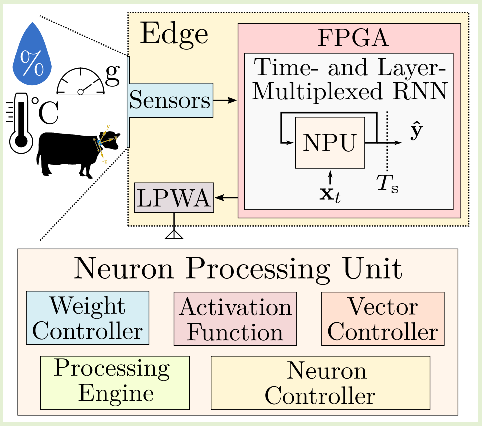
An Integer-Only Resource-Minimized RNN on FPGA for Low-Frequency Sensors in Edge-AI

Jim Bartels, *Graduate Student Member, IEEE*, Aran Hagihara, *Graduate Student Member, IEEE*, Ludovico Minati, *Senior Member, IEEE*, Korkut Kaan Tokgoz, *Member, IEEE*, and Hiroyuki Ito, *Member, IEEE*

***Abstract*—The growth of Artificial Intelligence (AI) and the Internet of Things (IoT) sensors has given rise to a synergistic paradigm known as AIoT, wherein AI functions as the decision-maker and sensors collect information. However, a substantial proportion of AIoT rely on cloud-based AI, which process wirelessly transmitted raw data, increasing power consumption and reducing battery life at sensor nodes. Edge-AI has emerged as a promising alternative, implementing AI directly on sensor nodes, eliminating the need of raw data transmission. Despite its potential, there is a scarcity of hardware architectures optimized for resource-constrained platforms, such as field programmable gate arrays (FPGAs), particularly for low-frequency sensors. This work presents a shared-scale integer-only recurrent neural network (RNN) implemented on a Lattice ICE40UP5K FPGA using a resource-minimized time and layer-multiplexed (TLM) hardware architecture. This architecture adopts real-time processing, setting clock frequency to complete a single RNN timestep preceding the next sensor sample, reducing power consumption significantly. Measurements on this FPGA implementing our proposed architecture applied to a pretrained RNN on cow behavior show a power consumption of 360 μW at a clock frequency of 146 kHz and negligible accuracy loss at 8-bit bitwidth. This finding suggests that our methods lead to the most accurate implementation of animal behavior estimation with a power consumption below 500 μW on an FPGA. The implementation in Systemverilog and Python code is publicly available, enabling adaptation of the RNN for various tasks involving low-frequency sensors on resource-constrained FPGAs, thereby contributing to the further advancement and democratization of Edge-AI solutions.**



***Index Terms*—Artificial intelligence (AI), edge-AI, field programmable gate array (FPGA), Internet of Things (IoT), machine learning, precision livestock farming (PLF), quantization, recurrent neural network (RNN).**

I. Introduction

THE advent of Artificial Intelligence (AI) has enabled increasingly complex tasks, such as 3-D protein structure estimation and plasma control in nuclear fusion reactors [1], [2]. Adjoining this development, the Internet of Things (IoT), utilizing microcontrollers and field programmable gate arrays (FPGAs), has become a vital part of applications in livestock, wild-life monitoring, robotics, and manufacturing [3]. By 2025, it is estimated that 100 billion IoT devices will generate an economic impact of 11 trillion USD [4]. Similarly, the global GDP is expected to increase by 14% until 2030 from use of AI across sectors of industry [5]. This surge in IoT and AI has led to a synergy, dubbed AIoT, where IoT sensor nodes serve as information gatherers and AI models as the decision maker, drawing similarities with sense organs, the nervous system, and the brain. Cloud computing is integral to a considerable share of AIoT implementations [6], providing vast compute resources for implementing AI classifiers that process wirelessly transmitted sensor data that include images, sound, temperature, acceleration, or humidity [7]. However, this transmission, which may involve the utilization of low-power wide area (LPWA) networks, incurs considerable power consumption that increases with the number of bytes transmitted and the frequency of transmission intervals [8].

Manuscript received 5 April 2023; revised 1 June 2023; accepted 6 June 2023. Date of publication 23 June 2023; date of current version 1 August 2023. This work was supported in part by the Japan Science and Technology Agency, Support for Pioneering Researcher Initated by the Next Generation Program (JST-SPRING) under Grant JPMJSP2106. The associate editor coordinating the review of this article and approving it for publication was Dr. Avik Santra. *(Corresponding author: Jim Bartels.)*

Jim Bartels, Aran Hagihara, Ludovico Minati, and Hiroyuki Ito are with the Nano Sensing Unit, Tokyo Institute of Technology, Yokohama 226-8503, Japan (e-mail: bartels.j.aa@m.titech.ac.jp; paper@lsi.pi.titech.ac.jp).

Korkut Kaan Tokgoz was with the Research Institute for the Earth Inclusive Sensing (EISESiV), Tokyo Institute of Technology, Tokyo 152-8552, Japan. He is now with the Faculty of Engineering and Natural Sciences, Sabancı University, 34956 İstanbul, Türkiye.

Code and guide available at: https://zenodo.org/record/7800728

Digital Object Identifier 10.1109/JSEN.2023.3286580

Edge-AI proposes to implement AI models on the sensor node itself, leading to a key advantage in power consumption by eliminating the necessity of raw sensor data transmission [9]. Consequently, Edge-AI substantially decreases the size and interval of transmitted data, potentially leading to a 1000-fold increase in battery life when utilizing LPWA technology [10]. Furthermore, Edge-AI offers benefits when deployed near end-users, as it enables processing and transmission of class labels in real time [11]. Despite these advantages, several challenges hinder its widespread adoption, primarily arising from the discrepancy between software-based models and hardware resources required for implementation [12]. Moreover, the majority of implementations are proprietary and rely on hardware with high design costs, such as application specific integrated circuits (ASICs), restricting public access from adopting novel hardware architectures for custom deployments on widely available FPGAs. FPGAs have emerged as an alternative to ASICs due to their programmable logic and customizable building blocks at significantly lower economic costs [13]. In addition, FPGAs have been proposed as a means to alleviate computing constraints at the edge and accelerate processing, resulting in a significant speedup compared to CPU-based platforms such as microcontrollers [14]. However, the emphasis on maximizing processing speed offers limited advantages for sensor node implementations since system latency is inherently constrained by sensor frequency.

The constraint of sensor frequency can be effectively leveraged by recurrent neural networks (RNNs), as these networks process data on an input-by-input basis. In other words, unlike fully connected and convolutional neural networks that require the entire sensor data space during inference, RNNs repeat operations for each new sample over a set number of timesteps. Taking into account also their ability to recognize spatiotemporal patterns within generative processes [15], RNNs seem to be particularly relevant for sensor applications. Other than sensors, RNNs have been extensively used in machine translation, video captioning, and actuator control, where latency is crucial for user-friendliness [16], [17], [18], [19]. In these applications, the primary performance metric is system latency, which is minimized through techniques such as parallelization, pipelining, quantization, and resource scheduling optimization [20], [21]. However, to the best of our knowledge, no RNN hardware architecture has been proposed that leverages the processing scheme of RNNs to minimize resources, and implements a real-time processing approach with regard to sensor inputs.

In this work, we present a resource-minimized RNN hardware architecture based on an extended quantization scheme for integer-only RNNs that reduces the number of fixed-point multiplications to one third. This architecture, referred to as time-and layer-multiplexed (TLM) RNN, adopts a real-time approach, enabling its implementation on heavily-constrained, low-cost FPGAs, such as the Lattice ICE40UP5K (Lattice Semiconductor Inc., Hillsboro OR). While the Lattice ICE40UP5K FPGA serves as a primary example in this work, other low-resource FPGAs, including the Gowin LittleBee (Gowin Semiconductor Corp., Guangzhou Guangdong, CN) and Efinix Trion (Efinix, Inc., Cupertino CA), are equally suitable for implementing the proposed architecture. We employ the ICE40UP5K FPGA to demonstrate the proposed architecture on three models, including a pretrained model on a previously-proven IoT application, cow behavior estimation [22], [23], and asses them using a measurement setup. During our evaluation, the internal clock frequency is adjusted within the range of 145 kHz to 12 MHz and a corresponding substantial increase in power consumption is observed. Based on this observation, we derive an expression that establishes a minimum clock frequency, ensuring that the RNN completes timestep processing just before the arrival of the next sensor input. Applying this expression to the implementation of the cow behavior model (model c) that receives input from a 25 Hz accelerometer, we achieve a power consumption of 360 μW at a clock frequency of 146 kHz. This effectively demonstrates the applicability of our approach to precision livestock farming (PLF) with sensors, as we present, to the best of our knowledge, the first highly accurate (top-1 accuracy >95%) animal behavior estimation model operating on an FPGA with a power consumption below 500 μW.

