Received February 17, 2022, accepted February 24, 2022, date of publication March 2, 2022, date of current version March 30, 2022.

Digital Object Identifier 10.1109/ACCESS.2022.3156278

TinyCowNet: Memory- and Power-Minimized RNNs Implementable on Tiny Edge Devices for Lifelong Cow Behavior Distribution Estimation

JIM BARTELS[1], **(Graduate Student Member, IEEE),**
KORKUT KAAN TOKGOZ[1,2], **(Member, IEEE), SIHAN A**[1], **(Graduate Student Member, IEEE),**
MASAMOTO FUKAWA[1,3], **SHOHEI OTSUBO**[1,3], **CHAO LI**[1], **(Graduate Student Member, IEEE),**
IKUMI RACHI[1], **KEN-ICHI TAKEDA**[4], **LUDOVICO MINATI**[1], **(Senior Member, IEEE), AND**
HIROYUKI ITO[1], **(Member, IEEE)**

[1]Nano Sensing Unit, Institute of Innovative Research, Tokyo Institute of Technology, Yokohama 226-8503, Japan
[2]Research Institute for the Earth Inclusive Sensing (EISESiV), Tokyo Institute of Technology, Tokyo 152-8552, Japan
[3]TechnoPro Design, Tokyo 106-6135, Japan
[4]Institute of Agriculture, Academic Assembly, Shinshu University, Matsumoto, Nagano 390-8621, Japan

Corresponding author: Jim Bartels (bartels.j.aa@m.titech.ac.jp)

This work was supported in part by the Center Of Innovation (COI) Program of the Japan Science and Technology Agency (JST) under Grant JPMJCE1309; and in part by the Research Program on the Development of Innovative Technology through Bio-Oriented Research of the National Agriculture and Food Research Organization (NARO), Japan.

This work involved animals in its research. Approval of all ethical and experimental procedures and protocols was granted by the Institutional Animal Care and Use Committee of Shinshu University, Japan.

ABSTRACT Precision livestock farming promises substantial advantages in terms of animal welfare, product quality and reducing methane emissions, but requires continuous and reliable data on the animal's behavior. While systems suitable for use within the barn exist, grazing over long distances poses challenges. Here, we address this issue by proposing an ultra low-power Edge AI device, minimizing data transmission requirements and potentially improving accuracy as compared to classification-based solutions. Namely, we propose cow behavior distribution regression with Recurrent Neural Networks (RNNs), dubbed TinyCowNet, to estimate mixed-label sample spaces. Without quantization, the random search to minimize resources and maximize accuracy shows networks requiring a memory of 76kB on average and offering an accuracy up to 95.7%. These are implementable on a wide range of low-power Micro Controller Units (MCU) and Field Programmable Gate Arrays (FPGA). Furthermore, our proposed post-training full-integer quantization for RNNs combined with power estimation on 45nm CMOS using experimental literature shows a TinyCowNet occupying a memory around ≈2kB, having a hypothetical power consumption on the order of 200nW, delivering an accuracy of 95.2% and a Matthews correlation coefficient of 0.86. This work paves the way for the future creation of low-cost, highly accurate cow behavior estimation devices with long battery life that reduce the entry barriers currently hindering precision livestock farming outside the barn.

INDEX TERMS Animal behavior, application-specific integrated circuits (ASIC), field programmable gate arrays (FPGA), Internet of Things (IoT), Pareto optimization, recurrent neural networks (RNN), quantization.

I. INTRODUCTION

Livestock farming is fundamental for economies around the world, estimated to be worth 50% of the overall agricultural GDP, coupled with a historical doubling over 40 years (1969-2009) [1]. As such, a United Nations report in 2013 shows that 14.5% of all human-induced greenhouse gas emissions originate from this industry, to which cattle farming contributes more than half [2]. In addition to emissions, the livestock sector causes environmental problems such as freshwater eutrophication and soil over-nutrition [3]. Another doubling of global livestock product demand is expected

The associate editor coordinating the review of this manuscript and approving it for publication was Jenny Mahoney.

by 2050, as living standards attempt to improve in developing countries [4]. Therefore, this industry is one of the critical areas for succeeding in global sustainability and reducing emissions in line with the climate agreements [5].

Recently, the field of Precision Livestock Farming (PLF) has been gaining ground, which is based on the observation, interpretation of behavior and control of animals. This field aims to fulfil a number of related but distinct goals, including improving animal welfare and product quality, as well as enhancing the economic, social and environmental sustainability of livestock farming [6], [7]. For example, the precise control of cow feeding could reduce methane emissions by up to 11% [8]. Moreover, optimal fertility maintenance could reduce gas emissions by up to 20% per herd [9]. Furthermore, early disease detection and improved animal welfare reduce antibiotic use and culling [6]. These results imply that PLF can effectively improve the ratio of meat or milk product produced per unit of emissions. All these possible applications require accurate behavior estimation, which provides, albeit indirectly, awareness about livestock health and conditions [10]–[13]. However, the adoption of PLF is presently hindered by technical complexity and variable levels of reliability [7].

The field of dairy cow farming is highly automated, and many commercially-available solutions enable detailed, ongoing monitoring within the barn and milking stations [14]–[17]. Compared to dairy cows, the diet of beef cows is more grass-based, which is necessary to achieve the growth rate expected for meat production [18]. A low-cost and generally used method of providing this nutrition is by letting cows graze on large pastures. This, however, poses particular challenges for monitoring technology, with no currently available solution for health and behavior monitoring outdoors over large areas. In this paper, we propose a workable solution to this challenge, developed initially in the context of Japanese beef cow farming, which traditionally involves grazing.

Behavior estimation systems suitable for this purpose can be created by implementing machine learning algorithms on the cloud. When in the context of the barn, short-range transmission for example via Bluetooth enables attaining a battery life on the order of 2 years [17]. However, in a grazing scenario the situation is different due to the longer distances and less predictable environment. This seems to often result in a shorter battery life caused by continuous sensor data transmission. To solve this issue, Edge Artificial Intelligence (AI) implements machine learning on the embedded devices themselves, producing a 1000-fold increase in battery life, enabling Low-Power-Wide-Area (LPWA) networks with temporally-sparse communication over long transmission distance [19], and thereby makes devices that can operate for years a concrete possibility [20], [21]. Besides, according to our knowledge, current animal behavior estimating machine learning systems, whether at the edge or on the cloud, often apply a moving average function (windowing) when extracting features. This creates a situation where single behaviors are rigidly assigned to mixed behavior data, generating significant error.

In this work, we describe the methodology of a random search with post-training integer-only quantization applied to RNNs, aiming to find energy- and memory-optimized architectures that regress cow behavior distribution from tri-axial accelerometer data. The results show highly accurate (>95%) distribution estimation with memory occupancy as low as ≈2kB and ≈43k Multiply-Accumulate (MAC) operations that, based on experimental literature [29], [30], we estimate would require 216nJ per inference at a power consumption of 154nW on 45nm CMOS. These networks are in principle implementable on low-cost hardware devices with an estimated battery life on the order of years. Therefore, we believe that this work will contribute to the broader adoption of PLF.

The contributions of this work are:

- An analysis of recent animal behavior Edge AI literature, large classification error when considering time-windowed data, and comparatively high power consumption. (Section III)
- The introduction of the notion of using a neural network such as RNN not to hardly classify behaviors, rather to estimate in a graded form their prevalence within each time window of the input time series. (Section III)
- The development of a novel distribution-regression algorithm based on parametrized RNNs that are optimized using random search, demonstrating highly accurate cow behavior estimation on low-power, low cost MCUs and FPGAs. (Sections III-VII and IX-X)
- A variable bit-width full-integer quantization scheme that maintains similar accuracy combined with 45nm CMOS RNN resource estimation, pointing to the possibility of future cow behavior estimation devices having years of battery life. (Sections VII-X)

II. RELATED WORK

In the academic literature, Edge AI systems suitable for animal behavior classification outside the barn are limited in number, and universally use accelerometers and learning algorithms.

Reference [22] showed a K-Nearest Neighbor (KNN) classifier running on an ARM Cortex processor achieving 96.2% accuracy for the discrimination of high vs. low activity of sows with 0.542mJ energy consumption per inference. Similarly, Reference [23] used heuristics, statistical analysis of raw sensor data, to achieve 90% precision in the classification of three postures combined with high and low activity of sows using an FPGA. These works did not classify any feeding behavior.

On the other hand, recent research on Edge AI applied to sheep, cattle and horses is available. Reference [24] used a Multi-Layer Perceptron (MLP) to achieve a precision of 80.8% in the classification of standing, walking, and trotting in horses. This MLP was implemented on a Nucleo MCU (ST Microelectronics, Spa, Agrate Brianza, Italy) using the Fast

TABLE 1. Literature survey of animal behavior classification with machine learning intended to be implemented on Edge AI hardware using tri-axial accelerometers.

Reference	[22] 2018	[23] 2011	[24] 2018	[25] 2021	[26] 2019	[27] 2021	[28] 2021	This Work 2021
Application	Sow	Sow	Horse	Horse	Sheep	Cow	Cow	Cow
Sensor position	Ear	Ear	Jaw	Jaw	Neck	Neck	Neck	Neck
Datasize (hh:mm)	1:25	10:40	00:17	00:13	16:11	3:17	01:30	3:17
Sampling frequency (Hz)	50	4	30	33	20	25	50	25
Window size in seconds (s)	1.28	120	?	?	6.4	2.56	10	-
Window size in samples	64	480	?	?	128	64	500	-
Algorithm	KNN	Heuristic	MLP	MLP	DT,LR	DT	MLP,DT	RNN
Number of behaviors	2	3	3	3	5	4	4	4
Device	MCU	FPGA	MCU	MCU	MCU	FPGA	MCU	Estimation*
Task	Classification	Classification	Classification	Classification	Classification	Classification	Classification	Regression
Features	3	2	3	3	34	3	3	3
Performance Measure	Accuracy	Precision	Precision	Accuracy	Accuracy	Accuracy	Correlation coefficient	Accuracy
Value	96.2%	90%	80.8%	98.3%	89.6%	86.8%	0.934	95.2%
Energy consumption	542 µJ	?	?	185.76 mJ	2050 µJ	557 µJ	?	216 nJ

* Power consumption estimation on 45nm CMOS based on experimental literature [29], [30]. Using only 2092 bytes, our model is also implementable on some of the smallest available MCUs and FPGAs (Table 5).

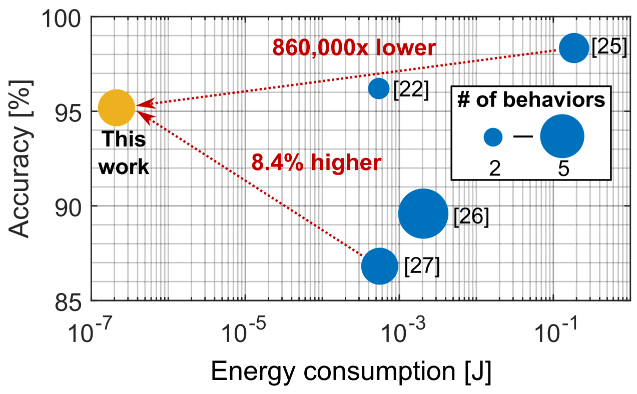


FIGURE 1. Energy consumption vs. accuracy of previous works compared to the present one. Dot size denotes the number of classified behaviors.

Artificial Neural Network library [31]. Later work shows an accuracy of 98.3% on the same three behaviors with a 163kB three-layer MLP floating-point model on a similar MCU, consuming 185.76mJ per inference [25].

Reference [26] combined energy consumption on a MSP430FR5-739 MCU (Texas Instruments Inc., Dallas TX) and accuracy to define and optimize a cost function towards selecting features and machine learning algorithms such as support vector machines, decision trees, random forest and KNN. Using linear regression with 34 features, they classified five sheep behaviors, including grazing, with 89.6% accuracy and 2050μJ energy consumption per inference. As for cattle-based systems, Reference [27] introduced a feature that determines the cow's neck angle in the horizontal-longitudinal axis. Using three features and a decision tree on a Lattice ICE40UP5k FPGA (Lattice Semiconductor Inc., Hillsboro OG), four cattle behaviors, including feeding and rumination, were classified with an accuracy of 86.8% and energy consumption of 557μJ per inference. Reference [28] shows an MLP network achieving a Matthews Correlation Coefficient (MCC) of 0.934 for the classification of four cow behaviors, including grazing and rumination using neck-attached tri-axial accelerometers, estimated to consume 2.29kB of memory. Notably, the MCC is a metric relatively insensitive to dataset imbalance [32], [33].

Table 1 shows details about each of these works, i.e., the window size, sampling frequency, sensor location, and so forth. Previous works use time-based windowing and classification that can lead to error, when considering the fact that animals provide mixed behavior data over time. In other words, real-life behaviors are not neatly segregated. Furthermore, all works implement either MLP or other learning algorithms that require feature extraction. This feature extraction is well-documented, but may bias data processing, resulting in sub-optimal feature sets. Hence, Ref. [34] introduced a new library that statistically determines optimal time-series feature extraction. It is difficult to compare to other works with windowing, since data are pre-processed differently for each dataset, altering performance. Therefore, we propose combining distribution regression of cow behavior with RNNs since they mimic the behavior of time windowing without requiring feature selection, and optimize their weights by training with stochastic gradient descent.

Fig. 1 shows a cow behavior distribution estimating RNN model created in this work with an 860k× lower energy consumption than the highest-consuming estimation algorithm [25] with 8.4% higher accuracy than the lowest energy-consuming ones [27]. Table 1 compares this work with existing ones; we introduce the first regression-based algorithm applied to cow behavior for future hardware applications. What further distinguishes this algorithm is the ability to learn features by itself by training weights with

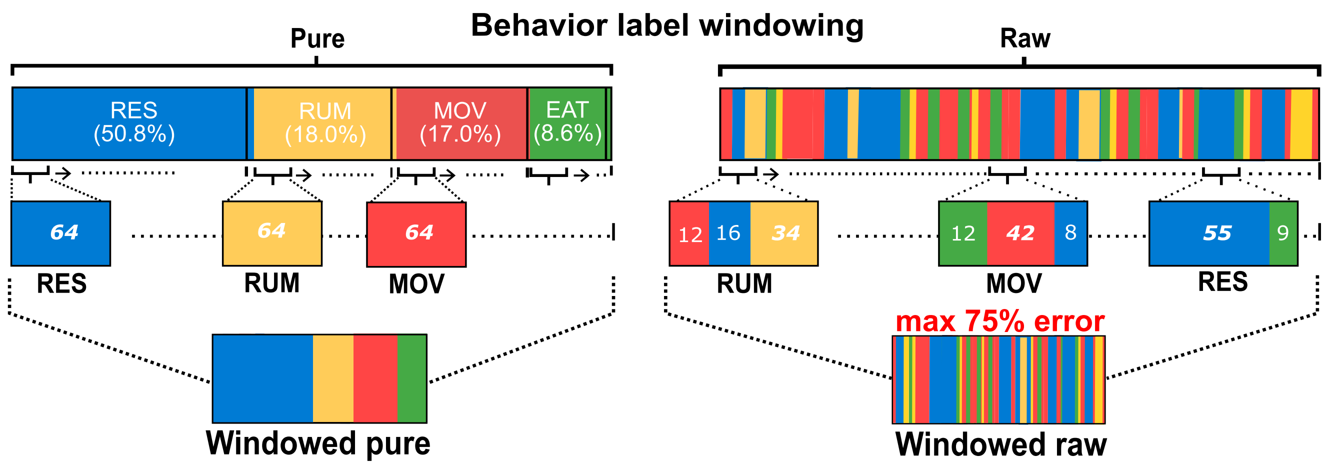


FIGURE 2. Label processing for pure data labels and raw data labels having a window size of 64. The colored stripes denote the behaviors of Resting (RES), Rumination (RUM), Moving (MOV) and Eating (EAT) with blue, yellow, red and green respectively.

stochastic gradient descent. Overall, to our knowledge, it is the most efficient cow behavior estimation algorithm in the literature. We achieve this by the methods that are described below.

