This article has been accepted for publication in IEEE Journal of Biomedical and Health Informatics. This is the author's version which has not been fully edited and content may change prior to final publication. Citation information: DOI 10.1109/JBHI.2023.3300511

An LSTM-based Neural Network Wearable System for Blood Glucose Prediction in People with Diabetes

Felix Tena, Oscar Garnica, Juan Lanchares Dávila, J. Ignacio Hidalgo

***Abstract*— This article proposes the first hardware implementation of a low-power LSTM neural network targeting a wearable medical device designed to predict blood glucose at a 30-minute horizon. This work aims to reduce energy consumption by proposing new activation functions that target hardware implementation. On top of this proposal, we also prove there is room for improvement in energy consumption by applying neural network optimizations at the algorithmic, such as quantization, and architecture level, LSTM hyperparameters, that consider the target hardware. To validate our proposal, we devise an optimized version of the neural network aimed to be wearable and, therefore, to reduce its energy consumption while preserving its accuracy as much as possible. The hardware is implemented on a Xilinx Virtex-7 FPGA VC707 Evaluation Kit. It is compared with (i) a faithful design of the original neural network implemented on the same evaluation kit, (ii) three state-of-the-art LSTM-based FPGA implementations, and (iii) software implementations running in cutting-edge smartphones: OnePlus® Nord™ and an Apple® iPhone 13 Pro™ with artificial intelligence hardware accelerators. Our proposal consumes between $\times 1020$ and $\times 7$ less energy than the software implementations, being the most efficient system compared to the smartphones. On the other hand, its energy efficiency, measured in GFLOP/J, is between $\times 2.84$ and $\times 7.82$ greater than other state-of-the-art LSTM implementations, proving to be the most suitable implementation for a wearable system for blood glucose prediction.**

***Index Terms*— artificial neural networks, blood glucose prediction, deep learning, diabetes, energy consumption in neural networks, FPGA, wearable sensors.**

I. Introduction

Diabetes mellitus (DM) is a group of metabolic disorders characterized by high blood glucose (BG) concentrations. It is accompanied by disturbances in the metabolism of carbohydrates (CH), proteins, and fats resulting from defects in insulin secretion, insulin action, or both [1]. The systems designed in this work focus on people with Type 1 diabetes (T1DM), with defects in insulin secretion, that need the injection of exogenous artificial insulin to compensate for the absence of natural insulin secretion. An insufficient insulin dose will remain BG levels at high values, producing acute hyperglycemia symptoms, such as dehydration frequent urination, thirst, or headache, among other symptoms. If acute hyperglycemia is not treated, several complications may appear: in the short term, it can produce ketoacidosis, which produces weakness, confusion, or even diabetic coma, in the long-term, hyperglycemia may cause complications in different organs of the human body [2]. On the other hand, too much insulin administration leads to low values of BG concentration. When BG levels are low, the autonomic nervous system activity increases, sending warning signs such as anxiety, sweating, hunger, or palpitations [3]. If this situation is not reversed, it can appear muscle weakness, inability to drink or eat, convulsions, unconsciousness, and may even lead to death [4].

To minimize the effects of the disease, people with diabetes should maintain healthy glucose levels, between $70\,\mathrm{mg\,dL^{-1}}$ and $180\,\mathrm{mg\,dL^{-1}}$ [3], by following a healthy lifestyle, continuous glucose monitoring, and immediate follow-up actions [5]. Performing the steps necessary to substitute a healthy pancreas daily tasks may be challenging since people with diabetes need to determine the BG levels at different times, monitor CH intakes, administer the appropriate insulin doses, or take correcting actions [6].

Predict BG levels is challenging because of the lack of a general response to the different variables affected by each patient's particularities. Classic glucose models use linear equations and define profiles that do not cover these particularities [5]. Continuous glucose monitoring systems (CGM) have made a significant change in the lives of people with diabetes by monitoring BG levels every five to fifteen minutes automatically [7]. Therefore, CGMs provide time series that can be used as input datasets for BG accurate prediction using machine learning (ML) techniques. Among the most promising ones are those based on neural networks (NN); more precisely on Long-Short Term Memory (LSTM) NNs, which are designed for time series forecasting [8].

The great boom of artificial intelligence (AI) is linked to the growth of available data and the consolidation of the cloud, being cloud AI the most common technology. The problem with this type of AI is the need for an internet connection compromising data security and increasing latency, which can harm real-time applications. To solve this problem, the concept of edge computing has been devised, devices that are not in the cloud but communicate with it process the information locally [9]. To do this, several optimization techniques must be implemented, in addition , designing specialized hardware to implement the NN on edge may reduce energy consumption. In this way, it is possible to have a wearable and autonomous NN device for BG prediction.