The contributions of this work are.

1) An extension to the post-training quantization scheme of integer-only RNNs with quantization scale sharing, reducing the number of fixed-point multiplications to one third. (Section II)
2) The introduction of a TLM RNN hardware architecture using a real-time approach with regard to incoming sensor data, minimizing resources, and reducing power consumption. (Section III)
3) Implementation of the proposed methods on a heavily-constrained and low-cost FPGA, including power consumption measurements. (Sections III and IV)
4) Open-sourcing of the code used for FPGA implementation and conversion from a Tensorflow 2.0 model [24], [25], [26], enabling public access.

II. Shared-Scale Integer-Only RNN

Based on an integer post-training quantization for convolutional neural networks [27], [22] and [28] proposed an approach for quantizing RNNs. We extend this quantization scheme by introducing sharing of quantization scales. In essence, this scheme is based on converting any real number into an unsigned integer

$$r = S(q - Z) \tag{1}$$

where r is a real number, q is an integer, and S and Z are the scale and zeropoint, described by

$$S = \frac{|\max(\mathbf{M})| + |\min(\mathbf{M})|}{2^{q_{\text{bits}}} - 1} \tag{2}$$

$$Z = \left\lfloor \frac{-\min(\mathbf{M})}{S} \right\rceil \tag{3}$$

where $\mathbf{M}$ is a matrix of any size and q_{bits} is the bit-width.

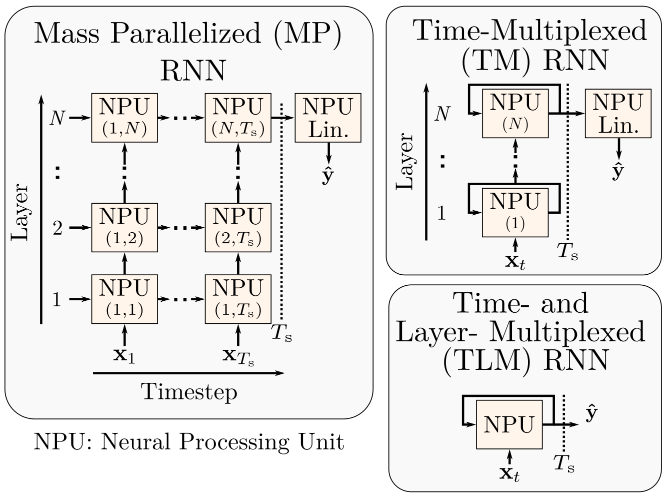


Fig. 1. MP, TM, and TLM RNN architectures. MP signifies the most rapid architecture with the minimum number of clock cycles required for inference, allocating an NPU to each layer and timestep. TLM employs a single NPU, minimizing resources while necessitating the greatest number of clock cycles for processing.

A. Methods

Let the system equation of the simple RNN be

$$\mathbf{h}_t = \tanh(\mathbf{b} + \mathbf{U}\mathbf{h}_{t-1} + \mathbf{W}\mathbf{x}_t) \quad (4)$$

where $\mathbf{b}$, $\mathbf{U}$, and $\mathbf{W}$ are the trained bias, hidden, and input weight matrices, respectively, and $\mathbf{h}_{t-1}$ and $\mathbf{x}_t$ are the hidden vector at time $t-1$ and the input vector at time t [29]. Using the methods in [22] and [28] and omitting the scale, the above matrix terms are quantized to

$$\mathbf{U}'_\mathrm{q} = (\mathbf{U} - Z_\mathrm{U})(\mathbf{h}_{t-1} - Z_\mathrm{h}) \quad (5)$$
$$\mathbf{W}'_\mathrm{q} = (\mathbf{W} - Z_\mathrm{W})(\mathbf{x}_t - Z_\mathrm{x}) \quad (6)$$
$$\mathbf{b}_\mathrm{q} = (\mathbf{b} - Z_\mathrm{b}) \quad (7)$$

where Z_i is the zeropoint and i represents the corresponding matrix. Including the above in (4) with their respective scales leads to

$$\mathbf{h}_{t,\mathrm{q}} = \tanh\left(\frac{1}{S_\mathrm{T}}\left(S_\mathrm{b}\mathbf{b}_\mathrm{q} + S_\mathrm{U}S_\mathrm{h}\mathbf{U}'_\mathrm{q} + S_\mathrm{W}S_\mathrm{x}\mathbf{W}'_\mathrm{q}\right) + Z_\mathrm{T}\right) \quad (8)$$

where S_i is the scale, i representing the corresponding matrix and tanh is the hyperbolic tangent. S_T and Z_T cast the summed up term into a lower bit representation [22], [27].

Scales are decimals, represented with floating point or fixed-point numbers. Operations using these representations require more logic to implement than integer-based operations [30]. We propose the sharing of these scales between matrices, and between input-output vectors after discovering that the resulting model error is negligible, as matrix values are bounded within a similar range post-training. Consequently, (8) reduces to

$$\mathbf{h}_{t,\mathrm{q}} = \tanh(M(\mathbf{b}_\mathrm{q} + \mathbf{U}'_\mathrm{q} + \mathbf{W}'_\mathrm{q}) + Z_\mathrm{T}) \quad (9)$$
$$M = \frac{S_\mathrm{M}S_\mathrm{v}}{S_\mathrm{T}} \quad (10)$$
$$S_\mathrm{M} = \frac{|\max(u_\mathrm{max}, w_\mathrm{max})| + |\min(u_\mathrm{min}, w_\mathrm{min})|}{2^{q_\mathrm{bits}} - 1} \quad (11)$$

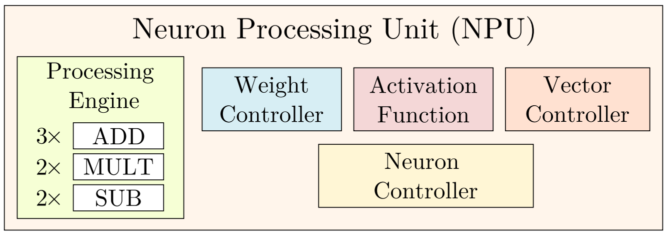


Fig. 2. NPU, containing five modules, processing engine, Activation Function and Weight, Vector, and Neuron Controllers. Expanded modules are shown in Fig. 3.

TABLE I
PARAMETERS OF MODELS USED FOR POWER MEASUREMENTS WITH INTERNAL CLOCK FREQUENCY IN SECTION III-B (a,b,c), AND COW BEHAVIOR PROOF OF CONCEPT IN SECTION IV (c)

	Model			
Parameter	Experimental		Cow Behavior	Description
	a	b	c	
N	20	10	13	Layer width
L	4	2	4	Number of layers
I	6	3	3	Number of inputs
O	8	4	4	Number of classes
q_bits	8	8	8	Bitwidth
T_s	50	50	35	Timesteps

$$S_\mathrm{v} = \max(S_\mathrm{x}, S_\mathrm{h}) \quad (12)$$
$$S_\mathrm{b} = S_\mathrm{M}S_\mathrm{v} \quad (13)$$

where $\{u, w\}_{\{\mathrm{max,min}\}}$ are the maximum and minimum matrix elements of $\mathbf{U}_\mathrm{q}$ and $\mathbf{W}_\mathrm{q}$, and M is a multiplicative decimal term, reducing non-integer multiplications to one third. This multiplication is performed with a multiply-and-shift operation

$$M_\mathrm{shift} = -\lceil\log_2(M)\rceil + (q_\mathrm{bits} - 1) \quad (14)$$
$$M_\mathrm{int} = \lfloor M \cdot 2^{M_\mathrm{shift}}\rceil \quad (15)$$
$$M = M_\mathrm{int} >> M_\mathrm{shift} \quad (16)$$

where q_bits is the bitwidth, and M_int and M_shift are integers, respectively. Here, $>>$ represents a bitwise right shift operator. The scales of $\mathbf{x}_t$ and $\mathbf{h}_t$, S_x, S_h, are shared, set based on the domain of the hyperbolic tangent, $D_\mathrm{tanh} \in [-2, 2]$. This method can be extended to other RNN types, long-short-term-memory and the gated recurrent unit [31], [32].

