large scale data processing framework. In *4th International Conference on Electrical and Electronic Engineering (ICEEE'17)*. IEEE, 281–285.

[6] Michael Baddeley, Reza Nejabati, George Oikonomou, Mahesh Sooriyabandara, and Dimitra Simeonidou. 2018. Evolving SDN for low-power IoT networks. In *4th IEEE Conference on Network Softwarization and Workshops (NetSoft'18)*. IEEE, 71–79.

[7] Erfan Bank-Tavakoli, Seyed Abolfazl Ghasemzadeh, Mehdi Kamal, Ali Afzali-Kusha, and Massoud Pedram. 2019. Polar: A pipelined/overlapped FPGA-based LSTM accelerator. *IEEE Trans. Very Large Scale Integ. Syst.* 28, 3 (2019), 838–842.

[8] Christopher M. Bishop and Nasser M. Nasrabadi. 2006. *Pattern Recognition and Machine Learning*. Vol. 4. Springer.

[9] Andrej Karpathy. 2015. The unreasonable effectiveness of recurrent neural networks. Retrieved from http://karpathy.github.io/2015/05/21/rnn-effectiveness

[10] Dario Bruneo and Fabrizio De Vita. 2019. On the use of LSTM networks for predictive maintenance in smart industries. In *IEEE International Conference on Smart Computing (SMARTCOMP'19)*. IEEE, 241–248.

[11] Shijie Cao, Chen Zhang, Zhuliang Yao, Wencong Xiao, Lanshun Nie, Dechen Zhan, Yunxin Liu, Ming Wu, and Lintao Zhang. 2019. Efficient and effective sparse LSTM on FPGA with bank-balanced sparsity. In *ACM/SIGDA International Symposium on Field-Programmable Gate Arrays*. 63–72.

[12] Lluís Casals, Bernat Mir, Rafael Vidal, and Carles Gomez. 2017. Modeling the energy performance of LoRaWAN. *Sensors* 17, 10 (2017), 2364.

[13] Andre Xian Ming Chang, Berin Martini, and Eugenio Culurciello. 2015. Recurrent neural networks hardware implementation on FPGA. *arXiv preprint arXiv:1511.05552* (2015).

[14] Jeffrey Chen, Sehwan Hong, Warrick He, Jinyeong Moon, and Sang-Woo Jun. 2021. Eciton: Very low-power LSTM neural network accelerator for predictive maintenance at the edge. In *31st International Conference on Field-Programmable Logic and Applications (FPL'21)*. IEEE, 1–8.