III. WINDOWING WITH CLASSIFICATION ALGORITHMS

Windowing is the process whereby a data sequence is parcellated over a pre-set sample space, the window size, through time. For a window size l_s, for every l_s-th sample, the features used in an ML algorithm are extracted and averaged. This windowing reduces the amount of data, decreasing transmission size and processing load, and often improves performance when not considering the mixed-label error introduced. Time-based windowing is described by

$$\mathbf{X} = \frac{\sum_{i=1}^{l_s} \mathbf{m}(i)}{l_s} \tag{1}$$

where $\mathbf{m}$ is the vector representing the features and $\mathbf{X}$ is the processed data used in training and inference.

A. LABELS IN CLASSIFICATION ALGORITHMS

For a labeled dataset in animal behavior estimation (Section IV), each sample has a label assigned by an observer based on video footage. The features are real numbers, but the assigned labels are categorical. Therefore, one cannot apply an averaging method during windowing, and a different approach is necessary. According to our knowledge, most PLF literature does not clearly describe this problem, its impact, and the necessity to consider it in edge applications for farmers [6].

Fig. 2 shows windowing over a window size of $l_s = 64$ with the cow behavior dataset described in the next Section. For "pure" mode, the raw data are sorted into behavioral categories and then windowed, forming long homogeneous behavior collations. Nevertheless, this method corrupts the temporal relationships in the data, important for real-life representation. On the other hand, "raw" mode includes multiple categories within each window, preserving this relationship. However, multiple categories pose a problem for classification because there is a single label output. We consider two ways in which this problem could be solved. In the case of pure data, the label has to match the data input to be correct. However, in the case of raw mixed-label data, there is ambiguity. One way is to consider the majority label as in Fig. 2, as the correct classification output. With this method, there is a worst-case error of 75% in cases of equal behavior distribution, and this error depends on the number of behaviors, i.e.,

$$\epsilon_{\max} = 1 - \frac{1}{N_b} \tag{2}$$

which approaches unity for a large amount of behaviors, N_b. Another way is to apply a nearest-neighbor approach and assign a label approaching the mean feature value of each label category. However, this method is sensitive to outliers. If one window contains label categories with largely separated feature values, this assigns a nonsensical label. Therefore, we posit that assigning the majority label as performed in literature, might lead to large error whenever an animal changes behavior. For example, in Table 1, across the existing studies, the window size ranges from 1 up to 120 seconds.

B. PROPOSED DISTRIBUTION REGRESSION

Instead of a classification algorithm that involves mixed label error, we hereafter propose a regression algorithm to estimate a distribution. This regression eliminates the error in Eq. (2) as it estimates the label category proportion over an arbitrary-sized window. The output vector of this algorithm is described by

$$\mathbf{y} = [\alpha_1, \ldots \alpha_i, \ldots \alpha_{N_b}] \tag{3}$$

where N_b is the number of behaviors, four in this work, $\alpha_i \in [0, 1]$ is the prevalence for behavior i and $\mathbf{y}$ is the output of the regression algorithm. This $\mathbf{y}$ is a vector having unitary L_1

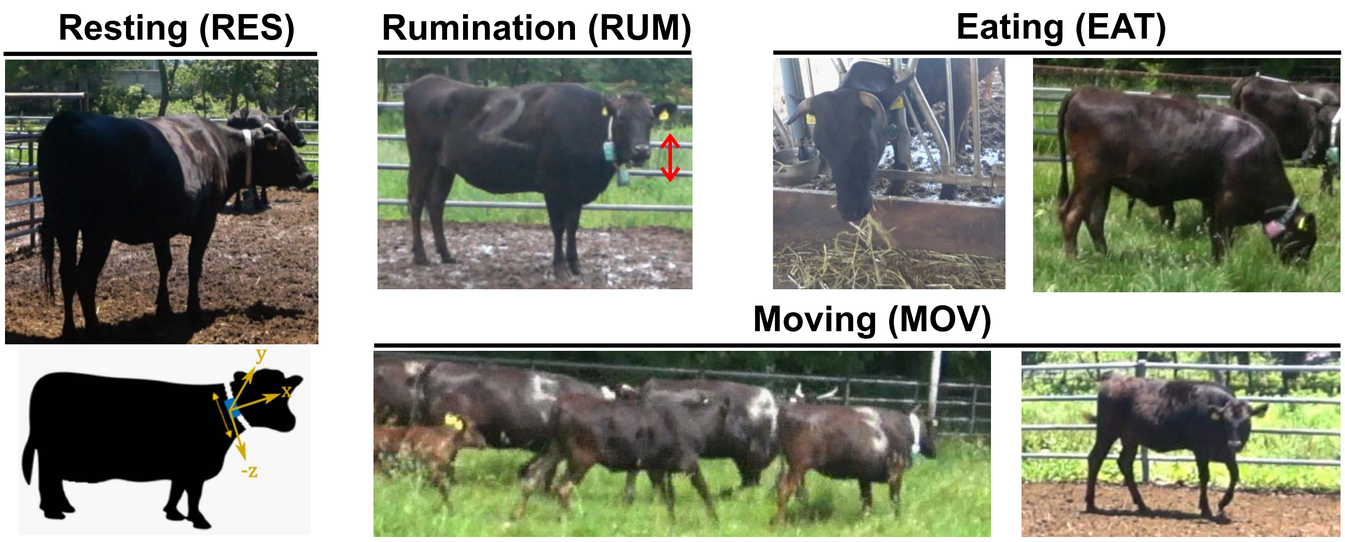


FIGURE 3. Video frames representing examples of the four labeled behaviors namely, Resting, (RES), Rumination (RUM), Moving (MOV) and Eating (EAT), alongside an illustration of the tri-axial accelerometer attached to the neck.

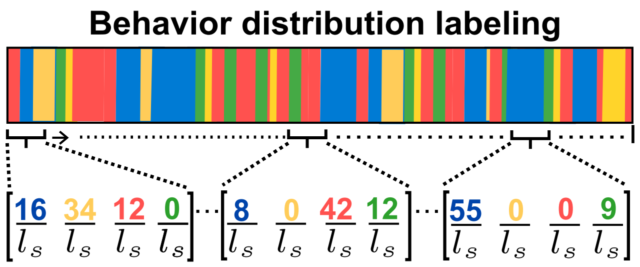


FIGURE 4. Method of labeling raw data into distribution vectors. Instead of a single label as in classification tasks, a distribution vector is used for regression. The l_s is the sample window size, normalizing the vector.

norm, i.e.,

$$||\mathbf{y}||_1 = \sum_{i=1}^{N_b} \alpha_i = 1 \tag{4}$$

where α_i is calculated as

$$\alpha_i = \frac{m_i}{l_s} \tag{5}$$

and m_i is the number of samples for each behavior i in a window. The amount of time per behavior in a window with window size l_s is then

$$t_i = \alpha_i \frac{l_s}{f_{\text{acc}}} \tag{6}$$

where f_{acc} is the sampling frequency of the accelerometer.

Fig. 4 shows raw data processed into distribution labels according to Eq. (3). This method allows similar data size reduction as classification, while enabling unbiased estimation of the time spent in each behavior.

IV. DATASET AND ANNOTATION

This Section describes the pre-labeled dataset used in the training and validation of the proposed neural network. Its preparation involved the gathering of 16-bit, 25Hz, ±2g tri-axis accelerometer data (type KX126-1063, Kionix Inc., Ithaca NY) accelerometers [35], attached to the neck of six adult Japanese black beef cows (Kuroge Washu) at a farm of Shinshu University in Nagano, Japan. Over two days, these cows were observed in farm pens and a grass field while being filmed with multiple video cameras. Hereafter, for each cow, three experts labeled the data from the video footage. In total, 691 minutes of unlabeled video were parsed into 197 minutes of high-quality labeled data by strict selection of segments that did not involve simultaneous behaviors or excessively frequent transitions. With an estimated 69 person-hours for this labeling and the data gathering at the farm, costs are high. Consequently, we are investigating self-supervised learning [36] that has been used in many applications [37], [38] but remains unexplored for cow behavior. Data gathering with these cows was reviewed and approved by the institutional animal care and use committee of Shinshu University, Japan. This dataset has been made publicly available under the Creative Commons CC BY-NC-ND license [39].

TABLE 2. Labeled data, description and corresponding prevalence.

Labeled behavior	Description	Samples (%)
Resting (RES)	resting while standing up	150130 (50.8%)
Rumination (RUM)	ruminating while standing up	53229 (18.0%)
Moving (MOV)	walking or running	50199 (17.0%)
Eating (EAT)	eating hay at stanchion or grazing	25547 (8.6%)
Others (Removed)	salt licking, mounting, drinking	16569 (5.6%)

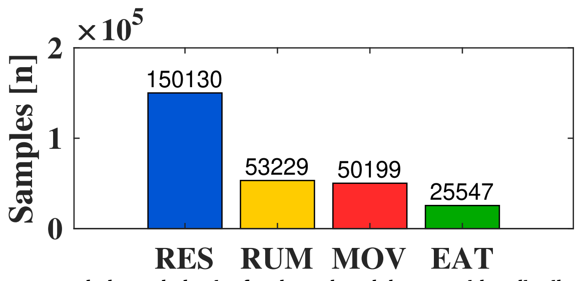


FIGURE 5. Labels per behavior for the reduced dataset with a distribution of roughly 6:3:3:1, which indicates large imbalance.

Fig. 3 shows video frames of the four behaviors and an illustration of the neck-attached accelerometer and its axes. Table 2 describes the labeled behaviors. There was access to water and hay in the pens with muddy floors, and the behaviors were primarily characterized by walking, ruminating, and resting. On the other hand, in the grass fields, where the cattle were allowed to graze freely, the primarily observed behaviors were grazing and running. In total, 94.4% of the 197 minutes of data represented resting (RES), rumination (RUM), eating (EAT), and moving (MOV). Therefore, we removed the remaining 5.6% that accounted for all other behaviors, having only a minor impact on representativeness. For the avoidance of misunderstanding, it should be clear that the purpose of this work is not to model the full spectrum of cow behaviors. The focus, rather, is on the most prevalent ones, changes in the distribution of which may signal hiddenly deteriorating health [12], [13]. Rare occurrences such as encounters with other animals were not included in the dataset.

Fig. 5 illustrates the imbalance in the reduced dataset, having a distribution of roughly 6:3:3:1, i.e., six times more resting than eating data. Methods such as oversampling minority classes or downsampling majority classes are often used to address such imbalance [40]. For example, oversampling may be simulated via random rotations of the accelerometer around the cow's neck [41]. However, here we decided not to apply these methods for three reasons. First, downsampling the majority classes would result in an unnecessarily smaller dataset. Second, the dataset represents the actual distribution of cow behavior. Therefore, oversampling would distort this distribution and create overfitting [40]. Third, the distribution of actual cow behavior is always imbalanced [42], [43] and should be preserved to obtain ecologically valid algorithm performance [44]. Data extension methods are not the focus of this work and are outside scope.

V. NEURAL NETWORK METHODS

This Section presents the methodology of different neural network layers essential for understanding the proposed model architecture in Section VI.

A. RECURRENT LAYERS

Recurrent Layers (RL) are algorithmic units with the following general input and output sequences

$$\mathbf{X} = \{\mathbf{x}_1; \mathbf{x}_2; \ldots \mathbf{x}_{T_s}\} \tag{7}$$

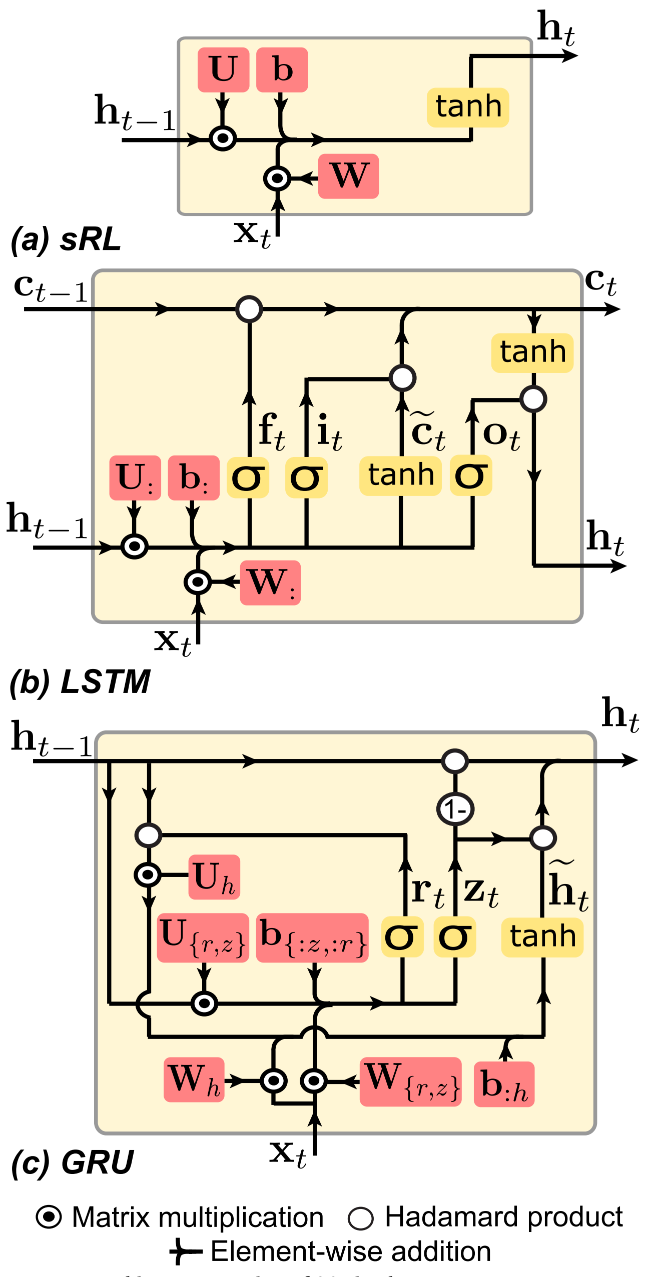


FIGURE 6. Architecture overview of (a) Simple Recurrent Layer, sRL, (b) Long-Short Term Memory, LSTM, and (c) Gated Recurrent Unit, GRU.

$$\mathbf{Y} = \{\mathbf{y}_1; \mathbf{y}_2; \ldots \mathbf{y}_{T_s}\} \tag{8}$$

where $\mathbf{X}$ represents the input sequence and $\mathbf{Y}$ the output sequence with a size of T_s timesteps. Using only the last output of $\mathbf{Y}$, $\mathbf{y}_{T_s}$, is commonly called many-to-one and is considered in this work. RLs can learn the characteristics of time-dependent problem data sets as they recur their outputs for each timestep. Fig. 6 shows three RNN architectures, equivalent to those in TensorFlow 2.0 with Keras backend [45], [46], that are used in this work.

1) SIMPLE RECURRENT LAYER (SRL)

Recurrent Layers have their roots in cognitive science, and early parallel computation [47], [48]. In its most

TABLE 3. Weight dimensions and multiply-accumulate (MAC) operation of the four different neural network layers. Variables *i*, *n* and *m* denote the input size, recurrent layer width and output vector size.