Table I describes the RNN model parameters of three models that are considered for FPGA implementation in subsequent Sections. Using the proposed scheme, each layer undergoes separate quantization, while input-output vector quantization across the model remains constant. Additionally, each RNN layer exhibits an equal width, N. The final RNN layer is followed by a single feed-forward layer, mapping the final hidden output vector at $t = T_\mathrm{s}$, $\mathbf{h}_{T_\mathrm{s},\mathrm{q}}$ to class prevalence, $\hat{\mathbf{y}}$, in a process referred to as many-to-one classification. Further details regarding the model structure can be found in [22].

III. RESOURCE-MINIMIZED HARWARE ARCHITECTURE

Fig. 1 shows three RNN architectures, mass parallelized (MP), time-multiplexed (TM), and TLM, employing a varying

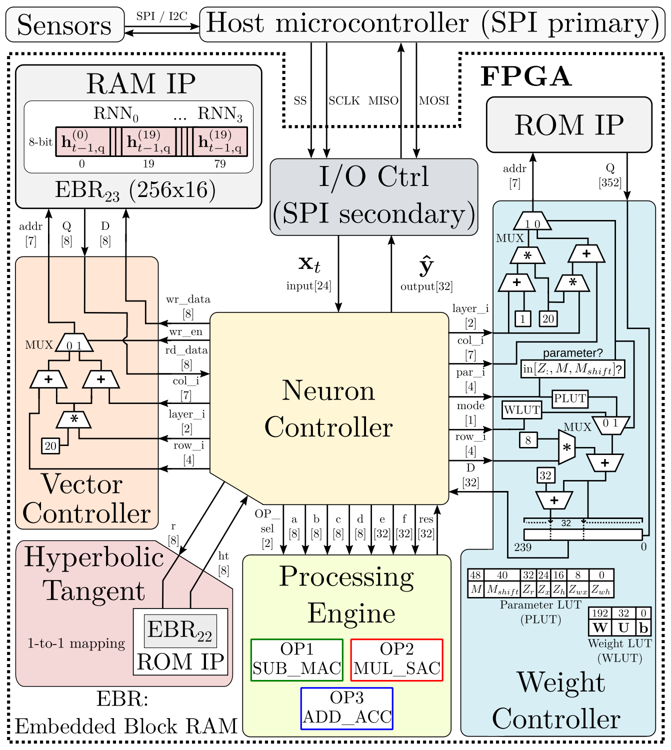


Fig. 3. Architectural overview of the NPU on FPGA at 8-bit bitwidth, implementing model a (parameter values described in Table I). The structure of the weight memory (ROM IP) and details of the processing engine are shown in Figs. 5 and 6, respectively. Clock, reset, and enable signals are omitted.

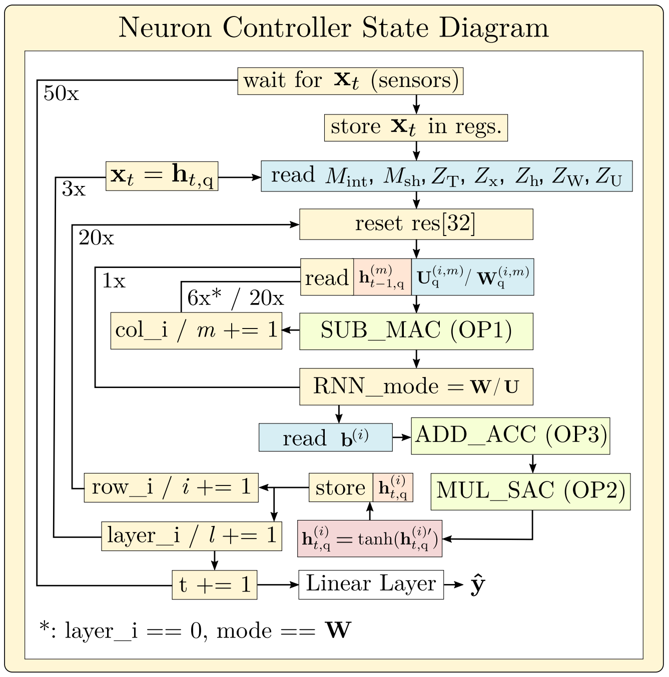

Neuron Controller State Diagram
wait for $\mathbf{x}_t$ (sensors)
store $\mathbf{x}_t$ in regs.
$\mathbf{x}_t = \mathbf{h}_{t,q}$
read $M_{int}, M_{sh}, Z_T, Z_x, Z_h, Z_W, Z_U$
reset res[32]
read $\mathbf{h}^{(m)}_{t-1,q}$ $\mathbf{U}^{(i,m)}_q$ / $\mathbf{W}^{(i,m)}_q$
col_i / m += 1
SUB_MAC (OP1)
RNN_mode = W/U
read $\mathbf{b}^{(i)}$
ADD_ACC (OP3)
MUL_SAC (OP2)
row_i / i += 1
store $\mathbf{h}^{(i)}_{t,q}$
layer_i / l += 1
$\mathbf{h}^{(i)}_{t,q} = \tanh(\mathbf{h}^{(i)\prime}_{t,q})$
t += 1
Linear Layer
$\hat{\mathbf{y}}$
50x 3x 20x 1x 6x* / 20x
*: layer_i == 0, mode == W

Fig. 4. State Diagram of the Neuron Controller for RNN inference on model a, described in Table I.

number of neuron processing units (NPUs). Fig. 2 shows this NPU, comprising five modules that are employed to process a single time step of the proposed RNN. Additionally, the NPU can implement a linear layer, mapping features from the RNN to classified labels. The MP architecture aims to maximize throughput by implementing an NPU at each time step and layer, resulting in the least required NPU cycles per inference. However, this architecture demands a significant amount of resources, rendering it unsuitable for heavily resource-constrained FPGAs. The MP architecture is better suited for large-scale FPGAs, such as the Xilinx Virtex (Xilinx, Inc., San Jose CA) or Intel Stratix series (Intel Corp., Santa Clara CA), frequently utilized in accelerators [33]. In contrast, the TM architecture offers a scaled-down approach, reducing the employed NPUs to the number of layers. Consequently, the speed becomes a function of the number of time steps. This architecture is appropriate for intermediate scenarios, where throughput remains crucial, but the considered FPGA for use contains moderate resource constraints. The focus of this work is the TLM that minimizes resource usage and allows implementation on an extremely small FPGA. This architecture necessitates a single NPU for constructing any RNN according to the previously discussed architectural setting, namely, an equal RNN layer width followed by a single linear layer. In this case, the processing speed becomes a function of the number of time steps and layers, increasing the number of clock cycles compared to MP and TM architectures. However, as will be demonstrated in Section III-B, this does not pose a challenge when input data originate from sensors with sufficiently low sampling frequency.

Within this architecture, the single NPU stores the intermediate recurrent hidden output vectors, $\mathbf{h}_{t,q}$ for the next layer and timestep in memory. We propose to implement the TLM architecture with a real-time approach, i.e., the internal clock frequency is set as a function of the architectural parameters that influence the number of clock cycles for a single timestep of the RNN to be completely processed by the NPU, i.e.,

$$f_{in} = p_{cycles} \cdot f_{sensor} \tag{17}$$

where f_{in} is the internal clock frequency, p_{cycles} is the number of clock cycles for processing a single timestep [(18)–(20)], and f_{sensor} is the sampling frequency of the sensor. This approach leads to a clock frequency on FPGA that is minimized using a clock divider under the constraint that the processing completes before the next sensor sample arrives.