[15] Zhe Chen, Hugh T. Blair, and Jason Cong. 2022. Energy-efficient LSTM inference accelerator for real-time causal prediction. *ACM Trans. Des. Autom. Electron. Syst.* 27, 5 (2022), 1–19.
[16] Ram Cherukuri. 2019. What is INT8 quantization and why is it popular for deep neural networks? Retrieved from https://www.mathworks.com/company/newsletters/articles/what-is-int8-quantization-and-why-is-it-popular-for-deep-neural-networks.html
[17] Antonio De La Piedra, An Braeken, and Abdellah Touhafi. 2012. Sensor systems based on FPGAs and their applications: A survey. *Sensors* 12, 9 (2012), 12235–12264.
[18] Lei Deng, Guoqi Li, Song Han, Luping Shi, and Yuan Xie. 2020. Model compression and hardware acceleration for neural networks: A comprehensive survey. *Proc. IEEE* 108, 4 (2020), 485–532.
[19] Google Developers. 2020. Multi-class Neural Networks: Softmax. Retrieved from https://developers.google.com/machine-learning/crash-course/multi-class-neural-networks/softmax
[20] Chang Gao, Antonio Rios-Navarro, Xi Chen, Shih-Chii Liu, and Tobi Delbruck. 2020. EdgeDRNN: Recurrent neural network accelerator for edge inference. *IEEE J. Emerg. Select. Topics Circ. Syst.* 10, 4 (2020), 419–432.
[21] Elena I. Gaura, James Brusey, Michael Allen, Ross Wilkins, Dan Goldsmith, and Ramona Rednic. 2013. Edge mining the internet of things. *IEEE Sensors J.* 13, 10 (2013), 3816–3825.
[22] Felix A. Gers, Jürgen Schmidhuber, and Fred Cummins. 2000. Learning to forget: Continual prediction with LSTM. *Neural Computat.* 12, 10 (2000), 2451–2471.
[23] Tiago Gomes, Sandro Pinto, Adriano Tavares, and Jorge Cabral. 2015. Towards an FPGA-based edge device for the Internet of Things. In *IEEE 20th Conference on Emerging Technologies & Factory Automation (ETFA'15)*. IEEE, 1–4.
[24] Sotirios K. Goudos, Panagiotis I. Dallas, Stella Chatziefthymiou, and Sofoklis Kyriazakos. 2017. A survey of IoT key enabling and future technologies: 5G, mobile IoT, sematic web and applications. *Wirel. Person. Commun.* 97, 2 (2017), 1645–1675.
[25] Yijin Guan, Zhihang Yuan, Guangyu Sun, and Jason Cong. 2017. FPGA-based accelerator for long short-term memory recurrent neural networks. In *22nd Asia and South Pacific Design Automation Conference (ASP-DAC'17)*. IEEE, 629–634.
[26] Song Han, Junlong Kang, Huizi Mao, Yiming Hu, Xin Li, Yubin Li, Dongliang Xie, Hong Luo, Song Yao, Yu Wang, et al. 2017. ESE: Efficient speech recognition engine with sparse LSTM on FPGA. In *ACM/SIGDA International Symposium on Field-Programmable Gate Arrays*. 75–84.
[27] 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).
[28] Cong Hao, Xiaofan Zhang, Yuhong Li, Sitao Huang, Jinjun Xiong, Kyle Rupnow, Wen-mei Hwu, and Deming Chen. 2019. FPGA/DNN co-design: An efficient design methodology for IoT intelligence on the edge. In *56th ACM/IEEE Design Automation Conference (DAC'19)*. IEEE, 1–6.
[29] Hashem M. Hashemian. 2010. State-of-the-art predictive maintenance techniques. *IEEE Trans. Instrument. Measur.* 60, 1 (2010), 226–236.
[30] Soumil Heble, Ajay Kumar, K. V. V. Durga Prasad, Soumya Samirana, Pachamuthu Rajalakshmi, and Uday B. Desai. 2018. A low power IoT network for smart agriculture. In *IEEE 4th World Forum on Internet of Things (WF-IoT'18)*. IEEE, 609–614.
[31] Benoit Jacob, Skirmantas Kligys, Bo Chen, Menglong Zhu, Matthew Tang, Andrew Howard, Hartwig Adam, and Dmitry Kalenichenko. 2018. Quantization and training of neural networks for efficient integer-arithmetic-only inference. In *IEEE Conference on Computer Vision and Pattern Recognition*. 2704–2713.
[32] Shuang Jiang, Dong He, Chenxi Yang, Chenren Xu, Guojie Luo, Yang Chen, Yunlu Liu, and Jiangwei Jiang. 2018. Accelerating mobile applications at the network edge with software-programmable FPGAs. In *IEEE Conference on Computer Communications*. IEEE, 55–62.
[33] Deokwoo Jung, Zhenjie Zhang, and Marianne Winslett. 2017. Vibration analysis for IoT enabled predictive maintenance. In *IEEE 33rd International Conference on Data Engineering (ICDE'17)*. IEEE, 1271–1282.
[34] Seongyoung Kang, Jinyeong Moon, and Sang-Woo Jun. 2020. FPGA-accelerated time series mining on low-power IoT devices. In *IEEE 31st International Conference on Application-specific Systems, Architectures and Processors (ASAP'20)*. IEEE, 33–36.
[35] Raghuraman Krishnamoorthi. 2018. Quantizing deep convolutional networks for efficient inference: A whitepaper. *arXiv preprint arXiv:1806.08342* (2018).
[36] P. Latha and M. A. Bhagyaveni. 2010. Reconfigurable FPGA based architecture for surveillance systems in WSN. In *International Conference on Wireless Communication and Sensor Computing (ICWCSC'10)*. IEEE, 1–6.
[37] Lattice. 2022. iCE40 LP/HX/LM. Retrieved from http://www.latticesemi.com/iCE40
[38] Lattice Semiconductor Corporation. 2023. Lattice Diamond. Retrieved from https://www.latticesemi.com/latticediamond
[39] Mads Lauridsen, Rasmus Krigslund, Marek Rohr, and Germán Madueno. 2018. An empirical NB-IoT power consumption model for battery lifetime estimation. In *IEEE 87th Vehicular Technology Conference (VTC Spring'18)*. IEEE, 1–5.