	Weight matrix size			Operations
Layer (k)	**U**	**W**	**b**	N. of MAC
sRL	$n \times n$	$i \times n$	$1 \times n$	$n^2 + ni$
LSTM	$4n \times n$	$i \times 4n$	$1 \times 4n$	$4n^2 + n(4i+3)$
GRU	$3n \times n$	$i \times 3n$	$2 \times 3n$	$3n^2 + n(3i+3)$
Linear	-	$n \times m$	$1 \times m$	mn

simplistic form, the simple recurrent layer (sRL) has a trainable bias, two trainable weight matrices, and one activation function. The model equation for this layer is

$$\mathbf{h}_t = \tanh(\mathbf{b} + \mathbf{U}\mathbf{h}_{t-1} + \mathbf{W}\mathbf{x}_t) \tag{9}$$

where $\mathbf{h}_t$, $\mathbf{h}_{t-1}$ and $\mathbf{x}_t$ represent the recurrent output, the previous recurrent output, and the input vectors. The recurrent, input, and bias matrices are denoted by $\mathbf{U}$, $\mathbf{W}$, and $\mathbf{b}$. Finally, the hyperbolic tangent, $\tanh(x)$, is chosen as activation function. Although least complex, this model suffers from vanishing-exploding gradient, complicating the training of large models. This problem can be solved with gradient clipping [49] or stochastic diagonal approximate greatest descent [50] during training.

2) LONG SHORT TERM MEMORY (LSTM)

Other than the above, Ref. [51] introduced Long Short Term Memory (LSTM) that addresses the vanishing-exploding gradient by introducing more gates. Since its formulation, LSTM has yielded a multitude of breakthroughs in sequence learning problems [52]. The model equations are

$$\mathbf{f}_t = \sigma(\mathbf{b}_f + \mathbf{U}_f\mathbf{h}_{t-1} + \mathbf{W}_f\mathbf{x}_t) \tag{10}$$
$$\mathbf{i}_t = \sigma(\mathbf{b}_i + \mathbf{U}_i\mathbf{h}_{t-1} + \mathbf{W}_i\mathbf{x}_t) \tag{11}$$
$$\mathbf{o}_t = \sigma(\mathbf{b}_o + \mathbf{U}_o\mathbf{h}_{t-1} + \mathbf{W}_o\mathbf{x}_t) \tag{12}$$
$$\widetilde{\mathbf{c}}_t = \tanh(\mathbf{b}_c + \mathbf{U}_c\mathbf{h}_{t-1} + \mathbf{W}_c\mathbf{x}_t) \tag{13}$$
$$\mathbf{c}_t = \mathbf{f}_t \circ \mathbf{c}_{t-1} + \mathbf{i}_t \circ \widetilde{\mathbf{c}}_t \tag{14}$$
$$\mathbf{h}_t = \mathbf{o}_t \circ \tanh(\mathbf{c}_t) \tag{15}$$

where $\mathbf{f}_t$, $\mathbf{i}_t$, and $\mathbf{o}_t$ are the forget, input and output vectors, respectively, the $\circ$ operator indicates the Hadamard product, and $\sigma(x)$ is the sigmoid activation function. Finally, there is an additional output vector, cell state vector $\mathbf{c}_t$, which is recurrent as $\mathbf{h}_t$, but not forwarded to the deeper layers.

3) GATED RECURRENT UNIT (GRU)

The GRU, introduced in Ref. [53], also addresses the vanishing-exploding gradient but reduces the gates and outputs. The model equations of the GRU are

$$\mathbf{z}_t = \sigma(\mathbf{b}_{rz} + \mathbf{b}_{iz} + \mathbf{U}_z\mathbf{h}_{t-1} + \mathbf{W}_z\mathbf{x}_t) \tag{16}$$
$$\mathbf{r}_t = \sigma(\mathbf{b}_{rr} + \mathbf{b}_{ri} + \mathbf{U}_r\mathbf{h}_{t-1} + \mathbf{W}_r\mathbf{x}_t) \tag{17}$$
$$\widetilde{\mathbf{h}}_t = \tanh(\mathbf{b}_{rh} + \mathbf{b}_{ih} + \mathbf{U}_h(\mathbf{r}_t \circ \mathbf{h}_{t-1}) + \mathbf{W}_h\mathbf{x}_t) \tag{18}$$

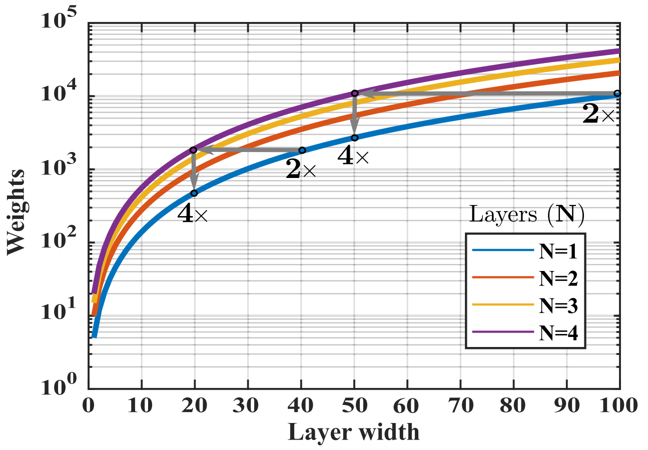


FIGURE 7. Layer width *n* for four different layer counts *N* plotted vs. total number of weights using simple Recurrent Layers. The gray lines indicate that a doubling of the layer width *n* for a quadrupling of the numbers of layers *N* is possible at the same weight cost.

$$\mathbf{h}_t = (1 - \mathbf{z}_t) \circ \mathbf{h}_{t-1} + \mathbf{z}_t \circ \widetilde{\mathbf{h}}_t \tag{19}$$

where $\mathbf{z}_t$, $\mathbf{r}_t$ and $\widetilde{\mathbf{h}}_t$ are the update, reset and candidate activation gates. Furthermore $\mathbf{b}_{r:}$ and $\mathbf{b}_{i:}$ represent the recurrent and input biases. The GRU has one more bias term than LSTM but a single recurrent output. An empirical study showed that the GRU outperforms the LSTM on all tasks except natural language processing [54]. However, both GRU and LSTM suffer from gradient decay for deep multi-layer networks, increasing the training time [55].

B. LINEAR LAYER (LL)

Linear Layers have two trainable weight matrices: the bias and the kernel. Each layer is a linear function, i.e.,

$$\mathbf{o} = \mathbf{W}\mathbf{x} + \mathbf{b} \tag{20}$$

where $\mathbf{o}$ and $\mathbf{x}$ are the output and input vectors. Furthermore, $\mathbf{W}$ and $\mathbf{b}$ represent the trainable kernel and bias. This layer is used frequently as an output layes, mapping features to class labels. Alexnet, which revolutionized deep learning for computer vision and Mobilenet, are networks that have leveraged these in the past [56], [57].

C. HARDWARE CONSIDERATIONS

Recurrent layers incur a high number of multiplications, proportional to the sizes of the input kernel, recurrent kernel, and bias matrices. Furthermore, activation functions are non-linear functions that require an approximation in digital hardware such as Application Specific Integrated Circuits (ASIC).

Table 3 shows the number of weights and Multiply-Accumulate (MAC) operations for each of the layers, i.e., sRL, GRU, LSTM, and Linear. The number of MACs increases linearly or quadratically with the number of weights. Thus, sparsifying the network, that is, minimizing the number of weights can reduce the amount of MAC operations considerably. The majority of dynamic energy

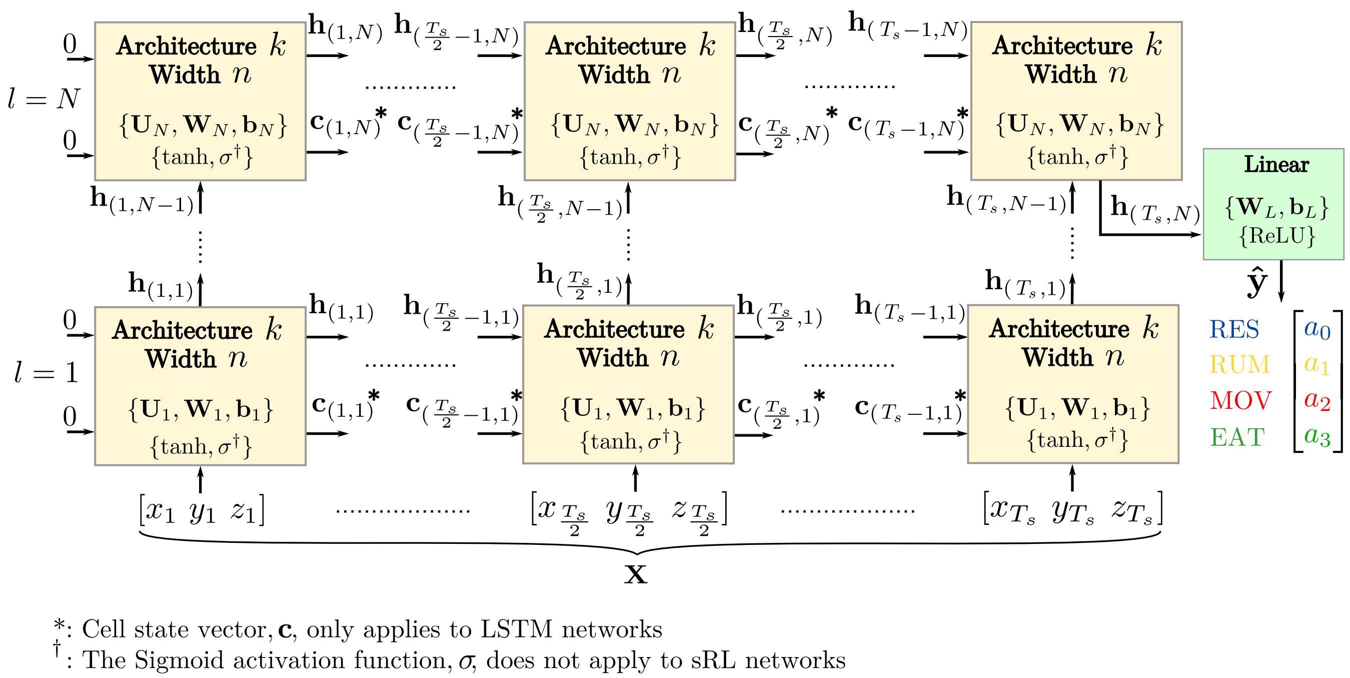

Architecture k Width n
$\{\mathbf{U}_N, \mathbf{W}_N, \mathbf{b}_N\}$ $\{\tanh, \sigma^\dagger\}$
$\{\mathbf{U}_1, \mathbf{W}_1, \mathbf{b}_1\}$ $\{\tanh, \sigma^\dagger\}$
$l = N$
$l = 1$
Linear $\{\mathbf{W}_L, \mathbf{b}_L\}$ $\{\text{ReLU}\}$
$\hat{\mathbf{y}}$
RES RUM MOV EAT
a_0 a_1 a_2 a_3
$[x_1\ y_1\ z_1]$ $[x_{\frac{T_s}{2}}\ y_{\frac{T_s}{2}}\ z_{\frac{T_s}{2}}]$ $[x_{T_s}\ y_{T_s}\ z_{T_s}]$
$\mathbf{X}$
*: Cell state vector, $\mathbf{c}$, only applies to LSTM networks
$\dagger$: The Sigmoid activation function, σ, does not apply to sRL networks

FIGURE 8. Cow behavior distribution estimating neural network, rolled out over time from left to right. The yellow blocks represent the recurrent layers with outputs, hidden state vector **h** and cell state vector **c**, and contain the trainable weights {**U**, **W**, **b**}. The green block represents the linear output layer with trainable weights {$\mathbf{W}_L$, $\mathbf{b}_L$}. Each state vector is indicated with entries (p, q) where p is the current timestep and q is the respective layer. The hyperparameters included in the search space are architecture k, width n and layer count N and timesteps T_S. Finally, $\hat{\mathbf{y}}$ denotes the regressed distribution vector containing cow behavior prevalence.

consumption of neural networks on hardware is due to memory access [58], which can be reduced by replacing dynamic with static RAM (e.g., 128× on 45nm CMOS [29]). Nevertheless, minimizing n is essential.

Fig. 7 shows the relationship of the weights with layer width n and the number of stacked layers N for the sRL. With every doubling of layer width n, layer count can be quadrupled at the same weight cost. Thus, stacking multiple recurrent layers increases the amount of memory and multiplications only linearly, in contrast with increasing the weight dimensions with parameter n. According to experimental characterization on 45nm CMOS, one 32-bit SRAM access costs 5pJ of energy while 32-bit floating-point and integer MAC operations cost a total of 4.6pJ and 3.2pJ, respectively [29].

VI. PROPOSED COW BEHAVIOR DISTRIBUTION ESTIMATING NEURAL NETWORK (TINYCOWNET)

Recurrent neural networks are known to learn time-based relationships of generative processes and are prevalent in applications of natural language processing, machine translation, and sequence modeling [59]. In hardware, over 70 implementations on ASIC, FPGA, or GPU/CPU have been demonstrated in Ref. [60], but only two were applied to action recognition using video footage [61], [62]. To our knowledge, no RNN for animal behavior monitoring on hardware has been described.

TABLE 4. Hyperparameters defined for the search space of the recurrent layers, i.e., the layer width n, the total number of layers N, the timesteps T_S and the cell architecture type k.

Hyperparameter	Range	Description
n	$R_n = 3, 4 \dots 250$	Layer width
N	$R_N = 1, 2 \dots 4$	Number of layers
T_s	$R_{L_s} = 2, 3 \dots 300$	Timesteps
k	$R_k = \{\text{sRL,GRU,LSTM}\}$	Recurrent layer type

The recurrent layers described in Section V are used to optimize network architecture towards estimating cow behavior distribution in a future low-power hardware implementation. This network is composed of one to four recurrent layers and a linear layer that outputs a behavior distribution vector as in Eq. (3). To find an optimal network in terms of accuracy and model size, the parameters, including those of Table 3 are optimized with neural architecture search (explained in Section VII). Fig. 8 shows the proposed parameterized neural network, TinyCowNet. Each yellow block represents the architecture k (GRU, LSTM or sRL) with a width n_f, trainable weights $\mathbf{U}_l$, $\mathbf{W}_l$ and $\mathbf{b}_l$, and activation functions $\tanh(x)$ and $\sigma(x)$. This n_f is the layer width n divided by layer count N to prevent the creation of excessively large models, i.e.,

$$n_f = \left\lfloor \frac{n}{N} \right\rfloor \tag{21}$$

TABLE 5. Low-power programmable devices potentially viable as targets for TinyCowNet implementation. The Lattice UL640 and Microsemi AGLN250 FPGA are ultra low-power devices suitable for the smallest networks. The STMicroelectronics MCU and Lattice UP5K are low-power FPGAs better suited for medium-sized models.

Manufacturer / Family	Device	Memory (kB)	Clock speed (MHz)	Sleep power (μW)	Area (mm^2)	Price USD ($)
Lattice / iCE40 Ultra Lite	[A] UL640 FPGA [63]	7.1	48	42	2	1.82
Microsemi / IGLOO nano	[B] AGLN250 FPGA [64]	4.7	250	10	196	16.43
STMicroelectronics / -	[C] STM32L071C8 MCU [65]	84	32	37	49	4.60
Lattice / iCE40 Ultra Plus	[D] UP5K FPGA [66]	143	48	90	6.4	5.96

where the fraction is floored. For each time step, tri-axial accelerometer data are submitted to the network until timestep T_s. Furthermore, each layer recurs the previous hidden state vector $\mathbf{h}_{t-1}$, that is also forwarded to the next layer. At timestep T_s, the final output of the deepest recurrent layer is linearly mapped to output a cow behavior distribution vector using the ReLU activation function, i.e., $\mathrm{ReLU}(x) = \max(0, x)$. In the case of an LSTM network, there is an additional recurrent output, cell state vector $\mathbf{c}_t$, which is not forwarded to the deeper layers.