A. Single NPU

The subsequent sections will present the implementation of the TLM architecture on the Lattice ICE40UP5K FPGA, encompassing intellectual property (IP) modules. This FPGA is extremely compact (smallest package measures 5.38 mm^2), featuring 5280 4-bit look up tables (LUTs) and 1-bit registers, as well as 30 Embedded Block RAMs (EBRs) consisting of 4096 bits each at a core voltage of 1.2 V. At a typical price <10 USD, this FPGA is an example of low-cost programmable hardware that is heavily constrained in terms of resources. Other similar examples suitable for implementation include the Gowin LittleBee and Effinix Trion FPGAs. The implementation incorporates five IPs: algorithmic IPs, such as Adders, Subtractors, and Multipliers, and random access

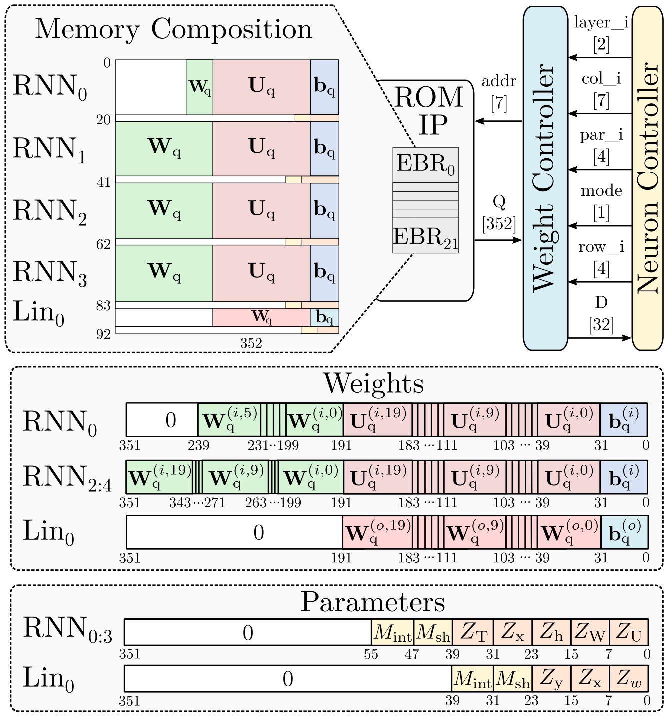


Fig. 5. Memory composition of the weight and parameters of the RNN, layer-by-layer ($RNN_{0:3}$). One word in the ROM comprises a single row of the input weight and hidden weight matrices, and the bias, leading to $(2N+4)q_{\text{bits}}$ bits per word. The final address of a layer, address $(N+1)$, contains the quantization parameters, i.e., M_{int}, M_{shift} and zeropoints Z_i [(4)].

memory (RAM) and read-only memory (ROM) IPs, which are realized using EBRs. The source code, accompanied by a comprehensive guide for adapting the implementation to a customized model on Lattice FPGAs and an overview of IPs to modify in the source code for compatibility with other vendors, can be found in [26].

Fig. 3 presents an expanded NPU featuring a system-level overview that incorporates the FPGA, sensor peripherals, and an external microcontroller (parameters of model a, Table I). In this configuration, the FPGA (secondary) communicates with a microcontroller (primary) to receive sensor data, $\mathbf{x}_t$, and transmit class predictions, $\hat{\mathbf{y}}$, utilizing the serial peripheral interface (SPI). The neuron controller, acting as the central component of this architecture, follows a predefined finite state machine (FSM) that distributes instructions and data to other modules using the described wires. Sections III-A1–III-A5 provide detailed descriptions of the key modules and their functions within the proposed architecture: the neuron controller, weight controller, vector controller, processing engine, and hyperbolic tangent (activation function).

1) Neuron Controller: Fig. 4 shows the state diagram of the neuron controller. The states involve instructing the processing engine to commence operations (OP1-3), guiding the memory and vector controller to store and load elements, and incrementing the column, row, layer index, and time, denoted as m, i, l, and t respectively. Initially, the neuron controller stores incoming sensor data, $\mathbf{x}_t$, in registers, proceeded by an instruction to read all quantization parameters M_{int}, M_{shift}, and

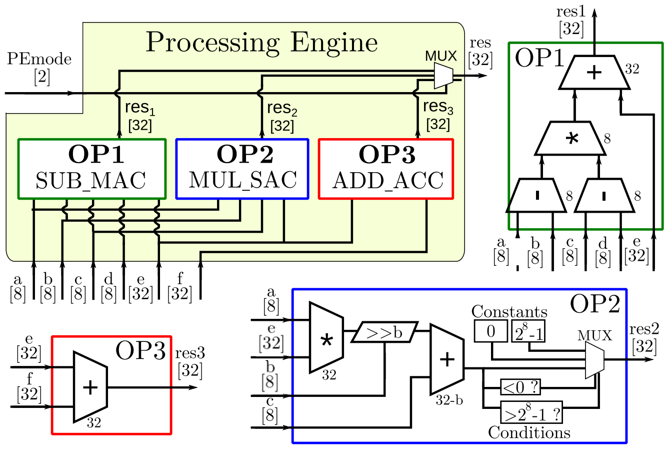


Fig. 6. Expanded diagram of the processing engine and three proposed operations at 8-bit bitwidth, SUB_MAC (OP1), MUL_SAC (OP2), and ADD_ACC (OP3).

Z. Following this, two distinct modes of operation are selected with wire "mode."

In the first mode, $\mathbf{W}_{\text{q}}^{(i)\prime}$ is processed [(6)]. The controller receives $\mathbf{W}_{\text{q}}^{(i,m)}$ from the memory and forward it along with $\mathbf{x}_t^{(m)}$ at layer $l=0$, or $\mathbf{h}_{t,\text{q}}^{(m)}$ of the previous layer $l-1$ for $l>0$ to the PE, initiating OP1. After 6 ($l=0$) or 20 ($l>0$) increments of m, the second mode of the RNN commences, similarly processing $\mathbf{U}_{\text{q}}^{(i)\prime}$ [(5)] using $\mathbf{U}_{\text{q}}^{(i,m)}$ and $\mathbf{h}_{t-1,\text{q}}^{(m)}$. Subsequently, OP3 and OP2 are initiated, yielding $\mathbf{h}_{t,\text{q}}^{(i)\prime}$. This scalar is input to the hyperbolic tangent function, mapping $\mathbf{h}_{t,\text{q}}^{(i)\prime} \mapsto \mathbf{h}_{t,\text{q}}^{(i)}$. The row, i, is then incremented, and the aforementioned steps are repeated twenty times, resulting in the hidden output vector $\mathbf{h}_{t,\text{q}}$ at layer l. This process is repeated for all layers. Accordingly, the hidden output vector of the previous layer is locally stored in registers, i.e., $\mathbf{x}_t=\mathbf{h}_{t,\text{q}}$ for layers $l>0$.

The neuron controller then waits for the next sensor sample to arrive and upon arrival repeats the aforementioned process for the number of timesteps set as $T_s=50$. Once 50 timesteps have passed, the NPU acts as linear layer with a rectified linear unit (ReLU) activation function, outputting classified vector, $\hat{\mathbf{y}}$.

2) Weight Controller and Structure: Fig. 5 shows the memory structure, encompassing the storage of weight and quantization parameters by means of ROM IP that allocates a specific number of EBRs. An EBR possesses a word length of 16 bits and a total of 256 addresses. As a result, 22 EBRs are required to store the weights of the RNN in accordance with the parameters outlined in Table I, given that the RNN word length amounts to 352 bits.

The weight controller acquires information about the RNN state from the neuron controller, determined by the layer, row, and column indices (l, i, and m), and the selection of the weight matrix ($\mathbf{U}_{\text{q}}$ or $\mathbf{W}_{\text{q}}$). The address to be loaded is determined by the layer and row indices, and whether a quantization parameter or a weight is requested. Upon receiving the 352-bit word from the ROM IP, the weight controller picks a 32-bit segment to return to the neuron controller. In the case of requesting a quantization parameter, no offset is added and the forwarded data are selected based on the index stored in the

TABLE II
NUMBER OF 4-BIT LUTS, 1-BIT REGISTERS, AND 4-KBIT EBR AND % OF TOTAL USED BY MODEL A, B, AND C ON THE LATTICE ICE40UP5K FPGA

Architecture	LUTs	Registers	EBRs
a	3764 (71%)	645 (12%)	24 (80%)
b	2769 (52%)	947 (18%)	14 (46%)
c	3172 (60%)	717 (14%)	17 (56%)
Lattice ICE40UP5K	5280	5280	30

Parameter LUT (PLUT), selected with wire "par_i." In case of a weight, two scenarios transpire: 1) a bias weight, $\mathbf{b}_{\mathrm{q}}^{(i)}$, is picked by taking the initial 32 bits of the 352-bit word and transmitted to the neuron controller; and 2) an element of the hidden or input weight matrix, $\mathbf{U}_{\mathrm{q}}^{(i,m)}$ or $\mathbf{W}_{\mathrm{q}}^{(i,m)}$, is requested, and the transmitted element is selected from the word based on column index m.