In previous work [6], we proposed two software NN-based ensemble models and compared them with ten state-of-the-art software NNs for BG predictions to find the best predictor in terms of accuracy and complexity. LSTMs have shown, among NNs, better accuracy in time series prediction using the lowest number of trainable parameters, which leads to less energy consumption, a critical constraint in the design of a wearable device. In this new work, our goal is to design a wearable hardware system with the same prediction accuracy as the best NN for BG prediction according to the criteria of our previous work. This reduces the dependency of the system on the network connection, increases its reliability, and improves data security and accessibility . A critical issue in wearable systems is to reduce their

This work was supported in part by Fundación Eugenio Rodríguez Pascual 2019—Development of Adaptive and Bioinspired Systems for Glycaemic Control with Continuous Subcutaneous Insulin Infusions and Continuous Glucose Monitors; the Spanish Ministerio de Innovación, Ciencia y Universidad—grant RTI2018-095180-B-I00; Madrid Regional Government–FEDER grants B2017/BMD3773 (GenObIA-CM) and Y2018/NMT- 4668 (Micro-Stress- MAP-CM); Consejería de Educación e Investigación de la Comunidad de Madrid; European Social Fund.

Felix Tena is with the Department of Computer Architecture and Automation, Facultad de Informática, Universidad Complutense de Madrid, 28040 Madrid, Spain (e-mail: feltena@ucm.es)

Oscar Garnica is with the Department of Computer Architecture and Automation, Facultad de Informática, Universidad Complutense de Madrid, 28040 Madrid, Spain (e-mail: ogarnica@ucm.es)

Juan Lanchares Dávila is with the Department of Computer Architecture and Automation, Facultad de Informática, Universidad Complutense de Madrid, 28040 Madrid, Spain (e-mail: julandan@ucm.es)

J. Ignacio Hidalgo is with the Department of Computer Architecture and Automation, Facultad de Informática, Universidad Complutense de Madrid, 28040 Madrid, Spain (e-mail: hidalgo@ucm.es)

This work is licensed under a Creative Commons Attribution 4.0 License. For more information, see https://creativecommons.org/licenses/by/4.0/

This article has been accepted for publication in IEEE Journal of Biomedical and Health Informatics. This is the author's version which has not been fully edited and content may change prior to final publication. Citation information: DOI 10.1109/JBHI.2023.3300511

energy consumption. To this aim, first, we select LSTM NN because they are best-in-class regarding the trade-off between accuracy and complexity. Secondly, we devise several optimizations to meet the power consumption constraints allowing greater battery durability and lower weight without compromising performance.. Recently, other NNs have been proposed, i.e., transformers [10], with outstanding results in text translation or time series prediction [11]. They apply the principles of the neural sequence models using an encoder-decoder architecture, where encoders comprise multiple multi-head self-attention modules and position feed-forward networks, and decoders use cross-attention models in addition to the same modules that encoders do. This architecture requires more hardware than the LSTM we have devised in this work, which increases the energy consumption that is the critical metric in wearable devices.

In summary, the contributions of this article are:

- We propose two activation functions aimed at reducing the energy consumption of NNs without compromising their accuracy since activation functions are the elements with higher computational and energy costs.
- We present a wearable, and therefore low-power, FPGA-based LSTM NN using our proposal in addition to state-of-the-art methods to reduce the computing demands on the NN. We compare its energy consumption and energy efficiency with the most faithful interpretation of an FPGA-based LSTM NN and with other state-of-the-art FPGA implementations. We also compare its energy consumption with other wearable technologies, namely cutting-edge Android and iOS smartphones.
- We apply our energy-optimized NN architecture to forecast blood glucose prediction for people with T1DM and validate the proposed design in a real-world medical open issue.

As with any deep learning implementation, NNs must be tailed to the meaning of the input and output data which is derived from the dataset and the application domain. This demands (i) a specific data preprocessing stage, paying particular attention to missing values and feature selection, and (ii) the definition of an appropriated NN architecture for the available dataset. Concerning the latter, biomedical data are scarce compared to other ML areas. For this reason, we must use models with reduced complexity so that they can learn using the few data available per patient. This constraint is one of the critical criteria in order to set the architectural parameters of the NN.

We implement our design on an FPGA because they are the most suitable devices to prototype a chip that implements a neural network hardware accelerator [12]: they provide fast development cycles, excellent flexibility and reusability, moderate costs, easy upgrading and feature extension, high performance, they can operate standalone, are COTS products available for everyone, and are a fair element of comparison versus smartphones. The latter means that any other technology to implement integrated circuits would provide better figures of merit than FPGAs (that is, if FPGAs are better than smartphones in this work, a fully designed ASIC would be even better). We compare FPGA implementation with smartphones models because smartphones are the most accessible COTS product to run NN models: everyone has one of them, they include all the hardware elements to run NN without any redesign, and they all have IDEs that facilitate the development and optimization of the programs that model NN.

The rest of this paper is structured as follows. In Section II, we explain how artificial NN works focusing on LSTMs, for those readers that are not familiar with this field and terminology. Section III presents the design concepts applied to the hardware LSTMs. Section IV is devoted to presenting (i) our feature engineering, (ii) the experimental results that define the architecture of the hardware LSTM, and (iii) a comparison with NN implemented in both cutting-edge smartphones and state-of-the-art FPGA NNs. Finally, Section V summarizes the conclusions.