[40] Iurii Lezhenin, Natalia Bogach, and Evgeny Pyshkin. 2019. Urban sound classification using long short-term memory neural network. In *Federated Conference on Computer Science and Information Systems (FedCSIS'19).* IEEE, 57–60.

[41] Ping Liu, Jin Wang, Arun Kumar Sangaiah, Yang Xie, and Xinchun Yin. 2019. Analysis and prediction of water quality using LSTM deep neural networks in IoT environment. *Sustainability* 11, 7 (2019), 2058.

[42] Nitheesh Kumar Manjunath, Hirenkumar Paneliya, Morteza Hosseini, W. David Hairston, Tinoosh Mohsenin, et al. 2020. A low-power LSTM processor for multi-channel brain EEG artifact detection. In *21st International Symposium on Quality Electronic Design (ISQED'20).* IEEE, 105–110.

[43] Philipp Mayer, Michele Magno, and Luca Benini. 2020. Smart power unit-mW-to-nW power management and control for self-sustainable IoT devices. *IEEE Trans. Power Electron.* (2020).

[44] Arnab Neelim Mazumder, Hasib-Al Rashid, and Tinoosh Mohsenin. 2020. An energy-efficient low power LSTM processor for human activity monitoring. In *33rd International System-on-Chip Conference (SOCC'20).* IEEE.

[45] Manav Mehra, Sameer Saxena, Suresh Sankaranarayanan, Rijo Jackson Tom, and M. Veeramanikandan. 2018. IoT based hydroponics system using deep neural networks. *Comput. Electron. Agric.* 155 (2018), 473–486.

[46] Kais Mekki, Eddy Bajic, Frederic Chaxel, and Fernand Meyer. 2018. Overview of cellular LPWAN technologies for IoT deployment: Sigfox, LoRaWAN, and NB-IoT. In *IEEE International Conference on Pervasive Computing and Communications Workshops (PerCom Workshops'18).* IEEE, 197–202.

[47] Microsemi. 2022. IGLOO. Retrieved from https://www.microsemi.com/product-directory/fpgas/1689-igloo

[48] Jinyeong Moon, Peter Lindahl, John Donnal, Steven Leeb, Lt. Ryan Zachar, Lt. William Cotta, and Christopher Schantz. 2016. A nonintrusive magnetically self-powered vibration sensor for automated condition monitoring of electromechanical machines. In *IEEE AUTOTESTCON.* IEEE, 1–7.

[49] D. Mourtzis, E. Vlachou, and N. J. P. C. Milas. 2016. Industrial big data as a result of IoT adoption in manufacturing. *Procedia Cirp* 55 (2016), 290–295.

[50] Karan Nair, Janhavi Kulkarni, Mansi Warde, Zalak Dave, Vedashree Rawalgaonkar, Ganesh Gore, and Jonathan Joshi. 2015. Optimizing power consumption in IoT based wireless sensor networks using Bluetooth low energy. In *International Conference on Green Computing and Internet of Things (ICGCIoT'15).* IEEE, 589–593.

[51] Rishiyur Nikhil. 2004. Bluespec system verilog: Efficient, correct RTL from high level specifications. In *2nd ACM and IEEE International Conference on Formal Methods and Models for Co-Design (MEMOCODE'04).* IEEE, 69–70.

[52] Keith E. Nolan, Wael Guibene, and Mark Y. Kelly. 2016. An evaluation of low power wide area network technologies for the Internet of Things. In *International Wireless Communications and Mobile Computing Conference (IWCMC'16).* IEEE, 439–444.