3) Vector Controller: To store the hidden vectors of RNN layers, $\mathbf{h}_{t,\mathrm{q}}$, each containing N elements, a single EBR is utilized. In this EBR, each address stores a single element, $\mathbf{h}_{t-1,\mathrm{q}}^{(m)}$, where m is the column index. During RNN operation, $\mathbf{h}_{t-1,\mathrm{q}}^{(m)}$ is loaded for every element multiplication involving the hidden weight matrix [(5)]. Conversely, the resulting scalar $\mathbf{h}_{t,\mathrm{q}}^{(i)}$ from a single RNN cycle in (4) is stored based on row index i.

4) Processing Engine: Fig. 6 shows the processing engine that performs matrix multiplications by means of N vector multiplications over row index i, leading to $\mathbf{h}_{t,\mathrm{q}}$ of layer l. For a single vector multiplication, we propose three operations using four 8-bit registers, namely, substract and multiply-accumulate (SUB_MAC, OP1), multiply and shift-accumulate (MUL_SAC, OP2), and add-accumulate (ADD_ACC, OP3). OP1 represents the operation of substracting the zeropoint from matrix elements and multiply-accumulating these elements over a single row, column by column. Using registers a–b, this operation outputs an accumulated scalar, denoted as "res." This operation is performed twice, once per matrix, at a total of $2N$ times, resulting in scalars $\mathbf{U}_{\mathrm{q}}^{(i)\prime}$ and $\mathbf{W}_{\mathrm{q}}^{(i)\prime}$ [(5) and (6)]. Then OP3 is initiated, summing up the three intermediate vector, $\mathbf{b}_{\mathrm{q}}^{(i)}$, $\mathbf{U}_{\mathrm{q}}^{(i)\prime}$ and $\mathbf{W}_{\mathrm{q}}^{(i)\prime}$. Finally, OP2 leads to $\mathbf{h}_{t,\mathrm{q}}^{(i)\prime}$ by means of a shift-and-add of the result of OP3 with M_{int}, M_{shift} and adding Z_{T}.

5) Hyperbolic Tangent: The hyperbolic tangent is a nonlinear function, commonly implemented using piecewise linearization or a one-to-one mapping, stored within memory or a LUT. Here we propose to perform a 1-to-1 mapping as the range at 8-bit bitwidth is sufficiently small, i.e., $\mathbf{h}_{t,\mathrm{q}}^{(i)\prime} \in [0,\ 255] \mapsto \mathbf{h}_{t,\mathrm{q}}^{(i)} \in [107,\ 149]$. Consequently, the hyperbolic tangent is implemented using a single EBR, and the address is determined by the value of $\mathbf{h}_{t,\mathrm{q}}^{(i)\prime}$, yielding $\mathbf{h}_{t,\mathrm{q}}^{(i)}$.

B. Measurement Setup and Implementations

Fig. 7(a) shows the experimental setup utilized to assess power consumption and validate the RNN implementation on the Lattice ICE40UP5K FPGA, employing the Lattice ICE40UP5K-B-EVN breakout board. The RNN I/Os, comprising 4 SPI pins for the SPI are connected to the digital I/O of

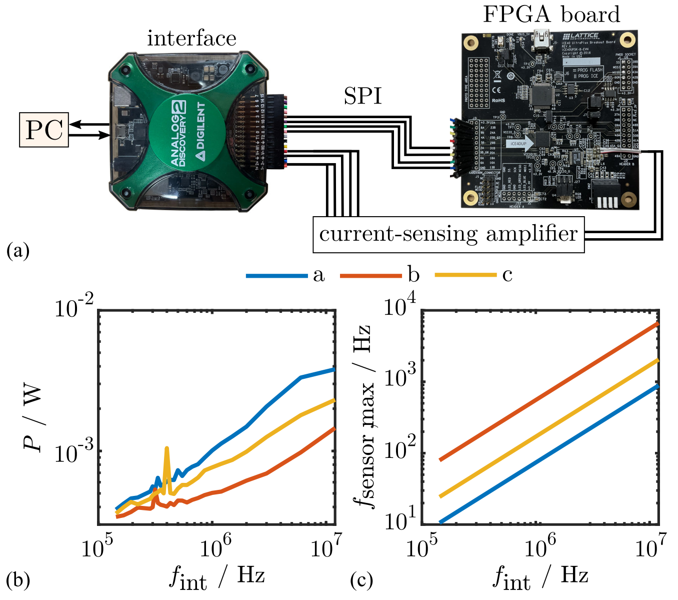


Fig. 7. (a) FPGA measurement setup using Analog Discovery 2 and Lattice ICE40UP5K-B-EVN breakout board. (b) Measured power consumption over the internal clock frequency. (c) Maximum sensor frequency for real-time RNN operation, a function of the clock frequency and number of clock cycles [(18)–(21)].

the Digilent Analog Discovery 2 (Digilent Inc., Pullman WA). To verify the design, emulated sensor data are transmitted to the FPGA via this interface from a personal computer. After T_{s} transmissions of $\mathbf{x}_t$, the classified vector, $\hat{\mathbf{y}}$, from the FPGA is compared with the original Python-based model.

To determine the power consumption of the implementation, we measure the voltage drop across a resistor connected in series with the core FPGA voltage of 1.2 V. The voltage measurement is initiated when an SPI message containing $\mathbf{x}_t$ is sent, resulting in a voltage increase due to RNN operation. The design for the FPGA implementation is written in SystemVerilog and parametrized, allowing customization of architectural parameters, sensor frequency, and a customized RNN weights. The code for the conversion from a TensorFlow 2.0 simple RNN model and the design files are publicly available and accompanied by a comprehensive guide [26].

Fig. 7(b) plots the power consumption of the FPGA implementations across a range of internal clock frequencies, f_{int}, from 142 kHz to 12 MHz, while Fig. 7(c) plots the maximum supported sensor frequency. The power consumption exhibits a near-logarithmic increase with the internal clock frequency, spanning from 340 μW at 145 kHz (model b) to 3.81 mW at 12 MHz (model a), with verification carried out on model c. Table II enumerates the resources utilized by the three models. The maximum sensor frequency is determined by the input dimension (I), layer width (N), number of layers (L), and output classes (O). We analytically determined and verified in post-synthesis simulation the number of cycles per timestep as

$$p_{\mathrm{cycles}} = p_{\mathrm{RNN}} + p_{\mathrm{lin}} \tag{18}$$

$$p_{\mathrm{RNN}} = N^2(8L-4) + N(14L+4I) + I - 2L \tag{19}$$

$$p_{\mathrm{lin}} = O(4N+10) + 3 \tag{20}$$

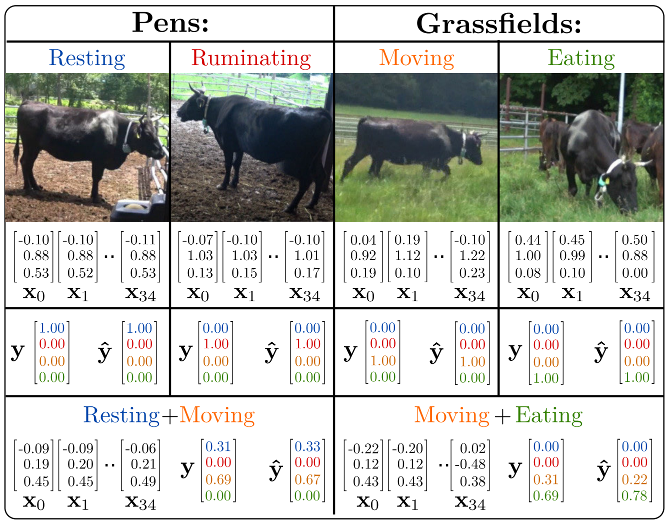


Fig. 8. Single behavior and mixed behavior classification scenarios of cow behavior. $\mathbf{x}_t$, $\mathbf{y}$, and $\hat{\mathbf{y}}$ describe the corresponding accelerometer data, ground truth distribution coefficients, and estimated coefficients by the implemented RNN on FPGA respectively.