Table 4 shows all parameter ranges of the search space subject to optimization. Because of composite generative processes in nature, it is expected that multi-layer networks will outperform wide single-layer networks [67]. As regards the layer width n, the minimum number (3) is determined by the need to avoid a trivial response, whereas the maximum number (250) reflects the maximum available memory in the devices under consideration. As regards the number of layers N, the minimum number (1) is self-evident, whereas the maximum number (4) was determined based on existing literature indicating that deeper networks are difficult to train when considering the range of timesteps [55]. As regards the timesteps T_s, the minimum number (2) is self-evident, and the maximum number was set to 300 since this is an intermediate level between the limits previously observed: sRLs become increasingly difficult to train for $T_s > 100$, whereas LSTMs and GRUs become increasingly difficult to train for $T_s > 500$ [55].

A. TARGET NETWORKS

Table 5 describes three low-power FPGAs and one low-power MCU with their total memory, clock speed, board area and purchase price. These are indicated purely as representative examples. Each FPGA has enough resources for the implementation of several multiply-accumulate units and control logic. Furthermore, TensorFlow Lite [45] for MCUs uses only 16kB of runtime memory, therefore we assume a remaining memory of 68kB. The average price of all these devices is 7.2$. In order to allow for future implementation of TinyCowNets, we have set the target of building networks which have a memory footprint below the smallest device. Furthermore, to be competitive with other works, we aimed to maintain an accuracy $> 93\%$.

VII. FINDING THE TARGET NETWORKS

Recent advances in neural network design have yielded computer-designed neural network architectures that outperform expert-designed neural networks in computer vision tasks [68], [69]. This field of research, neural architecture search, was first proposed in Ref. [70] using reinforcement learning. Since then, different methods using genetic algorithms and gradient descent have been applied to reduce search time and further increase model accuracy [71], [72]. A comparison of these methods for cow behavior estimation and recurrent neural networks is outside of the scope of this work. We decided to apply random search to determine the recurrent layers in Section VI and Table 4 because of its straightforward implementation and because defining suitable heuristics for more advanced methods was beyond scope. For example, determining a suitable fitness function and chromosome structure for efficient optimization using genetic algorithms requires careful consideration of a multitude of aspects, to make sure the evolutionary aspect meaningfully improves the process over a pure random search. Random search generates a model according to randomly assigned values within the allowed ranges of parameters, and as such it is the most parsimonious possibility with respect to the assumptions that have to be introduced. It was shown in Ref. [73] and argued by Ref. [74] that current neural architecture search methods do not significantly outperform this brute-force search method.

A. RANDOM SEARCH

Each point in the search space as denoted in Table 4 is defined as

$$\theta = \{n_f, N, T_s, k\} \tag{22}$$

where n_f is the floored layer width from Eq. (21), N is the layer count, T_s is the amount of timesteps and k is the recurrent layer type. There are 295,412 unique possible combinations with the indicated ranges, making a grid search unfeasible. Therefore, we set the hyperparameters to be uniformly-distributed discrete random variables so that each point in the parameter range has an equal probability of being selected. A total of 445 random searches were performed.

Fig. 9 shows the method of randomly selecting hyperparameters and training TinyCowNets. For each iteration,

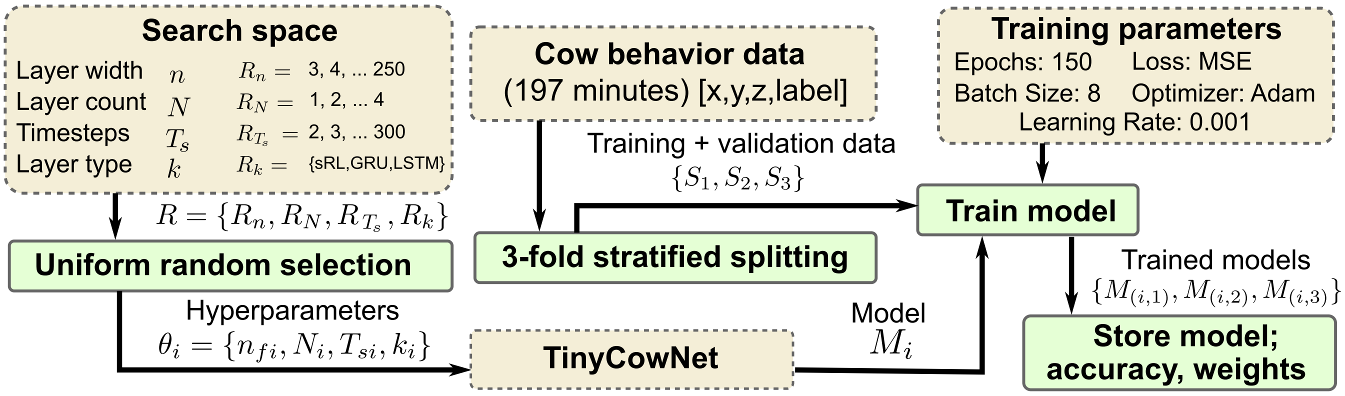

Search space
Layer width n $R_n =$ 3, 4, ... 250
Layer count N $R_N =$ 1, 2, ... 4
Timesteps T_s $R_{T_s} =$ 2, 3, ... 300
Layer type k $R_k =$ {sRL,GRU,LSTM}
$R = \{R_n, R_N, R_{T_s}, R_k\}$
Uniform random selection
Hyperparameters
$\theta_i = \{n_{fi}, N_i, T_{si}, k_i\}$
Cow behavior data
(197 minutes) [x,y,z,label]
Training + validation data
$\{S_1, S_2, S_3\}$
3-fold stratified splitting
TinyCowNet
Model
M_i
Training parameters
Epochs: 150 Loss: MSE
Batch Size: 8 Optimizer: Adam
Learning Rate: 0.001
Train model
Trained models
$\{M_{(i,1)}, M_{(i,2)}, M_{(i,3)}\}$
Store model; accuracy, weights

FIGURE 9. Diagram showing a single iteration i of a random search applied to four hyperparameters of TinyCowNet (Fig. 8). These hyperparameters represent layer width n, layer count N, layer type k, which have influence over the complexity of the network and timesteps T_s. For each iteration, a set of hyper parameters θ in the uniformly distributed range R is randomly assigned to TinyCowNet, in this case M_i. Hereafter, a three-fold stratified split training-validation dataset (Fig. 10) is used to train the model with Mean Squared Error (MSE) as loss function for 150 epochs. Finally the resulting three model accuracies and weights are stored for later resource-optimized candidate selection.

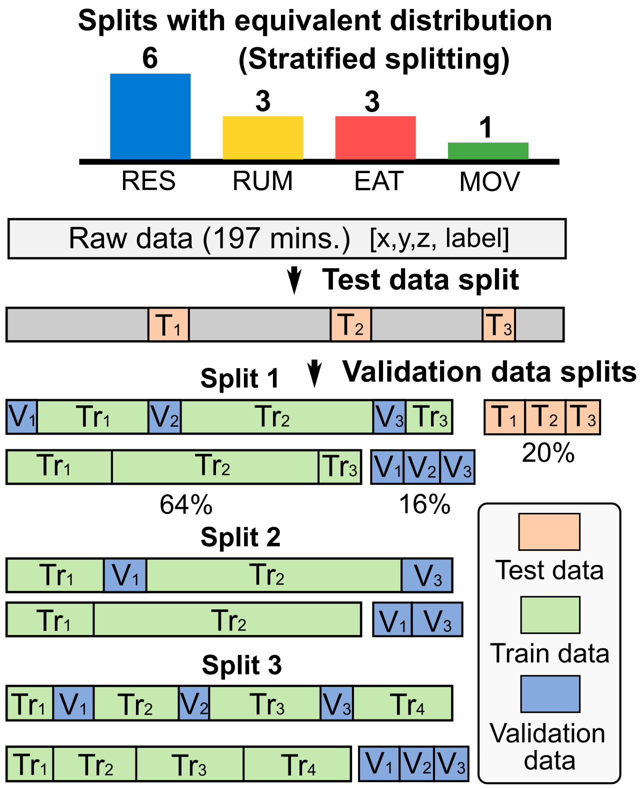


FIGURE 10. Stratified splitting of the imbalanced cow behavior dataset.

one unique subset of the search space denoted as $\theta_i = \{n_{fi}, N_i, T_{si}, k_i\}$, is randomly assigned from the uniform range R. The parametrized neural network, TinyCowNet (Section VI), is adapted with this subset and trained on a supercomputer for 150 epochs and three different splits of the cow behavior dataset using Adam optimizer and a batch size of 8. After training, the model characteristics, weights, subspace θ_i, and accuracy are stored, and a new iteration is initiated.

B. TRAINING CONDITIONS

Fig. 10 shows the data splitting methodology. First, the data are split into 80% and 20%, of which the latter are used as test dataset. The remaining data is then split into 16% and 64% representing the validation and training datasets, repeated over three folds. Each fold's data split is stratified, i.e., the training, validation, and testing dataset have similar distribution of labels as the original dataset (distribution as in Fig. 5, i.e., 6:3:3:1). This validation dataset is used to rate each candidate model by its accuracy. For each training iteration, the randomly assigned model is trained and validated with different data, resulting in three different models. Finally, the performance is averaged and stored for both the validation and the test datasets. Stratified splitting ensures that the datasets have comparable distributions and has been used to obtain reliable performance of cow behavior classification [44]. Each of these three data splits represents the original imbalanced dataset with possible overlapping, but performance on each fold is still considered as independent [75].

The evaluated metric during training is the standard Mean Squared Error (MSE) for a four behavior distribution vector which is described by

$$\mathrm{MSE} = \frac{1}{4}\sum_{i=1}^{4}(a_i - \hat{a}_i)^2 \tag{23}$$

where a_i is the prevalence of the i-th behavior and $\hat{a}_i$ is the estimated prevalence from the selected TinyCowNet. This MSE punishes the algorithm more heavily for error than a L_1 norm vector, decreasing training time.

C. CANDIDATE EVALUATION: ACCURACY, MACS AND MEMORY

After training, the Mean Absolute Error (MAE) is used to score each architecture over the validation dataset. This MAE represents the absolute difference between each estimated $\hat{a}_i$

and real prevalence a_i averaged over all behaviors, i.e.,

$$\text{MAE} = \frac{1}{4}\sum_{i=1}^{4}|a_i - \hat{a}_i| \tag{24}$$

Compared to the MSE, the MAE lends itself to a more intuitive interpretation, in light of the fact that the network output is a distribution of prevalence. On the other hand, from a different perspective, the accuracy of the algorithm in identifying the most prevalent behavior is obtained by considering expectation, i.e.,

$$\text{Accuracy} = \frac{1}{M}\sum_{i=1}^{M}E[\mathbf{y}(i), \hat{\mathbf{y}}(i)] \tag{25}$$

where M is the length of the dataset and the operator $E[\mathbf{y}(i), \hat{\mathbf{y}}(i)] = 1$ if the largest element matches between the two vector, and $E[\mathbf{y}(i), \hat{\mathbf{y}}(i)] = 0$ otherwise. Next to the MAE and accuracy, we employ the Matthews Correlation Coefficient (MCC) that ranges from -1 to 1, as it is a standard metric used on imbalanced datasets [33], [76]. Besides this, we have observed that the majority of the distribution vectors processed from the dataset contain a single behavior. However, TinyCowNet does not present a binary output (1 or 0) as the final layer is linear and non-ideal. As such, we use the Receiver Operating Characteristic (ROC) curves and calculate the Area Under Curve (AOC) instead of setting an arbitrary threshold to classify each behavior from the output vector independently. This a standard statistical procedure used for imbalanced datasets [76], [77].

Other than performance metrics, the memory occupation and number of MAC operations of each model are determined. The total number of MACs that a TinyCowNet uses, O_k, for each layer type k including linear layer, layer count N and timesteps T_s is given by

$$O_{\text{sRL}} = T_s(n_f^2(2N-1) + 3n_f) \tag{26}$$
$$O_{\text{LSTM}} = T_s(n_f^2(8N-4) + n_f(3N+12)) \tag{27}$$
$$O_{\text{GRU}} = T_s(n_f^2(6N-3) + n_f(3N+9)) \tag{28}$$
$$O_{ll} = 4n_f \tag{29}$$

where the contribution of the deeper layers is more significant. This is because the recurrent vector output, $\mathbf{h}_t$, from all layers has a size of n_f, changing the input weight matrix size of the deeper layers to $\mathbf{W} = n_f \times n_f$. Furthermore, the T_s term is necessary to determine the total amount of operations as the networks recur until the final timestep. The number of weights required to store the model for each recurrent layer type k and output layer ll is given by

$$P_{\text{sRL}} = n_f^2(2N-1) + n_f(N+3) \tag{30}$$
$$P_{\text{LSTM}} = n_f^2(8N-4) + n_f(4N+12) \tag{31}$$
$$P_{\text{GRU}} = n_f^2(6N-3) + n_f(6N+9) \tag{32}$$
$$P_{ll} = 4n_f + 4 \tag{33}$$

where the GRU has two bias weights instead of one. Overall, the number of MACs in sRL TinyCowNet is between 3-4 times lower than that of LSTM and GRU.

VIII. POST-TRAINING QUANTIZATION

TensorFlow models are trained and inferred using 32-bit floating-point representation. Such models on hardware use large amounts of memory and high-complexity arithmetic operations. By contrast, the industry standard in low-power digital processing hardware is using 8-bit fixed point numbers. Quantization of neural networks is generally split into two methods: pre- and post-training quantization. While the former has led to better results than the latter [78], quantization during training adds another dimension to the optimization problem, further complicating decisions on optimal architecture, beyond the scope of this work. Therefore, we decided to create an ad-hoc method of post-training, integer-only quantization aiming to minimize energy consumption and memory occupancy. A similar, although not fully equivalent, approach has been described in a recent preprint report [79]. The novelty in this work is the activation function scaling and dynamic bit-width optimization.

A. FULL-INTEGER QUANTIZATION FOR RNNS

Neural networks with integer-only arithmetic use lower-power modules in hardware, and show a lower latency for similar accuracy compared to floating-point models [80]. This fixed-bit-width integer-arithmetic-only quantization scheme from Ref. [80] is used in TensorFlow Lite, but limited to convolutional neural networks. Here, we adapt it to RNNs. An analysis of different quantization methods is out of scope and will be systematically performed in a future study. For now, we considered variable bit-widths for different parts of the RNN towards power-accuracy optimization with an in-depth analysis but without comparing different possibilities. This Section describes the proposed quantization scheme.