[53] University of New Brunswick. 2022. NSL-KDD dataset. Retrieved from https://www.unb.ca/cic/datasets/nsl.html

[54] Vilabha S. Patil, Yashwant B. Mane, and Shraddha Deshpande. 2019. FPGA based power saving technique for sensor node in wireless sensor network (WSN). In *Computational Intelligence in Sensor Networks.* Springer, 385–404.

[55] Chao Qian, Tianheng Ling, and Gregor Schiele. 2023. Enhancing energy-efficiency by solving the throughput bottleneck of LSTM cells for embedded FPGAs. In *International Workshops of ECML PKDD: Machine Learning and Principles and Practice of Knowledge Discovery in Databases.* Springer, 594–605.

[56] Emilio Rapuano, Tommaso Pacini, Luca Fanucci, et al. 2022. A post-training quantization method for the design of fixed-point-based FPGA/ASIC hardware accelerators for LSTM/GRU algorithms. *Computat. Intell. Neurosci.* 2022 (2022).

[57] Nesma M. Rezk, Madhura Purnaprajna, Tomas Nordström, and Zain Ul-Abdin. 2020. Recurrent neural networks: An embedded computing perspective. *IEEE Access* 8 (2020), 57967–57996.

[58] Marwen Roukhami, Mihai Teodor Lazarescu, Francesco Gregoretti, Younes Lahbib, and Abdelkader Mami. 2019. Very low power neural network FPGA accelerators for tag-less remote person identification using capacitive sensors. *IEEE Access* 7 (2019), 102217–102231.

[59] Justin Salamon, Christopher Jacoby, and Juan Pablo Bello. 2014. A dataset and taxonomy for urban sound research. In *22nd ACM International Conference on Multimedia.* 1041–1044.

[60] Hojjat Salehinejad, Sharan Sankar, Joseph Barfett, Errol Colak, and Shahrokh Valaee. 2017. Recent advances in recurrent neural networks. *arXiv preprint arXiv:1801.01078* (2017).

[61] Suresh Sankaranarayanan, Joel J. P. C. Rodrigues, Vijayan Sugumaran, Sergei Kozlov, et al. 2020. Data flow and distributed deep neural network based low latency IoT-edge computation model for big data environment. *Eng. Applic. Artif. Intell.* 94 (2020), 103785.

[62] Abhinav Saxena, Kai Goebel, Don Simon, and Neil Eklund. 2008. Damage propagation modeling for aircraft engine run-to-failure simulation. In *International Conference on Prognostics and Health Management.* IEEE, 1–9.

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

[64] Sule Selcuk. 2017. Predictive maintenance, its implementation and latest trends. *Proc. Instit. Mechan. Eng., Part B: J. Eng. Manuf.* 231, 9 (2017), 1670–1679.

[65] Lattice Semiconductor. 2022. Lattice sensAI Stack. Retrieved from https://www.latticesemi.com/sensAI

[66] Björn Stelte. 2010. Toward development of high secure sensor network nodes using an FPGA-based architecture. In *6th International Wireless Communications and Mobile Computing Conference*. 539–543.

[67] Xiao Sun, Jungwook Choi, Chia-Yu Chen, Naigang Wang, Swagath Venkataramani, Vijayalakshmi Viji Srinivasan, Xiaodong Cui, Wei Zhang, and Kailash Gopalakrishnan. 2019. Hybrid 8-bit floating point (HFP8) training and inference for deep neural networks. *Adv. Neural Inf. Process. Syst.* 32 (2019), 4900–4909.

[68] Foivos Tsimpourlas, Lazaros Papadopoulos, Anastasios Bartsokas, and Dimitrios Soudris. 2018. A design space exploration framework for convolutional neural networks implemented on edge devices. *IEEE Trans. Comput.-Aid. Des. Integ. Circ. Syst.* 37, 11 (2018), 2212–2221.

[69] Sapna Tyagi, Amit Agarwal, and Piyush Maheshwari. 2016. A conceptual framework for IoT-based healthcare system using cloud computing. In *6th International Conference-Cloud System and Big Data Engineering (Confluence'16)*. IEEE, 503–507.