TABLE III
COMPARISON OF ANIMAL BEHAVIOR ESTIMATION STUDIES IMPLEMENTING ALGORITHMS IN HARDWARE, INCLUDING POWER CONSUMPTION MEASUREMENTS OR ESTIMATIONS

Reference	This Work 2023	[34] 2021	[35] 2021	[36] 2019
Platform	Lattice FPGA	STMicro MCU	Lattice FPGA	TI MCU
Core voltage [V]	1.2	3	1.2	2
Core frequency [kHz]	146	80,000	2.9	24,000
Technology node [nm]	40	90	40	130
Application	Cow	Horse	Cow	Sheep
# of Behaviors	4	3	4	5
Algorithm	RNN	MLP	DT	LR
Bit-width	8	32	8	16
Accuracy [%]	95.1	98.3	86.8	89.6
Power [µW]	360	31,100	216	4300
Current [µA]	300	10,700	180	2150

where p_{cycles} represents the number of clocks for the NPU to process a single RNN timestep. Specifically, this represents the number of cycles necessary for the NPU to be able to accept the subsequent input vector from sensors, $\mathbf{x}_{t+1}$. The maximum sensor sampling frequency is determined as

$$f_{\text{sensor,max}} = \frac{f_{\text{in}}}{p_{\text{cycles}}} \tag{21}$$

where f_{in} can be any fraction of the 12 MHz oscillator on the breakout board using a clock divider.

IV. Proof of Concept: Cow Behavior Estimation With a 3-D Accelerometer

Utilizing the proposed TLM architecture and its implementation, we demonstrate a proof of concept by employing a RNN trained on a publicly available cow dataset [23] for the task of estimating cow behavior. This application is a part of a burgeoning field known as PLF, which concentrates on the observation, interpretation of the behavior and control of animals [37], [38]. Other PLF examples encompass precise control of cow feeding, fertility monitoring, and early disease detection. These practices have been shown to decrease greenhouse gas emissions and reduce antibiotic usage, suggesting that PLF can enhance the efficiency of milk and meat production per unit of emissions [38], [39], [40]. Decision Trees and Support Vector Machines are among the most commonly employed machine learning algorithms for PLF, with sensors such as accelerometers, video, gyrometers, and GPS being widely utilized [35], [41].

A. Implementation Results

Table I shows the architectural parameters for the cow behavior RNN model, referred to as model c. With a layer width of $N = 13$, inputs from a 3-D accelerometer at 25 Hz and 4 output coefficients ($O = 4$), model c is smaller than the RNN used for describing the architecture in Section III, model a. The 4 output coefficients represent the prevalence of 4 behaviors, i.e., eating (Eat), resting (Res), rumination (Rum), and Moving (Mov). For comparison purposes with other relevant works, the top-1 accuracy is utilized, taking the behavior that has the largest coefficient in output vector $\hat{\mathbf{y}}$.

Initially, we load the weights of the Tensorflow 2.0 model into our Python-based implementation of the shared-scale RNN. Utilizing sample cow behavior data, the Python model generates a ".mem" file that replicates the model weights in accordance with the memory composition depicted in Fig. 5. Subsequently, we adjust the parameters of the HDL design according to Table I, point to the created memory file, and set the sensor frequency to 25 Hz. With these parameters and a sensor frequency of $f_{\text{sensor}} = 25$ Hz, the internal frequency is set to $f_{\text{int}} = 146$ kHz using (21).

Fig. 8 shows six cow behavior regression scenarios and snapshots of actual camera footage. Here, $\mathbf{x}_t$ is the corresponding accelerometer data. The behavior coefficients are represented by $\mathbf{y}$ and $\hat{\mathbf{y}}$, i.e., the ground truth determined the cow dataset and the regressed coefficients obtained from the implemented RNN on FPGA with the setup described in Section III-B.

Table III compares the performance of the RNN on cow behavior with works that show hardware implementations with the task of estimating animal behavior. Compared to the original RNN model on Tensorflow 2 of [22], our RNN implementation shows a minimal top-1 accuracy loss from 95.2% to 95.1%. The measured power consumption of the implemented cow behavior model is 360 µW. Although this is 1.66× higher than [35], the accuracy increases by 8.3%. Compared to works that show implementations on an STMicro (Geneva, Switzerland) and Texas Instruments (Dallas, TX) microcontroller, our implementation consumes only 1.16% and 8.37% of reported power consumption at 31 100 µW and 4300 µW [34], [36]. In contrast, these implementations show an accuracy higher, increase of 3.2%, and lower, drop of 5.5% for a 3 and 5 behavior classification, respectively.

The considerable advantage in power consumption when using an FPGA for AI implementation is evident, as our proposed TLM hardware architecture applied to the shared-scale RNN can be leveraged. As a result, this proof of concept on itself has the potential to significantly impact the field of PLF by providing a low-power and accurate solution for monitoring animal behavior in real-time.

V. Discussion

In this work, we have made three significant contributions to the field. First, we introduced a shared-scale, integer-only RNN that reduces the number of fixed-point multiplications to one third. Second, we proposed a hardware architecture designed to minimize resource utilization for implementation and incorporated a real-time processing scheme aligned with sensor sampling frequency, referred to as TLM. The key advantage of this hardware architecture is its ability to minimize the required logic for implementation while trading off throughput. Although this is not ideal for high-throughput applications such as machine translation and audio processing, low-frequency sensor applications remain unaffected. This distinction arises because, unlike machine translation where the input sample space is available from the onset, sensor input arrives on a sample-by-sample basis. Third, we demonstrated a proof of concept by implementing the proposed TLM architecture on a highly compact FPGA, the Lattice ICE40UP5K, for a critical application in PLF. To the best of our knowledge, there is no existing integer-only RNN with a resource-minimized hardware architecture and a processing scheme tailored for sensor applications at the edge. Consequently, we believe this work, accompanied by the code and guide made publicly accessible in [26], substantially advances the fields of IoT and Edge-AI by enabling multilayer RNNs, neural networks proven to be capable of solving various complex tasks, to be implemented on nearly any FPGA.

The proposed RNN hardware architecture currently presents several potential improvements that we suggest exploring. At present, the RNN only supports layers with equivalent layer widths. Additionally, the input vector cannot have a size that exceeds the layer width. Adding the possibility of multiple linear classification layers, commonly used in foundation models, could be a valuable improvement. Furthermore, the current RNN can only be employed in a many-to-one setup, where the final layers hidden output vector is used by the linear output layer for classification. Enabling many-to-many, varying layer widths, and incorporating multiple linear output layers may pave the way for implementing encoder–decoder structures and natural language processing models on smaller hardware platforms, albeit with a trade-off in processing speed. For these applications, we believe the TM architecture would be more advantageous in balancing speed and resource use.

In addition to architectural improvements, theoretical or model-based enhancements and modifications can be considered. At present, the only supported type of RNN is its simplest form. However, many applications utilize long-short term memory or gated recurrent units. These RNN types address the vanishing-gradient problem and maintain longer temporal memory during training. Furthermore, recent advances have been made in training sparse RNNs. Drawing inspiration from the stability of signals in nature, [42] introduced a delta mechanism that skips matrix multiplications if the difference in hidden or input vector elements between timesteps remains below a threshold Θ, referred to as Delta RNN. In fact, the examination of accelerometer sensor data (Fig. 8) and hidden output vector elements confirms this observation on the task of cow behavior estimation. Accelerators on FPGA developed with Delta RNN on GRU and LSTM have demonstrated considerable speed-ups and high percentages of weight sparsity for negligible accuracy loss on high-throughput natural language processing benchmarks such as TIMIT and Librispeech [43], [44].

VI. Conclusion

In light of these advancements, it is conceivable to extend this work to include these sophisticated RNN types and techniques, thereby unlocking greater potential for a wider range of applications and further enhancing the efficiency and performance of the proposed RNN architecture. The integration of our proposed TLM architecture with these theoretical advancements is posited as a subsequent step for the extensive use of RNNs in IoT and resource-constrained Edge-AI applications. Moreover, the synergy between such theoretical and architectural advancements could bridge the gap between resource-limited hardware and large-scale RNN models. Surmounting this barrier would enable the execution of complex tasks in real time, including machine translation, control, and reinforcement learning, on low cost, widely available FPGAs, thus contributing to the democratization of AI.