Consider the mapping of a real number r to an integer q with scale S and zeropoint Z

$$r = S(q - Z) \tag{34}$$

the scale is then defined as

$$S = \frac{a}{2^l - 1} \tag{35}$$

where a is the range, and l is the bit-width. In the case of a static matrix such as the trainable weights of a neural network, i.e., TinyCowNet, we assign this range to

$$a_w = |\max(\mathbf{W})| + |\min(\mathbf{W})| \tag{36}$$

as $\mathbf{W}$ is one of the matrices to be quantized. The zeropoint offsets any negative numbers avoiding the use of two's complement representation, further reducing logic occupancy. We do this by relating the zeropoint to the minimum number in the weight matrix as in

$$Z_w = \left\lfloor \frac{-\min(\mathbf{W})}{S_w} \right\rceil \tag{37}$$

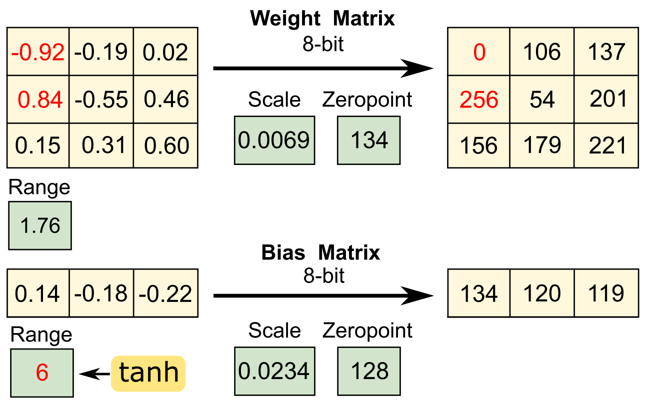


FIGURE 11. Example of full-integer quantization for RNNs with a weight matrix and bias matrix for which the range is defined by the activation function at 8-bit bit-width.

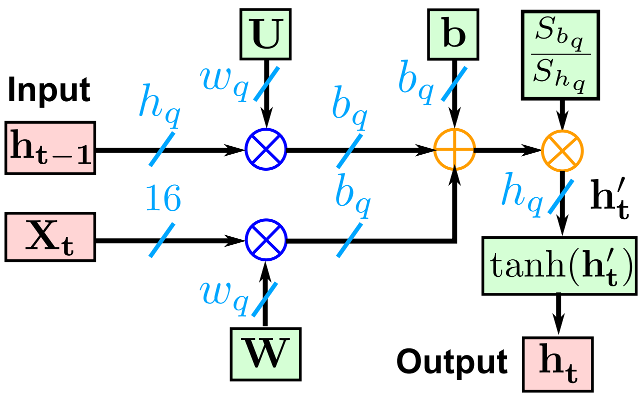


FIGURE 12. Overview of the simple recurrent layer with integer quantization. The red and green blocks represent the input/output and stored weights, respectively. The light blue stripes indicate the bit-width of the buses, output bit-width h_q, weight bit-width w_q and bias bit-width b_q. The dark blue multipliers are equivalent to the matrix multiplication as described in Eq. (41). The hyperbolic tangent function is implemented as a look-up where $\mathbf{h'_t}$ are the indices. The orange blocks are element-wise operation blocks where the output of the addition operation is rescaled to h_q bits.

where S_w is the scale of quantized $\mathbf{W}$, and $\lfloor x \rceil$ denotes rounding. Each weight of the matrix is subsequently quantized by

$$q_w^{(i,j)} = \left\lfloor \frac{r_w^{(i,j)}}{S_w} + Z_w \right\rceil \tag{38}$$

where $r_w^{(i,j)}$ is the floating point number and $q_w^{(i,j)}$ is the quantized number.

In the case of bias $\mathbf{b}$, terms $\mathbf{Uh}_{t-1}$ and $\mathbf{Wx}_t$, the scales and zeropoints have to be equivalent for straightforward element-wise addition. Therefore, the domains of the activation functions are considered to set the quantization range α as in Eq. (35). We assume the sigmoid has a range value, $a_\sigma = 12$, and the hyperbolic tangent has a range value of 6, $a_{\tanh} = 6$, considering their asymptotes. The above described terms use the same quantization scheme as in Eq. (38), but their scale S_3 and zeropoint Z_3 are determined by the activation function. This is a_σ, for LSTM and GRU networks or $a_{\tanh}$ for the sRL network. Fig. 11 shows an example of quantizing a 3×3 matrix and a 1×3 bias matrix with the methods described above.

A matrix multiplication involves two differently-quantized numbers for each entry with different scale and zeropoint. In that case multiplication for each entry with the quantization scheme is described as

$$q_3 = \frac{S_1 S_2}{S_3}(q_1 - Z_1)(q_2 - Z_2) + Z_3 \tag{39}$$

where S_3 and Z_3 are the scale and zeropoint of the resulting number. The results of matrix multiplications with $\mathbf{U}$ and $\mathbf{W}$ are added to the bias matrix $\mathbf{b}$. This means that S_3 and Z_3 have the range of a_σ or $a_{\tanh}$. In turn, S_1 and Z_1 are defined by $\mathbf{U}$ and $\mathbf{W}$. Similarly, S_2 and Z_2 are set to the ranges of the accelerometer data ($\pm$ 2g) and the recurrent output vector with range $a_{\tanh}$. Multiplication of an $I \times M$ and an $J \times M$ matrix are performed according to

$$q_3^{(i,m)} = \frac{S_1 S_2}{S_3} \sum_{j=1}^{M} \left[(q_1^{(i,j)} - Z_1)(q_2^{(j,m)} - Z_2) \right] + Z_3 \tag{40}$$

TABLE 6. Quantization parameters defining a quantization search space to be used on three TinyCowNets selected from the random search. While the weight bit-width w_q and h_q are selected arbitrarily and independent, b_q is dependent on the former because of the relationship with matrix multiplications in the recurrent layers.

Param.	Range	Description
w_q	6, 7 ... 10 12,14 ... 16	Weight bit-width
h_q	6, 7 ... 10 12,14 ... 16	Output / Activation bit-width
b_q	$3w_q$	Bias bit-width

which can be rewritten as

$$q_3^{(i,m)} = \frac{S_1 S_2}{S_3} \left[M Z_1 Z_2 - Z_2 \sum_{j=1}^{M} q_1^{(i,j)} - Z_1 \sum_{j=1}^{M} q_2^{(j,m)} + \sum_{j=1}^{M} q_1^{(i,j)} q_2^{(j,m)} \right] + Z_3 \tag{41}$$

where the first two terms in the brackets are known in advance since each entry in weight matrix $q_1^{(i,j)}$ is available [80]. Therefore, these can be stored to speed up the multiplication. However, for large bit-widths, this second term takes up a considerable amount of memory. For this reason, we have decided to store no other than the zeropoints, scales, and column size M. These equations are similar to Ref. [80].

B. OPTIMAL QUANTIZATION WITH GRID SEARCH

Fig. 12 shows the above-described quantization scheme applied to a simple recurrent layer considering the weight bit-width w_q, bias bit-width b_q and output/activation bit-width h_q. These parameters replace l in Eq. (35) for each weight matrix. Matrices $\mathbf{U}$ and $\mathbf{W}$ are quantized with w_q, bias matrix $\mathbf{b}$ with b_q and the recurrent output vectors $\mathbf{h}_t$ and $\mathbf{c}_t$ are quantized with h_q. The S_{b_q}/S_{h_q} term rescales the added matrices to h_q for activation function tanh(x). This hyperbolic tangent and the sigmoid use, possibly with interpolation, a Look-Up Table (LUT) with size 2^{h_q}. As such, rescaled $\mathbf{h}'_t$ is used as an index of this LUT. The size and precision of this table are other important factors for performance and memory. The architecture search space already includes hundreds of thousands of possibilities. Therefore, we only consider the three best-trained candidates from the random search for the quantization scheme to limit the compute required for dynamic quantization.

Table 6 shows the quantization search space with b_q, w_q and h_q. The bias, $\mathbf{b}$, needs sufficient precision as it is involved with the element-wise addition with the two other terms in Eq. (9). Therefore, we select this precision as $b_q = 3w_q$ since the number of weights, especially for the GRU and LSTM layers, is large. This b_q has the smallest influence over the memory occupancy, because it is only involved in the addition of vectors with size n_f (layer width) and rescaled to h_q. Nonetheless, the inclusion of w_q and h_q with ranges of 13 steps in the random search, would increase the involved search space by 169×. In other words, performing quantization before random search drastically enlarges the required computation load as the unique combinations increases from 300 thousand to 50 million. In addition, as quantization injects noise into the dataset, the training process would be further complicated. Therefore, we use a separate grid search to find bit-width combinations that yield a minimum loss of accuracy after training. We define this loss as the absolute distribution error, the absolute difference between the quantized model and TensorFlow model output vectors using the test dataset. Next to this, we consider the minimum memory and energy consumption to balance resources and performance. Starting from the three best trained candidates given by the random search, a total of 507 quantized models are generated and tested on the test dataset.

C. MEMORY ESTIMATION FOR THE QUANTIZED MODELS

Eqs. (30)-(33) are transformed to include the two parameters w_q and h_q, and determine the memory in bits for all layers, i.e.,

$$P_{\text{qsRL}} = w_q(n_f^2(2N-1)+3n_f)+b_q n_f N \tag{42}$$

$$P_{\text{qLSTM}} = w_q(n_f^2(8N-4)+12n_f)+4b_q n_f N \tag{43}$$

$$P_{\text{qGRU}} = w_q(n_f^2(6N-3)+9n_f)+6b_q n_f N \tag{44}$$

$$P_{\text{qll}} = 4w_q n_f + 4b_q \tag{45}$$

w_q and b_q are 32-bit floating-point for vanilla TensorFlow models [81]. Furthermore, the memory required by the hyperbolic tangent and sigmoid LUTs for layer type k is

$$P_{\text{qact},k} = \begin{cases} 2^{hq-1}, & \text{if } k = \text{sRL} \\ 2\cdot 2^{h_q-1}, & \text{if } k = \text{LSTM} \\ 2\cdot 2^{h_q-1}, & \text{if } k = \text{GRU} \end{cases} \tag{46}$$

where h_q is the output bit-width and the 2^{-1} term indicates half LUT size since the functions are both odd. In the case of a 32-bit implementation, these functions would use up the majority of resources. However, with small bit-widths, the LUTs are few and only a single SRAM access per activation is required.

D. POWER AND ENERGY ESTIMATION

Starting from the stated 32-bit SRAM access energy of 5pJ on 45nm CMOS [29], we can as a first approximation assume that a bit-by-bit operation can be linearly scaled for a multiplier-adder and SRAM considering the defining properties of logic circuits [82]. Therefore, we estimate the energy consumption of all SRAM accesses for a single inference to be

$$E_{\text{DSRAM}} = \frac{5(T_s(P_{qk}+C_{\text{qact},k})+P_{qll})}{32}\ \text{pJ} \tag{47}$$

where the recurrent layer memory is accessed for each timestep until T_s and $C_{\text{qact},k}$ is the number of activation function bit look-ups. This is represented by the following equation

$$C_{\text{qact},k} = \begin{cases} h_q n_f, & \text{if } k = \text{sRL} \\ h_q 5n_f, & \text{if } k = \text{LSTM} \\ h_q 3n_f, & \text{if } k = \text{GRU} \end{cases} \tag{48}$$

Differently from SRAMs, we estimate the dynamic energy consumption of MAC operations by transforming the experimentally-verified 32-bit integer MAC energy consumption of 3.2 pJ [29]. This transformation similarly scales the experimentally-verified energy as a multiplier with a single bit-width determined by either h_q or w_q, wherein

$$E_{\text{MAC}} = \frac{3.2(O_k+O_{ll})\max(w_q,h_q)}{32}\ \text{pJ} \tag{49}$$

which would offer flexibility on hardware. This energy per MAC and SRAM effectively provides a lower bound of the dynamic energy requirement. Regarding the static energy consumption, as a first approximation we focus on the SRAM, neglecting everything else, on the assumption that its contribution is dominant. We therefore consider a 45nm CMOS 1kb SRAM that has been experimentally determined to consume 3.7nW of static power [30] and calculate the required amount as

$$m_{\text{SRAM}} = \left\lceil \frac{(P_{qk}+P_{qll}+P_{qact,k})}{1024} \right\rceil \tag{50}$$

to store all bits of the model in memory. The total static energy consumption for maintaining the weights and activation functions in the multiple SRAMs for a single inference is then shown to be

$$E_{\text{SSRAM}} = 3700\frac{m_{\text{SRAM}}T_s}{f_{acc}}\ \text{pJ} \tag{51}$$

where f_{acc} is the accelerometer sampling frequency. Assuming that $P_{qk} \gg (P_{qll}+P_{qact,k})$ in case of large networks

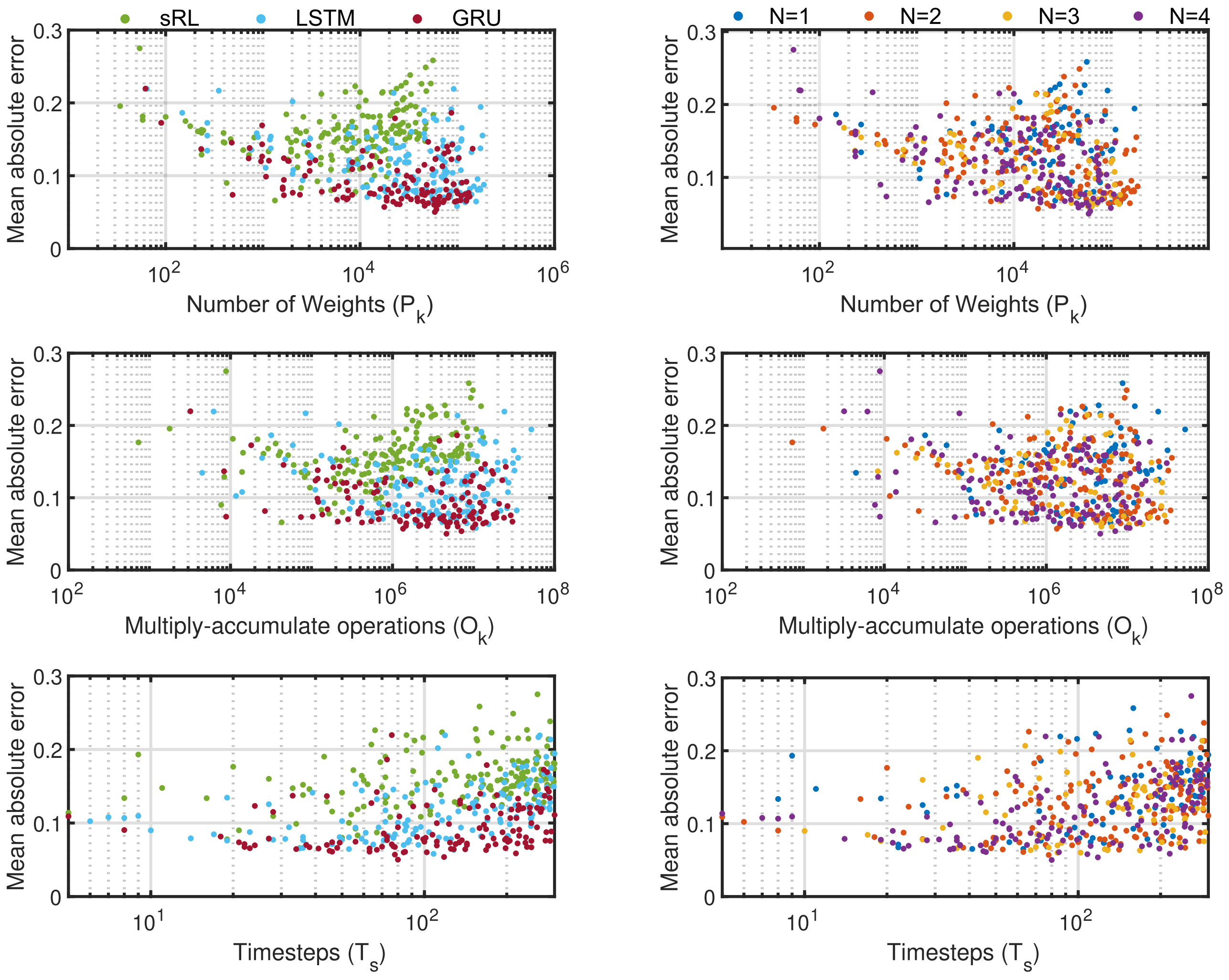


FIGURE 13. Scatterplots of the 445 TinyCowNets (parametrized RNNs) found using random search. In the left column, the color denotes layer type k. In the right column, the colors denote number of layers N. Furthermore, the top, middle, and bottom row show the mean absolute error vs. the number of weights P_k, the MAC operation count O_k, and timesteps T_s.

and sufficiently small h_q, we can estimate the relationship between the energy required to store the weights and the total SRAM access energy, i.e.,

$$E_{\mathrm{SSRAM}} \approx \frac{23}{f_{acc}} E_{\mathrm{DSRAM}} \tag{52}$$

showing that the static energy will dominate for accelerometer frequencies under 23 Hz. Importantly, this indicates that the power consumption cannot be arbitrarily reduced by lowering sampling rate. The total energy for a single inference is thus

$$E_{\mathrm{inf}} = E_{\mathrm{MAC}} + E_{\mathrm{SSRAM}} + E_{\mathrm{DSRAM}} \ \ \mathrm{pJ} \tag{53}$$

from which the power consumption in can be calculated as

$$\mathrm{Pow}_m = \frac{f_{\mathrm{acc}} E_{\mathrm{inf}}}{T_s} \ \ \mathrm{pW} \tag{54}$$

where the f_{acc} is the sampling frequency of the accelerometer, 25Hz in this work. It is important to underline that this estimation approach only considers a lower bound, and in practice factors such as the static current in the MAC unit, other logic etc. are likely to have a substantial impact. Nevertheless, this approach allows a meaningful comparison and ranking of networks, which suffices for the present work's purpose.