[70] Arijit Ukil, Soma Bandyoapdhyay, Chetanya Puri, and Arpan Pal. 2016. IoT healthcare analytics: The importance of anomaly detection. In *IEEE 30th International Conference on Advanced Information Networking and Applications (AINA'16)*. IEEE, 994–997.

[71] Juan Valverde, Andres Otero, Miguel Lopez, Jorge Portilla, Eduardo De la Torre, and Teresa Riesgo. 2012. Using SRAM based FPGAs for power-aware high performance wireless sensor networks. *Sensors* 12, 3 (2012), 2667–2692.

[72] Naigang Wang, Jungwook Choi, Daniel Brand, Chia-Yu Chen, and Kailash Gopalakrishnan. 2018. Training deep neural networks with 8-bit floating point numbers. *Adv. Neural Inf. Process. Syst.* 31 (2018).

[73] Qi Wang, Siqi Bu, and Zhengyou He. 2020. Achieving predictive and proactive maintenance for high-speed railway power equipment with LSTM-RNN. *IEEE Trans. Industr. Inform.* 16, 10 (2020), 6509–6517.

[74] Qianlong Wang, Yifan Guo, Lixing Yu, and Pan Li. 2017. Earthquake prediction based on spatio-temporal data mining: An LSTM network approach. *IEEE Trans. Emerg. Topics Comput.* (2017).

[75] Daqian Wei, Bo Wang, Gang Lin, Dichen Liu, Zhaoyang Dong, Hesen Liu, and Yilu Liu. 2017. Research on unstructured text data mining and fault classification based on RNN-LSTM with malfunction inspection report. *Energies* 10, 3 (2017), 406.

[76] Clifford Wolf and Mathias Lasser. 2022. Project IceStorm. Retrieved from http://www.clifford.at/icestorm/

[77] Shuang Wu, Guoqi Li, Feng Chen, and Luping Shi. 2018. Training and inference with integers in deep neural networks. *CoRR* abs/1802.04680 (2018).

[78] Yoji Yamato, Yoshifumi Fukumoto, and Hiroki Kumazaki. 2017. Predictive maintenance platform with sound stream analysis in edges. *J. Inf. Process.* 25 (2017), 317–320.

[79] Yukuan Yang, Lei Deng, Shuang Wu, Tianyi Yan, Yuan Xie, and Guoqi Li. 2020. Training high-performance and large-scale deep neural networks with full 8-bit integers. *Neural Netw.* 125 (2020), 70–82.

[80] Chuanlong Yin, Yuefei Zhu, Jinlong Fei, and Xinzheng He. 2017. A deep learning approach for intrusion detection using recurrent neural networks. *IEEE Access* 5 (2017), 21954–21961.

[81] Xiaofan Zhang, Anand Ramachandran, Chuanhao Zhuge, Di He, Wei Zuo, Zuofu Cheng, Kyle Rupnow, and Deming Chen. 2017. Machine learning on FPGAs to face the IoT revolution. In *IEEE/ACM International Conference on Computer-Aided Design (ICCAD'17)*. IEEE, 894–901.

[82] Kai Zhen, Hieu Duy Nguyen, Raviteja Chinta, Nathan Susanj, Athanasios Mouchtaris, Tariq Afzal, and Ariya Rastrow. 2022. Sub-8-bit quantization aware training for 8-bit neural network accelerator with on device speech recognition. In *Interspeech Conference*. Retrieved from https://www.amazon.science/publications/sub-8-bit-quantization-aware-training-for-8-bit-neural-network-accelerator-with-on-device-speech-recognition

[83] Chao Hu Zhiyong, Liu Yingzi Pan, Zhenxing Zeng, and Max Q.-H. Meng. 2009. A novel FPGA-based wireless vision sensor node. In *IEEE International Conference on Automation and Logistics*. IEEE, 841–846.

[84] Xiaokang Zhou, Wei Liang, Shohei Shimizu, Jianhua Ma, and Qun Jin. 2020. Siamese neural network based few-shot learning for anomaly detection in industrial cyber-physical systems. *IEEE Trans. Industr. Inform.* 17, 8 (2020), 5790–5798.

Received 10 June 2022; revised 6 July 2023; accepted 27 September 2023