Conflict of Interest Statement

Ludovico Minati discloses previous consulting for a subsidiary of Lattice Semiconductor Corporation Inc., regarding topics not related to the presented research.

References

[1] J. Jumper et al., "Highly accurate protein structure prediction with AlphaFold," *Nature*, vol. 596, pp. 583–589, Aug. 2021.

[2] J. Degrave et al., "Magnetic control of tokamak plasmas through deep reinforcement learning," *Nature*, vol. 602, pp. 414–419, Feb. 2022.

[3] R. T. Yarlagadda, "Internet of Things & artificial intelligence in modern society," *Int. J. Creat. Res. Thoughts*, vol. 10, pp. 2320–2882, Jan. 2018.

[4] K. Rose, S. Eldridge, and L. Chapin, "The Internet of Things: An overview," *Internet Soc.*, vol. 80, pp. 1–50, Oct. 2015.

[5] A. S. Rao and G. Verweij, "Sizing the prize, what's the real value of AI for your business and how can you capitalise," PwC Publication, 2018, pp. 1–32.

[6] B. Dong, Q. Shi, Y. Yang, F. Wen, Z. Zhang, and C. Lee, "Technology evolution from self-powered sensors to AIoT enabled smart homes," *Nano Energy*, vol. 79, Jan. 2021, Art. no. 105414.

[7] D. Sehrawat and N. S. Gill, "Smart sensors: Analysis of different types of IoT sensors," in *Proc. 3rd Int. Conf. Trends Electron. Informat. (ICOEI)*, Apr. 2019, pp. 523–528.

[8] J. Finnegan and S. Brown, "An analysis of the energy consumption of LPWA-based IoT devices," in *Proc. Int. Symp. Netw., Comput. Commun. (ISNCC)*, Jun. 2018, pp. 1–6.

[9] Y. Mao, C. You, J. Zhang, K. Huang, and K. B. Letaief, "A survey on mobile edge computing: The communication perspective," *IEEE Commun. Surveys Tuts.*, vol. 19, no. 4, pp. 2322–2358, 4th Quart., 2017.

[10] R. S. Sinha, Y. Wei, and S.-H. Hwang, "A survey on LPWA technology: Lora and NB-IoT," *ICT Exp.*, vol. 3, no. 1, pp. 14–21, Mar. 2017.

[11] V. Mazzia, A. Khaliq, F. Salvetti, and M. Chiaberge, “Real-time apple detection system using embedded systems with hardware accelerators: An edge AI application,” *IEEE Access*, vol. 8, pp. 9102–9114, 2020.
[12] O. Bringmann et al., “Automated HW/SW co-design for edge AI: State, challenges and steps ahead,” in *Proc. Int. Conf. Hardw./Softw. Codesign Syst. Synth.*, Sep. 2021, pp. 11–20.
[13] Y. Chi, W. Qiao, A. Sohrabizadeh, J. Wang, and J. Cong, “Democratizing domain-specific computing,” *Commun. ACM*, vol. 66, no. 1, pp. 74–85, Jan. 2023.
[14] C. Xu et al., “The case for FPGA-based edge computing,” *IEEE Trans. Mobile Comput.*, vol. 21, no. 7, pp. 2610–2619, Jul. 2022.
[15] T. Young, D. Hazarika, S. Poria, and E. Cambria, “Recent trends in deep learning based natural language processing,” *IEEE Comput. Intell. Mag.*, vol. 13, no. 3, pp. 55–75, Aug. 2018.
[16] S. Mittal and S. Umesh, “A survey on hardware accelerators and optimization techniques for RNNs,” *J. Syst. Archit.*, vol. 112, Jan. 2021, Art. no. 101839.
[17] Y. Wu et al., “Google’s neural machine translation system: Bridging the gap between human and machine translation,” 2016, *arXiv:1609.08144*.
[18] B. Zhao, X. Li, and X. Lu, “CAM-RNN: Co-attention model based RNN for video captioning,” *IEEE Trans. Image Process.*, vol. 28, no. 11, pp. 5552–5565, Nov. 2019.
[19] C. Gao, R. Gehlhar, A. D. Ames, S. Liu, and T. Delbruck, “Recurrent neural network control of a hybrid dynamical transfemoral prosthesis with EdgeDRNN accelerator,” in *Proc. IEEE Int. Conf. Robot. Autom. (ICRA)*, May 2020, pp. 5460–5466.
[20] Y. Chen, Y. Xie, L. Song, F. Chen, and T. Tang, “A survey of accelerator architectures for deep neural networks,” *Engineering*, vol. 6, no. 3, pp. 264–274, Mar. 2020. [Online]. Available: https://www.sciencedirect.com/science/article/pii/S2095809919306356
[21] P. Dong et al., “RTMobile: Beyond real-time mobile acceleration of RNNs for speech recognition,” in *Proc. 57th ACM/IEEE Design Autom. Conf. (DAC)*, Jul. 2020, pp. 1–6.
[22] J. Bartels et al., “TinyCowNet: Memory- and power-minimized RNNs implementable on tiny edge devices for lifelong cow behavior distribution estimation,” *IEEE Access*, vol. 10, pp. 32706–32727, 2022.
[23] H. Ito et al., “Japanese black beef cow behavior classification dataset,” Sep. 2021, doi: 10.5281/zenodo.5399259.
[24] M. Abadi et al. (2015). *TensorFlow: Large-Scale Machine Learning on Heterogeneous Systems*. [Online]. Available: http://tensorflow.org/
[25] F. Chollet. (2015). *Keras*. [Online]. Available: https://github.com/fchollet/keras
[26] J. Bartels, A. Hagihara, L. Minati, K. K. Tokgoz, and H. Ito, “An integer-only resource-minimized RNN on FPGA for low-frequency sensors in edge-AI,” Apr. 2023, doi: 10.5281/zenodo.7800728.
[27] B. Jacob et al., “Quantization and training of neural networks for efficient integer-arithmetic-only inference,” in *Proc. IEEE/CVF Conf. Comput. Vis. Pattern Recognit.*, Jun. 2018, pp. 2704–2713.
[28] E. Sari, V. Courville, and V. Partovi Nia, “IRNN: Integer-only recurrent neural network,” 2021, *arXiv:2109.09828*.
[29] J. L. Elman, “Finding structure in time,” *Cognit. Sci.*, vol. 14, no. 2, pp. 179–211, Mar. 1990.
[30] S. Han, H. Mao, and W. J. Dally, “Deep compression: Compressing deep neural networks with pruning, trained quantization and Huffman coding,” 2015, *arXiv:1510.00149*.
[31] S. Hochreiter and J. Schmidhuber, “Long short-term memory,” *Neural Comput.*, vol. 9, no. 8, pp. 1735–1780, Nov. 1997.
[32] K. Cho et al., “Learning phrase representations using RNN encoder–decoder for statistical machine translation,” 2014, *arXiv:1406.1078*.
[33] A. Shawahna, S. M. Sait, and A. El-Maleh, “FPGA-based accelerators of deep learning networks for learning and classification: A review,” *IEEE Access*, vol. 7, pp. 7823–7859, 2019.
[34] J. P. Dominguez-Morales, L. Duran-Lopez, D. Gutierrez-Galan, A. Rios-Navarro, A. Linares-Barranco, and A. Jimenez-Fernandez, “Wildlife monitoring on the edge: A performance evaluation of embedded neural networks on microcontrollers for animal behavior classification,” *Sensors*, vol. 21, no. 9, p. 2975, Apr. 2021.
[35] J. Bartels et al., “A 216 μW, 87% accurate cow behavior classifying decision tree on FPGA with interpolated Arctan2,” in *Proc. IEEE Int. Symp. Circuits Syst. (ISCAS)*, May 2021, pp. 1–5.
[36] S. P. L. Roux, R. Wolhuter, and T. Niesler, “Energy-aware feature and model selection for onboard behavior classification in low-power animal borne sensor applications,” *IEEE Sensors J.*, vol. 19, no. 7, pp. 2722–2734, Apr. 2019.
[37] D. Lovarelli, J. Bacenetti, and M. Guarino, “A review on dairy cattle farming: Is precision livestock farming the compromise for an environmental, economic and social sustainable production?” *J. Cleaner Prod.*, vol. 262, Jul. 2020, Art. no. 121409.
[38] E. Tullo, A. Finzi, and M. Guarino, “Review: Environmental impact of livestock farming and precision livestock farming as a mitigation strategy,” *Sci. Total Environ.*, vol. 650, pp. 2751–2760, Feb. 2019, doi: 10.1016/j.scitotenv.2018.10.018.
[39] Y. Blaise et al., “Influences of feeding behaviour and forage quality on diurnal methane emission dynamics of grazing cows,” *Precis. Livestock Farming*, vol. 17, pp. 759–769, Jan. 2017.
[40] J. Wilkinson and P. Garnsworthy, “Impact of diet and fertility on greenhouse gas emissions and nitrogen efficiency of milk production,” *Livestock*, vol. 22, no. 3, pp. 140–144, May 2017.
[41] R. García, J. Aguilar, M. Toro, A. Pinto, and P. Rodríguez, “A systematic literature review on the use of machine learning in precision livestock farming,” *Comput. Electron. Agricult.*, vol. 179, Dec. 2020, Art. no. 105826.
[42] D. Neil, J. H. Lee, T. Delbruck, and S.-C. Liu, “Delta networks for optimized recurrent network computation,” in *Proc. Int. Conf. Mach. Learn.*, 2017, pp. 2584–2593.
[43] C. Gao, D. Neil, E. Ceolini, S.-C. Liu, and T. Delbruck, “DeltaRNN: A power-efficient recurrent neural network accelerator,” in *Proc. ACM/SIGDA Int. Symp. Field-Program. Gate Arrays*, Feb. 2018, pp. 21–30.
[44] C. Gao, T. Delbruck, and S. Liu, “Spartus: A 9.4 TOP/s FPGA-based LSTM accelerator exploiting spatio-temporal sparsity,” *IEEE Trans. Neural Netw. Learn. Syst.*, early access, Jun. 10, 2022, doi: 10.1109/TNNLS.2022.3180209.