IX. RESULTS OF TINYCOWNET WITH RANDOM SEARCH AND QUANTIZATION

For each of the 445 iterations of random search, three TinyCowNets were trained for 150 epochs subject to stratified splitting. Overall, this process took a compute time of 859 hours on a Xeon E5-2680 v4 2.4GHz core, averaging 116 minutes per iteration [83].

A. WEIGHT, MAC AND TIMESTEPS ANALYSIS

Fig. 13 shows the Mean Absolute Error (MAE) of 445 random search iterations in six scatter plots. The number of weights, P_k, is strongly inversely related to the MAE, which

TABLE 7. Average mean absolute error (MAE) of the best-performing 10% of found models per decade vs. the model weights P_k and MACs per inference O_k (10^1-10^8) with random search. This is considered with the hyperparameters for recurrent layer type k and layer count N as well as the result of the entire random search results regardless of hyperparameter (All (average)).

	Mean Absolute Error (MAE)									
Hyperparameter	**Number of Weights (P_k)**					**Number of MACs (O_k)**				
Layer type (k)	10^1	10^2	10^3	10^4	10^5	10^3	10^4	10^5	10^6	10^7
sRL	0.1765	0.1093	0.0881	0.01135	-	0.0900	0.0899	0.08658	0.1361	0.1503
LSTM	0.2192	0.1345	0.0934	0.0704	0.0648	0.1345	0.1024	0.0733	0.0702	0.0705
GRU	0.1726	0.0738	0.0678	0.0577	0.0658	0.0738	0.0733	0.0676	0.0573	0.0644
Layer count (N)										
1	-	0.1345	0.0783	0.0711	0.0653	0.1345	0.1339	0.0783	0.06758	0.1091
2	0.1723	0.1352	0.0767	0.0668	0.0659	0.1956	0.0816	0.0789	0.0657	0.0678
3	-	0.1197	0.0782	0.0618	0.0635	0.1366	0.1286	0.0743	0.0622	0.0634
4	0.2192	0.0819	0.0696	0.0594	0.0768	0.0738	0.0696	0.0695	0.0588	0.0620
All (average)	0.1726	0.1019	0.0728	0.0627	0.0649	0.0738	0.0736	0.0722	0.0625	0.0647

TABLE 8. Number of weights, total MACs per inference, timesteps, recurrent layer count and recurrent layer type of the best implementable 32-bit floating-point TinyCowNets for each of the four devices considered (Tables 5, 9). In addition, the independent accuracy for each of four cow behaviors, i.e., resting (RES), rumination (RUM), moving (MOV) and eating (EAT) on the test dataset is described as determined by optimal ROC threshold.

	Model details					Accuracy per behavior on test dataset			
Best model per device	Number of weights (P_k)	Total MACs (O_k)	Timesteps (T_s)	Layers (N)	Layer type (k)	RES	RUM	MOV	EAT
[D]	17,720	1,348,480	78	2	GRU	96.71%	99.72%	99.57%	96.90%
[C],[A]	1330	42,822	35	4	sRL	96.16%	98.87%	98.67%	96.37%
[B]	488	8836	21	4	GRU	96.75%	98.67%	99.10%	97.24%

TABLE 9. Maximum number of 32-bit weights and MACs per timestep when assuming a single multiplier unit on three low power FPGAs and one MCU as described in Table 5. In light of these resources, the best MAE on the validation dataset of an implementable TinyCowNet (described in Table 8) from the random search results is indicated.

Device	Max MACs per timestep	Max weights (32-bit)	Best possible MAE
UP5K FPGA [D]	1,920,000	35,750	0.0571
STM32L07 MCU [C]	1,280,000	17,000	0.0660
AGLN250 FPGA [B]	10,000,000	1,175	0.0738
UL640 FPGA [A]	1,920,000	1,775	0.0660

decreases near-logarithmically over the first two decades (10^1-10^3). Furthermore, four- and two-layer models seem to be most MAC-efficient. In addition, it seems that sRL networks have increasing error for MACs and number of weights above 10^6 and 10^4, confirming the vanishing-exploding gradient problem that makes large sRL models harder to train.

Of the in total 445 models, 148 have an MAE equal to or lower than 0.1. Within this error bound, 85 or 59% are of the layer type GRU followed by 55 LSTM and only 8 sRL models. Similarly, 48 and 56 of all the models within this region have two or four layers, representing the majority. The bottom row of Fig. 13 shows the best models to maintain similar accuracy with changing T_s. In other words, there is a weak relationship between the timesteps and the MAE.

We statistically analyzed the models within each decade's 10% lowest average MAE in terms of number of weights and MACs. Table 7 describes the results of this analysis. Supporting our earlier claim, the MAE decreases by 0.0707 for the initial weight decade, the highest drop within all decades. Furthermore, the MAE slightly increases over the last decade, indicating a possible lack of training epochs. The best performing models are GRUs at the lowest average MAE of 0.0557 and 0.0573 within 10^4 weights and 10^6 MACs. Also, GRU represents the best models at low decades, i.e., an average MAE of 0.0738 within 10^2 weights and 10^3 MACs. The second block of Table 7 shows the layer count vs. MAE. The best-performing models tend to be those with four layers. In other words, the best models in terms of resource and accuracy are four-layer GRUs. This is in line with Ref. [54].

B. 32-BIT MODELS ON LOW-POWER MCU AND FPGA

Table 9 shows the maximum number of weights and MACs for each of the devices in Table 5. In addition, the MAE of the best implementable model found in the random search

TABLE 10. Three selected TinyCowNets that are smallest or have best performance for a validation dataset accuracy above 93%. In addition to amount of weights, MACs, timesteps and layers, the performance on the test dataset for each behavior is described from the optimal ROC threshold.

	Model details				Accuracy per behavior on test dataset			
Models <0.03 MAE	N. of Weights (P_k)	Total MACs (O_k)	Timesteps (T_s)	Layers (N)	RES	RUM	MOV	EAT
Best/Smallest sRL (bsRL)	1,330	42,822	35	4	92.81%	98.5%	94.5%	98.1%
Best GRU (bGRU)	58,712	4,630,288	80	4	94.9%	99.9%	96.7%	98.90%
Smallest GRU (sGRU)	5,788	210,768	38	2	92.40%	98.5%	95.2%	98.1%

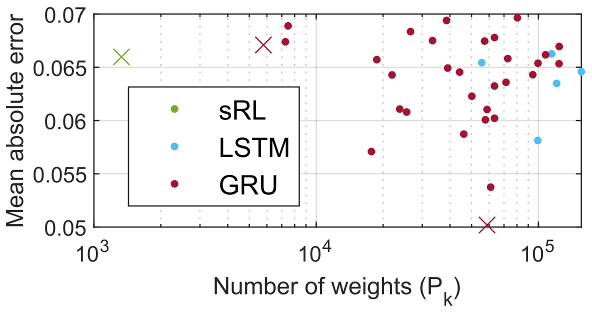


FIGURE 14. Scatterplot of all TinyCowNets from the random search with an MAE below 0.07. The three models plotted with a cross indicate the candidates subject to full-integer quantization and ASIC resource estimation.

is described. The maximum MACs are calculated by dividing the max operating frequency with the accelerometer frequency at 25Hz. In other words, for FPGA, this order-of-magnitude estimate simply assumes that at least one MAC unit fits the device. This MAC count should be at least more than the term factorized by T_s in Eqs. (26)-(29). We set this limit to ensure that the network can be implemented with a single multiplier-accumulator module. Table 8 shows the model weights, MACs, timesteps, and the best accuracy determined from the ROC curve independently for each behavior with the test dataset. Each of these models is the best model that could be implemented on any of the four devices. The accuracy as in Eq. (25) is 95.7% for model [D], 95.2% for model [C], [A], and 95.3% for model [B].

C. MINIMUM MEMORY AND QUANTIZED ARITHMETIC FOR ASIC IMPLEMENTATION

The above models use 32-bit floating-point arithmetic, which requires complex logic and is wasteful of memory because of redundant precision. Fig. 14 shows a scatterplot of all models below an MAE threshold of 0.07 with number of weights P_k. The three selected candidates subject to quantization and 45nm CMOS resource estimation are indicated with a cross and described in Table 10. The sRL shows an MAE of 0.0669, and the sGRU and bGRU have an MAE of 0.0671 and 0.0502. As these are the best GRU, best sRL, and smallest GRU, we shall call these models bGRU, bsRL, and sGRU. The bsRL is the same model for implementation on MCU [C] and FPGA [A]. Below 0.07 MAE, most models are GRU with many MACs, making these impossible to implement

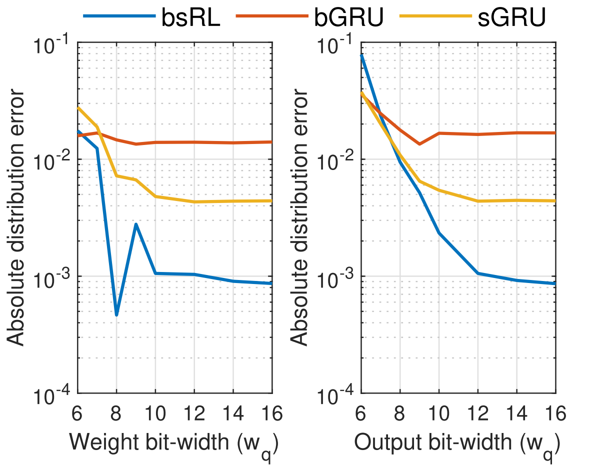


FIGURE 15. Absolute distribution error vs weight bit-width and output bit-width, which is the error between the full-integer and floating point counterparts for the bsRL, bGRU and sGRU models, described in Table 10.

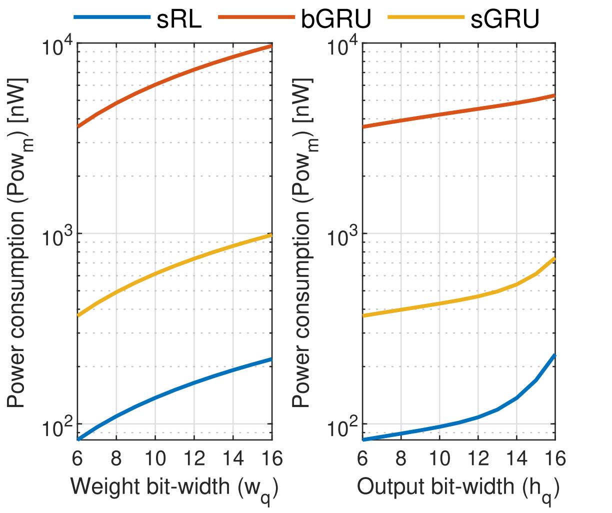


FIGURE 16. Estimated power consumption for eight weight and output bit-widths of full-integer quantization applied to the bsRL, bGRU and sGRU models.

on low-power devices with 32-bit floating-point precision. As we shall show, our dynamic full-integer quantization allows all models to be implemented on low-power FPGAs and MCUs.

Fig. 15 shows the Absolute Distribution Error (ADE) vs. the weight and output bit-width for integer-quantized bsRL,

TABLE 11. Characteristics of the selected best-performing quantized models, i.e., energy, power, memory and accuracy vs. weight bitwidth w_q and output bit-width h_q.

	Quantization bit-widths		Hypothetical 45nm CMOS resource estimation			
Quantized models	Weight (w_q)	Output (h_q)	Inference energy (E_{inf}) [nJ]	Power (Pow_m) [nW]	Memory [kB]	**Accuracy on Test Dataset**
bsRL	10	12	216	154	2.043	95.2%
bGRU	9	9	17417	5443	67.47	97.4%
sGRU	10	12	1035	660	8.351	95.6%

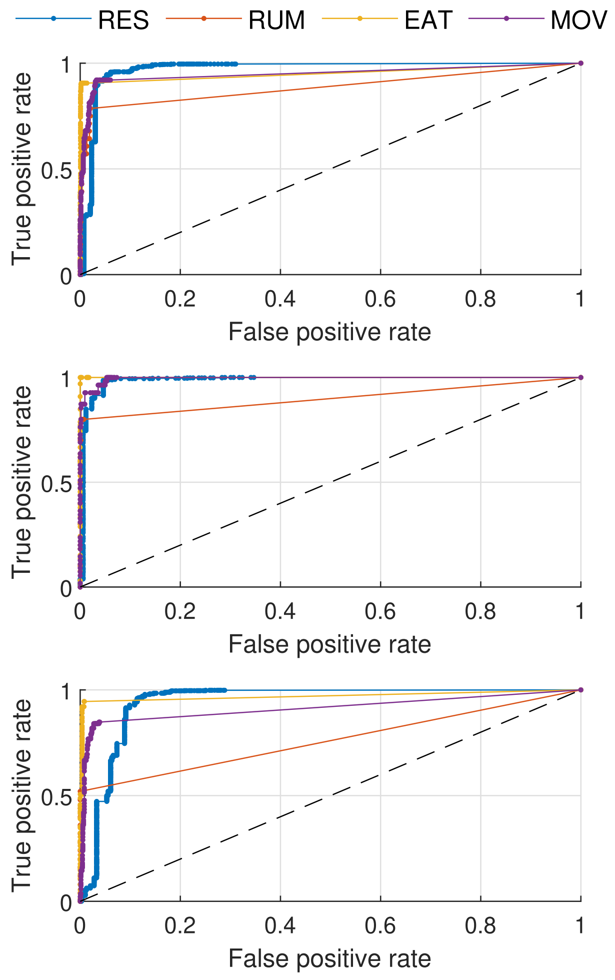


FIGURE 17. The Receiver Operating Characteristic (ROC) curves of the three models; best sRL, best GRU and smallest GRU from top to bottom.

sGRU, and bGRU. Differently from Section VII-C, this ADE is the absolute mean difference of the estimated floating-point and full-integer distribution output vector entries. The output bit-width, h_q, influences this error most, as it controls the precision of the activation functions and recurrent outputs. On the other hand, the weight bit-width w_q, which controls the quantization of $\mathbf{U}$ and $\mathbf{W}$, ceases to improve around a bit-width of 10. The bsRL has the lowest ADE, followed by the sGRU and bGRU. We posit this is because of the difference in size while the quantization search space is equivalent. Therefore, larger models incur more operations, resulting in a higher ADE from weight quantization error. Furthermore, the bias bit-width b_q, set to a value of $3w_q$, might be insufficient for larger models.

TABLE 12. Matthews correlation coefficient (MCC) and area under curve (AUC) per behavior from the ROC curves of Fig. 17.

		Area Under Curve (AUC)			
Quantized models	Matthews correlation coefficient	RES	RUM	MOV	EAT
bsRL	0.86	0.97	0.89	0.95	0.95
bGRU	0.93	0.99	0.90	1.00	1.00
sGRU	0.88	0.94	0.76	0.92	0.97

Fig. 16 shows the estimated power consumption for the ranges of w_q and h_q. The power consumption increases by 2.7× between bit-width w_q of 6 and 16 for all models. On the other hand, the output bit-width, h_q, has a similar influence for bsRL at 2.8× but only 1.5× for bGRU in the same range. Furthermore, h_q influences the power mostly linearly up till $h_q = 12$. Therefore, we select an output bit-width, $h_q = 12$, for bsRL and sGRU because of its large influence over the ADE with a small weight bit-width, $w_q = 10$, to balance ADE and power consumption. Furthermore, we select $\{w_q, h_q\} = 9$ for the bGRU as the ADE remains constant at higher precisions.

Fig. 17 shows the ROC curves of these three models that we have selected from the above discussion. These curves show the sensitivity and specificity in false and true positive rates as a function of different thresholds for each behavior versus all others. Although our cow dataset is highly imbalanced (6:3:3:1), all curves approach the vicinity of the left-upper corner and reach an Area Under Curve (AUC) near unity, indicating a high accuracy across all behaviors.

Tables 11 and 12 further describe these models. While the bGRU has the highest overall accuracy at 97.4%, the bsRL and sGRU models have an accuracy of 95.2% and 95.6%. However, the estimated memory and power consumption of the bsRL is roughly 33 and 35 times lower than the bGRU and 4.1 and 4.3 times lower than the sGRU. Next to this, the bsRL, bGRU, and sGRU models report an MCC of 0.86, 0.93, and 0.88, showing that these three models distinguish well between behaviors regardless of dataset imbalance. This is further reinforced with the AUC at an average 0.94, 0.97 and 0.90 over all behaviors for the bsRL, bGRU and sGRU models. Because of its low estimated power and memory, we think that the bsRL is the best option for future ASIC creation or

TABLE 13. Different possible applications for bsRL determined from the amount of 12-byte LPWA transmissions each day consisting of 12 hours using a AX-Sigfox module and a 1300mAh battery.

Transmission interval	Hypothetical battery life	Example applications
10 minutes	<138 days	grazing behavior monitoring
1 hour	<745 days	oestrus detection
3 hours	<1800 days	heat-stress detection
12 hours	<3842 days	lameness, mastitis detection

implementation on the embedded devices ([A]-[D]) with long battery life.

X. DISCUSSION AND FUTURE WORK

The discretized bsRL model increases the accuracy by 8.4% (86.8% to 95.2%) compared to the lowest power-consuming machine learning algorithm considered in Section II [27]. In addition, our model's hypothetical power consumption represents a 99.93% decrease compared to 216μW when inference is implemented on FPGA [D] in Ref. [27]. In general, ASICs are more power-efficient than FPGAs. Reference [84] experimentally determined that GRUs are around 7$\times$ more efficient in terms of performance per watt on ASICs than on FPGAs, which was obtained from experimental data and normalized attempting to account for CMOS technology differences. We estimate that bsRL may consume an energy per inference of at least 1.51μJ on FPGA, which is 369$\times$ less than Ref. [27] per inference. By contrast, the highest-accuracy algorithm discussed in Section II at 98.3% has a 185.8 mJ energy consumption per inference [25]. This energy consumption is drastically higher than ours. In other words, although our estimation is hypothetical, the methods we have applied unquestionably lead to more efficient networks than previous works considering a large margin.

In practice, attaining lowest power consumption also hinges around system-level improvements and reduction methods for neural networks. One technique is network pruning [85], as an alternative to searching for small networks as in this work. Other than using Edge-AI with LPWA, the distribution of algorithms over diversified scales and stages of computing, such as cloud, edge and intermediate or "fog" level, can provide substantial improvements in energy efficiency and latency in high-speed wireless environments [86].

For a hypothetical future device, purely as a representative example, one could use an AX-SIGFOX module [87] which consumes 0.14C/day in sleep mode and 0.46C to transmit a single 12-byte message. Furthermore, Kionix KX126-1063 [35] accelerometer consumes 0.605C/day at 25Hz. Assuming that the bsRL has a supply voltage of 0.8V on 45nm CMOS, we estimate a hypothetical charge consumption at 0.0131C/day, considerably lower than the other modules. Therefore, a 618mAh battery would be required for up to 5 years of battery life, considering a daily 12-byte message. Even though this is an idealized estimate and real-life performance may be poorer, this battery life is well within the life expectancy of a livestock cow of around 3-4 years [88]. As said, these numbers remain absolutely tentative as this stage, even though they are derived from experimental lab measurements. In the field, consideration of multiple non-idealities becomes essential, for example in relation to static currents and power conversion efficiency. Nevertheless, the obtained values are such that even after substantial degradation, practical viability would not be compromised. Other than a future ASIC, this model could be implemented on a suitable FPGA, one example of which is the UL640 FPGA [A], costing only 1.82$. For example, hypothetically combining that with the KX126-1063 accelerometer at 2.02$, a 1300mAh battery at 2.20$ and AX-sigfox module at 5.40$, the total module costs would be 11.44$. While this does not include all auxiliary components, board fabrication, enclosure and battery, it proves that in principle TinyCowNet allows for future low-cost devices with long battery lifetimes and high accuracy. These monetary indications are purely offered as a budgetary order-of-magnitude example, as they are susceptible to market fluctuations and the authors refrain from endorsing any particular implementation device.

Table 13 presents an overview of farm applications for different transmission frequencies of 12-byte messages with a 1300mAh battery. At an interval of 10 minutes, the time that a cow ruminates and eats from Eq. (6) can be directly used to determine grass intake during short intervals. The regressed prevalence also serve as the base information to detect abnormalities such as diseases and oestrus in the future. For disease detection, a daily update could be enough, but transmission in hourly intervals is necessary to detect oestrus that lasts around 12 hours. Even though these numbers represent idealized estimates, they confirm the impact of frequency on device lifetime without recharging.

Future implementation of TinyCowNet on embedded devices could help towards improving farm efficiency by reducing diseases, enhancing feed control, and automating fertility management. These three areas have a potentially significant impact on gas emission reduction of cattle farms [6], [8], [9]. This work identified the issues of current Edge AI animal behavior estimation literature and proposed to solve these with cow behavior distribution regressing RNNs titled TinyCowNet. Random Search and integer quantization were introduced to create minimal but accurate edge-implementable TinyCowNets tentatively estimated by 45nm CMOS experimental literature on SRAM and operation cost. Although ASIC development and implementation on FPGA remains to be done, this work points to the in-principle possibility of future low-power but highly accurate cow behavior estimating devices.

ACKNOWLEDGMENT

The authors would like to thank Ayuka Okumura of Shinshu University, Kazuhiro, Toda, Hiroaki Matsushima and Megumi Sato of Information Services International-Dentsu Ltd., and Junko Asakawa of EBI Marketing Company Ltd., for their collaboration during data acquisition and annotation of the cow behavior dataset. Korkut Kaan Tokgoz was with the Tokyo Institute of Technology, Japan, for the entire duration of the research project, and is now with the Faculty of Engineering and Natural Sciences, Sabanci University, Turkey.

REFERENCES

[1] T. Raney *et al.*, "The state of food and agriculture: Livestock in the balance," Food Agricult. Org. United Nations, Rome, Italy, 2009.

[2] P. J. Gerber, H. Steinfeld, B. Henderson, A. Mottet, C. Opio, J. Dijkman, A. Falcucci, and G. Tempio, *Tackling Climate Change Through Livestock: A Global Assessment of Emissions and Mitigation Opportunities*. Food and Agriculture Organization of the United Nations (FAO), 2013.

[3] J. Martinez, P. Dabert, S. Barrington, and C. Burton, "Livestock waste treatment systems for environmental quality, food safety, and sustainability," *Bioresource Technol.*, vol. 100, no. 22, pp. 5527–5536, Nov. 2009, doi: 10.1016/j.biortech.2009.02.038.

[4] M. M. Rojas-Downing, A. P. Nejadhashemi, T. Harrigan, and S. A. Woznicki, "Climate change and livestock: Impacts, adaptation, and mitigation," *Climate Risk Manage.*, vol. 16, pp. 145–163, Jan. 2017.

[5] *UNFCCC, Adoption of the Paris agreement. COP*, Proc. 25th Session Paris, P. Agreement, UNFCCC, Bonn, Germany, vol. 30, 2015. [Online]. Available: https://unfccc.int/about-us/contact-and-directions/find-contact-by-issue

[6] 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.

[7] 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.

[8] Y. Blaise, A. Andriamandroso, B. Heinesch, Y. Beckers, E. C. Muñoz, F. Lebeau, and J. Bindelle, "Influences of feeding behaviour and forage quality on diurnal methane emission dynamics of grazing cows," *Precis. Livestock Farming*, vol. 17, pp. 759–769, Sep. 2017.

[9] 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.

[10] G. M. Pereira, B. J. Heins, and M. I. Endres, "Technical note: Validation of an ear-tag accelerometer sensor to determine rumination, eating, and activity behaviors of grazing dairy cattle," *J. Dairy Sci.*, vol. 101, no. 3, pp. 2492–2495, Mar. 2018.

[11] J. Wang, M. Bell, X. Liu, and G. Liu, "Machine-learning techniques can enhance dairy cow estrus detection using location and acceleration data," *Animals*, vol. 10, no. 7, p. 1160, Jul. 2020.

[12] J. Siivonen, S. Taponen, M. Hovinen, M. Pastell, B. J. Lensink, S. Pyörälä, and L. Hänninen, "Impact of acute clinical mastitis on cow behaviour," *Appl. Animal Behav. Sci.*, vol. 132, nos. 3–4, pp. 101–106, Jul. 2011.

[13] A. Stone, "Precision dairy farming technology solutions for detecting dairy cow disease to improve dairy cow well-being," Ph.D. dissertation, Dept. Animal Food Sci., Univ. Kentucky, Lexington, Kentucky, 2017.

[14] *T4C Management System—Lely*. Accessed: Aug. 12, 8, 2021. [Online]. Available: https://www.lely.com/farming-insights/t4c-management-system/

[15] *Dairy Cow Monitoring—Allflex Livestock Intelligence Australia*. Accessed: Aug. 12, 8, 2021. [Online]. Available: https://www.allflex.global/au/dairy-cow-monitoring/

[16] *Dairy Health Monitoring—Herd Health Management—Nedap*. Accessed: Aug. 12, 8, 2021. [Online]. Available: https://www.nedap-livestockmanagement.com/dairy-farming/solutions/nedap% -cowcontrol/health-monitoring/

[17] *Cloud Cattle Herd Management System Farmnote Cloud—Farmnote co., Ltd.* Accessed: Aug. 12, 8, 2021. [Online]. Available: https://farmnote.jp/features/

[18] P. P. A. Oliveira, R. R. S. Corte, S. L. Silva, P. H. M. Rodriguez, L. S. Sakamoto, A. F. Pedroso, R. R. Tullio, and A. Berndt, "The effect of grazing system intensification on the growth and meat quality of beef cattle in the Brazilian Atlantic forest biome," *Meat Sci.*, vol. 139, pp. 157–161, May 2018.

[19] 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.

[20] J. P. Dominguez-Morales, A. Rios-Navarro, M. Dominguez-Morales, R. Tapiador-Morales, D. Gutierrez-Galan, D. Cascado-Caballero, A. Jimenez-Fernandez, and A. Linares-Barranco, "Wireless sensor network for wildlife tracking and behavior classification of animals in Doñana," *IEEE Commun. Lett.*, vol. 20, no. 12, pp. 2534–2537, Dec. 2016.

[21] 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.

[22] V. M. Suresh, R. Sidhu, P. Karkare, A. Patil, Z. Lei, and A. Basu, "Powering the IoT through embedded machine learning and Lora," in *Proc. IEEE 4th World Forum Internet Things (WF-IoT)*, Feb. 2018, pp. 349–354.

[23] G. F. Marchioro, C. Cornou, A. R. Kristensen, and J. Madsen, "Sows' activity classification device using acceleration data—A resource constrained approach," *Comput. Electron. Agricult.*, vol. 77, no. 1, pp. 110–117, Jun. 2011.

[24] D. Gutierrez-Galan, J. P. Dominguez-Morales, E. Cerezuela-Escudero, A. Rios-Navarro, R. Tapiador-Morales, M. Rivas-Perez, M. Dominguez-Morales, A. Jimenez-Fernandez, and A. Linares-Barranco, "Embedded neural network for real-time animal behavior classification," *Neurocomputing*, vol. 272, pp. 17–26, Jan. 2018.

[25] 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.

[26] S. P. le 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.

[27] J. Bartels, K. K. Tokgoz, M. Fukawa, S. Otsubo, L. Chao, I. Rachi, K. Takeda, and H. Ito, "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.

[28] R. Arablouei, L. Currie, B. Kusy, A. Ingham, P. L. Greenwood, and G. Bishop-Hurley, "*In-situ* classification of cattle behavior using accelerometry data," *Comput. Electron. Agricult.*, vol. 183, Apr. 2021, Art. no. 106045.

[29] M. Horowitz, "Energy table for 45 nm process," Stanford VLSI Wiki, Stanford, CA, USA.

[30] S. Akashe, "A low-leakage current power 45-nm CMOS SRAM," *Indian J. Sci. Technol.*, vol. 4, no. 4, pp. 440–442, Apr. 2011.

[31] S. Nissen *et al.*, "Implementation of a fast artificial neural network library (FANN)," Dept. Comput. Sci. Univ. Copenhagen, Copenhagen, Denmark, Tech. Rep., 2003, p. 29, vol. 31.

[32] Q. Zhu, "On the performance of Matthews correlation coefficient (MCC) for imbalanced dataset," *Pattern Recognit. Lett.*, vol. 136, pp. 71–80, Aug. 2020.

[33] M. A. Lones, "How to avoid machine learning pitfalls: A guide for academic researchers," 2021, *arXiv:2108.02497*.

[34] M. Christ, N. Braun, J. Neuffer, and A. W. Kempa-Liehr, "Time series feature extraction on basis of scalable hypothesis tests (tsfresh—A Python package)," *Neurocomputing*, vol. 307, pp. 72–77, Sep. 2018.

[35] *Kx126-1063, Tri-Axis, User Selectable ± 2G, 4G, 8G, Digital (I^2C/SPI) Output, Integrated Pedometer*. Accessed: Jun. 17, 2020. [Online]. Available: https://www.kionix.com/product/KX126-1063

[36] L. Jing and Y. Tian, "Self-supervised visual feature learning with deep neural networks: A survey," *IEEE Trans. Pattern Anal. Mach. Intell.*, vol. 43, no. 11, pp. 4037–4058, Nov. 2021.

[37] A. Chowdhury, J. Rosenthal, J. Waring, and R. Umeton, "Applying self-supervised learning to medicine: Review of the state of the art and medical implementations," *Informatics*, vol. 8, no. 3, p. 59, Sep. 2021.