Jim Bartels (Graduate Student Member, IEEE) received the B.Sc. (Hons.) degree in electrical engineering from the Eindhoven University of Technology, Eindhoven, The Netherlands, in 2019, and the M.E. degree from the Department of Electrical and Electronic Engineering, Tokyo Institute of Technology, Tokyo, Japan, in 2021 with the “excellent master thesis award.” He is currently pursuing the Ph.D. degree in electrical and electronic engineering with the Tokyo Institute of Technology.

His current research interests include embedded machine learning, spiking neural networks, time-to-first-spike coding, quantization, and neural architecture search.

Aran Hagihara (Graduate Student Member, IEEE) received the B.E. degree in electrical engineering from the Tokyo Institute of Technology, Tokyo, Japan, in 2022, awarded “Excellent Undergraduate Student Award.” He is currently pursuing the M.E degree in electrical and electronic engineering with the Tokyo Institute of Technology.

His current research interests include embedded machine learning and digitally assisted RF circuits.

Ludovico Minati (Senior Member, IEEE) received the B.S. degree in physical science and the M.S. degree in medical physics from the Open University, Milton, keynes, U.K., in 2009, the M.S. degree in applied cognitive neuroscience from the University of Westminster, London, U.K., in 2008, the B.S. degree in physical science and the M.S. degree in medical physics from The Open University, in 2009, the Ph.D. degree in neuroscience from the Brighton and Sussex Medical School, Falmer, U.K., in 2012, and the D.Sc. (doktor habilitowany) degree in physics from the Institute of Nuclear Physics, Polish Academy of Sciences, Krakow, Poland, in 2017.

He has held research and consulting roles across public institutions and private companies, including the Institute of Nuclear Physics, Polish Academy of Sciences, the Carlo Besta Neurological Institute, Milan, Italy, and the Brighton and Sussex Medical School. He is currently a Specially Appointed Associate Professor with the Institute of Innovative Research, Tokyo Institute of Technology, Tokyo, Japan, an Affiliate Research Fellow with the Center for Mind/Brain Sciences, University of Trento, Italy, and a Freelance Research and Development Consultant. He has authored more than 140 articles and several patents. His research interests include non-linear dynamical systems, chaotic oscillators, reconfigurable analog and digital computing, functional magnetic resonance imaging, advanced techniques for bio-signal analysis, brain machine/computer interfaces, and robotics.

Dr. Minati is a European Engineer (Eur. Ing.), a Chartered Engineer and a member with the Institution of Engineering and Technology, U.K., a Chartered Physicist and a member with the Institute of Physics, U.K., and a Chartered Scientist and a member with the Institute of Physics and Engineering in Medicine, U.K. He is also a member with the Institute of Electronics, Information, and Communication Engineers (IEICE), Institute of Electrical Engineers of Japan, and the Japan Neuroscience Society.

Korkut Kaan Tokgoz (Member, IEEE) received the B.Sc. and M.Sc. degrees from the Electrical and Electronics Engineering Department, Middle East Technical University, Ankara, Turkey, in 2009 and 2012, respectively, and the M.Eng. and Ph.D. degrees from the Department of Physical Electronics, Tokyo Institute of Technology, Tokyo, Japan, in 2014 and 2018, respectively.

From 2018 to 2019, he worked as a Senior Researcher/Assistant Manager at NEC Corporation, Kawasaki, Kanagawa, Japan, where he was involved in 5G systems and fixed point-to-point wireless links. From 2019 to 2022, he worked as an Assistant Professor at the Tokyo Institute of Technology, Tokyo, Japan. He is currently working as a Faculty Member with the Faculty of Engineering and Natural Sciences, Sabanci University, Istanbul, Turkey. He also serves as the Co-Founder and CTO of Evrim Company Ltd., Yokohama, Japan. His research interests include analog/RF/millimeter-wave/sub-terahertz transceivers for wireless communications, low-power Edge-AI for monitoring systems, the IoT, sensors and systems, de-embedding, device characterization, and high-power, high-efficiency PAs for wireless systems.

Dr. Tokgoz was a recipient of several awards, scholarships, and grants, including the TUBITAK 2232-B International Fellowship for Early-Stage Researchers in 2022, the Marie Skłodowska-Curie Actions Post-Doctoral Fellowship in 2022, the SSCS Predoctoral Achievement Award in 2018, the IEEE MTT-S Graduate Student Fellowship in 2017, the IEICE Student Encouragement Prize in 2017, the Seiichi Tejima Overseas Student Research Award, and the IEEE/ACM ASP-DAC University LSI Design Contest 2017 Best Design Award.

Hiroyuki Ito (Member, IEEE) received the B.E. degree from the Department of Electronics and Mechanical Engineering, Chiba University, Chiba, Japan, in 2002, and the M.E. and Ph.D. degrees from the Department of Advanced Applied Electronics, Tokyo Institute of Technology, Yokohama, Japan, in 2004 and 2006, respectively.

From 2004 to 2007, he was a Research Fellow of the Japan Society for the Promotion of Science. He was a temporary Visiting Researcher and a Visiting Professor with the Communications Technology Laboratory, Intel Corporation, Hillsboro, OR, USA, in 2006 and 2007, respectively. He was an Assistant Professor at the Precision and Intelligence Laboratory, Tokyo Institute of Technology, from 2007 to 2013. From 2008 to 2010, he was with Fujitsu Laboratories Ltd., Yokohama, where he developed an RF CMOS transceiver and digital calibration techniques for mobile-WiMAX application. From 2013 to 2015, he was an Associate Professor at the Precision and Intelligence Laboratory, Tokyo Institute of Technology. Since 2016, he has been an Associate Professor with the Institute of Innovative Research, Tokyo Institute of Technology. He has been the Co-Funder/CEO of EVRIM Company Ltd., Yokohama, Japan, since 2020. His research interests include ultralow-power RF circuits, a high-sensitivity MEMS accelerometer, a low-noise and crystal-less RF synthesizer, and a cow management system exploiting IoT and deep/machine learning.

Dr. Ito is a member of the IEEE Solid-State Circuits Society, the Institute of Electronics, Information and Communication Engineers (IEICE), the Japan Society of Applied Physics (JSAP), and the Japanese Society of Agricultural Machinery and Food Engineers. He was a recipient of the European Solid-State Circuits Conference (ESSCIRC) 2008 Best Paper Award.