[38] B. Khaertdinov, E. Ghaleb, and S. Asteriadis, "Contrastive self-supervised learning for sensor-based human activity recognition," in *Proc. IEEE Int. Joint Conf. Biometrics (IJCB)*, Aug. 2021, pp. 1–8.

[39] H. Ito, K. I. Takeda, K. K. Tokgoz, L. Minati, M. Fukawa, L. Chao, J. Bartels, I. Rachi, and A. Sihan, "Japanese black beef cow behavior classification dataset," Jan. 2022, doi: 10.5281/zenodo.5849025.

[40] A. Singh and A. Purohit, "A survey on methods for solving data imbalance problem for classification," *Int. J. Comput. Appl.*, vol. 127, no. 15, pp. 37–41, Oct. 2015.

[41] C. Li, K. Tokgoz, M. Fukawa, J. Bartels, T. Ohashi, K.-I. Takeda, and H. Ito, "Data augmentation for inertial sensor data in CNNs for cattle behavior classification," *IEEE Sensors Lett.*, vol. 5, no. 11, pp. 1–4, Nov. 2021.

[42] S. Palacio, R. Bergeron, S. Lachance, and E. Vasseur, "The effects of providing portable shade at pasture on dairy cow behavior and physiology," *J. Dairy Sci.*, vol. 98, no. 9, pp. 6085–6093, Sep. 2015.

[43] L. Schmeling, G. Elmamooz, P. T. Hoang, A. Kozar, D. Nicklas, M. Sünkel, S. Thurner, and E. Rauch, "Training and validating a machine learning model for the sensor-based monitoring of lying behavior in dairy cows on pasture and in the barn," *Animals*, vol. 11, no. 9, p. 2660, Sep. 2021.

[44] S. Hosseininoorbin, S. Layeghy, B. Kusy, R. Jurdak, G. J. Bishop-Hurley, P. L. Greenwood, and M. Portmann, "Deep learning-based cattle behaviour classification using joint time-frequency data representation," *Comput. Electron. Agricult.*, vol. 187, Aug. 2021, Art. no. 106241.

[45] M. Abadi *et al.* (2015). *TensorFlow: Large-Scale Machine Learning on Heterogeneous Systems*. [Online]. Available: http://tensorflow.org/

[46] F. Chollet. (2015). *Keras*. [Online]. Available: https://github.com/fchollet/keras

[47] D. E. Rumelhart, G. E. Hinton, and R. J. Williams, *Learning Internal Representations By Error Propagation*. Cambridge, MA, USA: MIT Press, 1988.

[48] J. L. Elman, "Finding structure in time," *Cognit. Sci.*, vol. 14, no. 2, pp. 179–211, Mar. 1990.

[49] R. Pascanu, T. Mikolov, and Y. Bengio, "On the difficulty of training recurrent neural networks," in *Proc. Int. Conf. Mach. Learn.*, 2013, pp. 1310–1318.

[50] H. H. Tan and K. H. Lim, "Vanishing gradient mitigation with deep learning neural network optimization," in *Proc. 7th Int. Conf. Smart Comput. Commun. (ICSCC)*, Jun. 2019, pp. 1–4.

[51] S. Hochreiter and J. Schmidhuber, "Long short-term memory," *Neural Comput.*, vol. 9, no. 8, pp. 1735–1780, 1997.

[52] S. Ghosh, O. Vinyals, B. Strope, S. Roy, T. Dean, and L. Heck, "Contextual LSTM (CLSTM) models for large scale NLP tasks," 2016, *arXiv:1602.06291*.

[53] K. Cho, B. van Merrienboer, C. Gulcehre, D. Bahdanau, F. Bougares, H. Schwenk, and Y. Bengio, "Learning phrase representations using RNN encoder-decoder for statistical machine translation," 2014, *arXiv:1406.1078*.

[54] R. Jozefowicz, W. Zaremba, and I. Sutskever, "An empirical exploration of recurrent network architectures," in *Proc. Int. Conf. Mach. Learn.*, 2015, pp. 2342–2350.

[55] S. Li, W. Li, C. Cook, C. Zhu, and Y. Gao, "Independently recurrent neural network (IndRNN): Building a longer and deeper RNN," in *Proc. IEEE/CVF Conf. Comput. Vis. Pattern Recognit.*, Jun. 2018, pp. 5457–5466.

[56] A. Krizhevsky, I. Sutskever, and G. E. Hinton, "Imagenet classification with deep convolutional neural networks," in *Proc. Adv. Neural Inf. Process. Syst.*, vol. 25, 2012, pp. 1097–1105.

[57] A. G. Howard, M. Zhu, B. Chen, D. Kalenichenko, W. Wang, T. Weyand, M. Andreetto, and H. Adam, "MobileNets: Efficient convolutional neural networks for mobile vision applications," 2017, *arXiv:1704.04861*.

[58] S. Han, J. Pool, J. Tran, and W. Dally, "Learning both weights and connections for efficient neural network," in *Proc. Adv. Neural Inf. Process. Syst.*, vol. 28, 2015.

[59] 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.

[60] 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.

[61] M. Van Keirsbilck, A. Keller, and X. Yang, "Rethinking full connectivity in recurrent neural networks," 2019, *arXiv:1905.12340*.

[62] F. Silfa, G. Dot, J.-M. Arnau, and A. Gonzalez, "E-Pur: An energy-efficient processing unit for recurrent neural networks," in *Proc. 27th Int. Conf. Parallel Architectures Compilation Techn.*, 2018, pp. 1–12.

[63] *iCE40 Ultralite Family Data Sheet—Lattice*. Accessed: Jun. 18, 2020. [Online]. Available: https://www.latticesemi.com/view_document?document_id=50945

[64] *Igloo Nano Low Power Flash FPGAs Datasheet—Microsemi*. Accessed: Jun. 18, 2020. [Online]. Available: https://www.microsemi.com/document-portal/doc_download/130695-igloo-na% no-low-power-flash-fpgas-datasheet

[65] *Access Line Ultra-Low-Power 32-Bit MCU Arm-Based Cortex-m0+, up to 192 kb Flash, 20 kb SRAM, 6 kb EEPROM, ADC Datasheet—Stmicroelectronics*. Accessed: Jun. 18, 2020. [Online]. Available: https://www.st.com/resource/en/datasheet/stm32l071c8.pdf

[66] *Ice40 Ultraplus Family Data Sheet—Lattice*. Accessed: Jun. 18, 2020. [Online]. Available: https://www.latticesemi.com/view_document?document_id=51968

[67] H. W. Lin, M. Tegmark, and D. Rolnick, "Why does deep and cheap learning work so well?" *J. Stat. Phys.*, vol. 168, pp. 1223–1247, Sep. 2017.

[68] A. Howard, M. Sandler, B. Chen, W. Wang, L.-C. Chen, M. Tan, G. Chu, V. Vasudevan, Y. Zhu, R. Pang, H. Adam, and Q. Le, "Searching for MobileNetV3," in *Proc. IEEE/CVF Int. Conf. Comput. Vis. (ICCV)*, Oct. 2019, pp. 1314–1324.

[69] M. Tan, B. Chen, R. Pang, V. Vasudevan, M. Sandler, A. Howard, and Q. V. Le, "MnasNet: Platform-aware neural architecture search for mobile," in *Proc. IEEE/CVF Conf. Comput. Vis. Pattern Recognit. (CVPR)*, Jun. 2019, pp. 2820–2828.

[70] B. Zoph and Q. V. Le, "Neural architecture search with reinforcement learning," 2016, *arXiv:1611.01578*.

[71] L. Xie and A. Yuille, "Genetic CNN," in *Proc. IEEE Int. Conf. Comput. Vis. (ICCV)*, Oct. 2017, pp. 1379–1388.

[72] H. Liu, K. Simonyan, and Y. Yang, "DARTS: Differentiable architecture search," in *Proc. Int. Conf. Learn. Represent.*, 2019, pp. 1–13. [Online]. Available: https://openreview.net/forum?id=S1eYHoC5FX

[73] K. Yu, C. Sciuto, M. Jaggi, C. Musat, and M. Salzmann, "Evaluating the search phase of neural architecture search," 2019, *arXiv:1902.08142*.

[74] I. Goodfellow, Y. Bengio, A. Courville, and Y. Bengio, *Deep Learning*. vol. 1, no. 2. Cambridge, MA, USA: MIT Press, 2016.

[75] T.-T. Wong and N.-Y. Yang, "Dependency analysis of accuracy estimates in K-fold cross validation," *IEEE Trans. Knowl. Data Eng.*, vol. 29, no. 11, pp. 2417–2427, Nov. 2017.

[76] S. Boughorbel, F. Jarray, and M. El-Anbari, "Optimal classifier for imbalanced data using Matthews correlation coefficient metric," *PLoS ONE*, vol. 12, no. 6, Jun. 2017, Art. no. e0177678.

[77] A. P. Bradley, "The use of the area under the ROC curve in the evaluation of machine learning algorithms," *Pattern Recognit.*, vol. 30, no. 7, pp. 1145–1159, 1997.

[78] J. Choi, Z. Wang, S. Venkataramani, P. I.-J. Chuang, V. Srinivasan, and K. Gopalakrishnan, "PACT: Parameterized clipping activation for quantized neural networks," 2018, *arXiv:1805.06085*.

[79] E. Sari, V. Courville, and V. P. Nia, "IRNN: Integer-only recurrent neural network," 2021, *arXiv:2109.09828*.

[80] B. Jacob, S. Kligys, B. Chen, M. Zhu, M. Tang, A. Howard, H. Adam, and D. Kalenichenko, "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.

[81] *Tensorflow for Mobile & IoT—Model Optimization*. Accessed: Jun. 26, 2020. [Online]. Available: https://www.tensorflow.org/lite/performance/model_optimization

[82] B. Prince, *High Performance Memories: New Architecture DRAMs and SRAMs-Evolution and Function*. Hoboken, NJ, USA: Wiley, 1999.

[83] *Tsubame 3.0 Computing Services Top Page*. Accessed: Jun. 26, 2020. [Online]. Available: https://www.t3.gsic.titech.ac.jp/index.php/en

[84] E. Nurvitadhi, J. Sim, D. Sheffield, A. Mishra, S. Krishnan, and D. Marr, "Accelerating recurrent neural networks in analytics servers: Comparison of FPGA, CPU, GPU, and ASIC," in *Proc. 26th Int. Conf. Field Program. Log. Appl. (FPL)*, Aug. 2016, pp. 1–4.

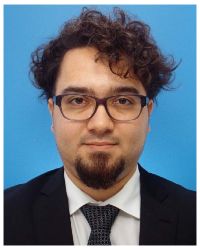
[85] M. Zhu and S. Gupta, "To prune, or not to prune: Exploring the efficacy of pruning for model compression," 2017, *arXiv:1710.01878*.

[86] N. Janbi, I. Katib, A. Albeshri, and R. Mehmood, "Distributed artificial Intelligence-as-a-Service (DAIaaS) for smarter IoE and 6G environments," *Sensors*, vol. 20, no. 20, p. 5796, Oct. 2020.

[87] *AX-SIGFOX Antstamp: Ultra-Low Power, at Command Controlled. SIGFOX Compliant Modules*. Accessed: Jun. 6, 2020. [Online]. Available: https://www.onsemi.com/pdf/datasheet/ax-sigfox-mods-d.pdf

[88] A. De Vries and M. I. Marcondes, "Review: Overview of factors affecting productive lifespan of dairy cows," *Animal*, vol. 14, no. S1, pp. s155–s164, Mar. 2020.

JIM BARTELS (Graduate Student Member, IEEE) received the B.Sc. degree (Hons.) in electrical engineering from the Eindhoven University of Technology, 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, where he is currently pursuing the Ph.D. degree in the same field. He is the Vice-Chair of the IEEE Student Branch, Tokyo Institute of Technology. His current research interests include embedded machine learning, spiking neural networks, quantization, and neural architecture search.



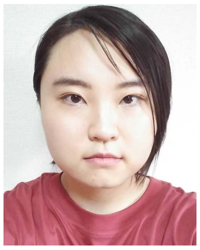
KORKUT KAAN TOKGOZ (Member, IEEE) received the B.Sc. and M.Sc. degrees from the Electrical and Electronics Engineering Department, Middle East Technical University, Ankara, Turkey, in 2009 and 2012, respectively, and the M.Eng. and Ph.D. degrees from the Department of Physical Electronics, Tokyo Institute of Technology, Tokyo, Japan, in 2014 and 2018, respectively. From 2018 to 2019, he worked as a Senior Researcher/Assistant Manager with NEC Corporation, 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 with the Tokyo Institute of Technology. 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 with Evrim Co. 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. He was a recipient of several awards and scholarships, including 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.

SIHAN A (Graduate Student Member, IEEE) received the B.Sc. (Agr.) degree in animal science from China Agricultural University, in 2016. He is currently pursuing the M.E. degree in electrical and electronic engineering with the Tokyo Institute of Technology, Yokohama, Japan. His research interest includes farm applications of deep learning on edge devices to improve both animal welfare and production.

MASAMOTO FUKAWA joined TechnoPro Design, in 2017, and is currently working on a joint research project with the Tokyo Institute of Technology, Yokohama, Japan, where his tasks involve machine learning and data science.

SHOHEI OTSUBO joined TechnoPro Design, in 2012, and has worked on a joint research project with the Tokyo Institute of Technology, Yokohama, Japan, where he was involved in implementing low-power IoT devices with power management systems.

CHAO LI (Graduate Student Member, IEEE) received the B.E. degree in electronic information and electrical engineering from the Dalian University of Technology, China, in 2017, and the M.E. degree in electrical and electronic engineering from the Tokyo Institute of Technology, Yokohama, Japan, in 2020, where she is currently pursuing the Ph.D. degree. She has been the Chair of the IEEE Student Branch, Tokyo Institute of Technology, since 2021. Her research interests include data collection for machine learning and activity classification at the edge. She is a Student Member of the Institute of Electronics, Information and Communication Engineers (IEICE).



IKUMI RACHI received the B.E. degree from the Tokyo Institute of Technology, Yokohama, Japan, in 2021, where she is currently pursuing the M.E. degree. Her current research interests include investigation of accelerometer sampling rate and its effect on cow behavior estimation systems.

KEN-ICHI TAKEDA received the B.Agr. degree from Nippon Veterinary and Animal Science University, Tokyo, in 1995, and the M.Agr. and Ph.D. degrees from Tohoku University, in 1997 and 2000, respectively. He is currently an Associate Professor at the Institute of Agriculture, Academic Assembly, Shinshu University. His research interests include farm animal welfare and herbivores management in grassland.

LUDOVICO MINATI (Senior Member, IEEE) received the B.S. degree in information technology and computing and the M.S. degree in science from The Open University, Milton Keynes, U.K., in 2004 and 2006, respectively, 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, 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, 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. He is a European Engineer (Eur. Ing.), a Chartered Engineer, and a member with the Institution of Engineering and Technology, U.K. He is a Chartered Physicist and a member with the Institute of Physics, U.K. He is 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), the Institute of Electrical Engineers of Japan, and the Japan Neuroscience Society.

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 at the Institute of Innovative Research, Tokyo Institute of Technology. Furthermore, he has been the Co-Founder/CEO of EVRIM Company Ltd., Yokohama, since 2020; and the CTO of Adapt-IP, Inc., Yokohama, since 2021. His research interests include ultra low-power RF circuits, a high-sensitivity MEMS accelerometer, a low-noise and crystal-less RF synthesizer, a cow management system exploiting IoT and deep/machine learning, a closed hydroponic equipment for space agriculture, an intraoral implantable measurement systems, dental-bite evaluation and optimization technology, a breast cancer detection system exploiting RF technologies, circuit technology for a next-generation EB pattern exposure systems, and physical design of bump-less 3D IC. He 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.