II. Neural Networks

An artificial NN is a ML technique based on simple processing elements, called neurons, which are interconnected in a structure that mimics the neural composition of human brains [13]. Their goal is to process information with a series of mathematical operations with trainable parameters to recognize data patterns.

The number and type of layers define its structure. A layer is a set of neurons arranged so that the outputs of all the neurons in a given layer are connected to all the neurons of the next layer. In this way, data is fed into the NN through the input layer, which sorts the input data to be processed by the NN. Then, a group of hidden layers makes all the mathematical transformations. Finally, the output layer returns the NN's outcome.

Finally, NNs must be trained to adjust weights and biases . The training set is fed into the NN with the predicted values for each timestep. With a learning algorithm, the trainable parameters (weights and biases) are modified to adjust the output of the NN to the actual value. After the training, the test dataset is used to measure the generalization capabilities of the NN. This dataset is not used for training, so the NN cannot learn from the new data patterns that may appear in this subset. Finally, the predictions of the NN are measured against the actual values to obtain the NN's performance.

A. LSTM Neural Network

In this work, we focus on LSTM NNs, since they are the ones that have proven a better performance at BG prediction [6]. LSTMs were created for long-term dependence time series since it was observed that in vanilla recurrent NNs, the problems of vanishing or exploding gradient appear. These problems are characterized by the significant increase or decrease of the error gradient, leading to highly significant weight updates or no weight updates during training. LSTMs solve this problem by enforcing a constant error flow [8]. They keep an internal state that summarizes the input history. So, the current state's output depends on the previous states' output. Thus, the architecture of LSTM is specially designed for problems in which the output depends on the previous history, such as time series forecasting.

Figure 1 illustrates the internal structure of the k-th LSTM neuron at timestep t. It comprises the forget gate, the input gate, which is a combination of two sub gates (the new candidate gate and the input gate) that selects the new candidates, and the output gate. The forget gate selects the irrelevant data and removes it, the input gate selects which information will be updated and conforms the cell state for the next timestep, and the output gate sends the hidden state for the next time step and the next layer. It has two outputs: the hidden state h_t^k, and the cell state. The hidden state is the short-term memory, i.e., it characterizes the last timestep data, depends on the input vector $\mathbf{X}_t^k$, the hidden state of the previous timestep h_{t-1}^k, and cell state c_t^k. In contrast, the cell state is the long-term memory, which travels through the constant error carousel (CEC) with almost no change between the consecutive states allowing multiple timesteps to avoid the vanishing or exploding gradient problems and depends on the input vector, and the previous hidden and cell states.

LSTM neurons have two different activation functions, typically: the hyperbolic tangent, Equation (1), and the sigmoid function, Equation (2). Figure 4 illustrates them. The sigmoid function allows a smooth activation whereas the hyperbolic tangent function has a negative activation; this characteristic is useful for subtracting the information that is no longer relevant.

This work is licensed under a Creative Commons Attribution 4.0 License. For more information, see https://creativecommons.org/licenses/by/4.0/

This article has been accepted for publication in IEEE Journal of Biomedical and Health Informatics. This is the author's version which has not been fully edited and content may change prior to final publication. Citation information: DOI 10.1109/JBHI.2023.3300511

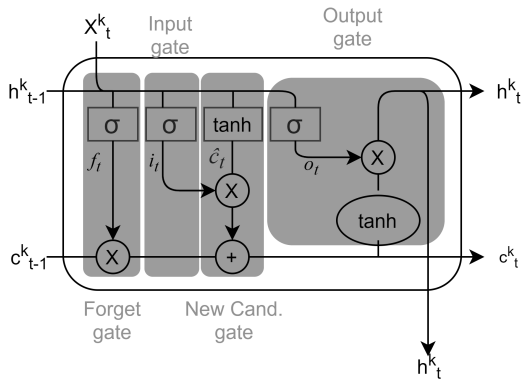


Fig. 1: Block diagram of an LSTM neuron.

$$\tanh(x) = \frac{\exp(2x) - 1}{\exp(2x) + 1} \tag{1}$$

$$\sigma(x) = \frac{1}{1 + \exp(-x)} \tag{2}$$

Equations (3) to (8) illustrate the mathematical operations inside the k-th LSTM neuron. Parameters are identified by a letter that denotes the gate: f (forger, i (input), c (cell state), h (hidden state), and o (output). Each dense neuron has its weights (W^{f^k}, W^{i^k}, $W^{\widehat{c}^k}$, W^{o^k} that multiply the input vector, and U^{f^k}, U^{i^k}, $U^{\widehat{c}^k}$, U^{o^k} that multiply the previous hidden state), and biases (b^{f^k}, b^{i^k}, $b^{\widehat{c}^k}$, and b^{o^k}). Finally, these dense neurons are combined to generate the cell state, c_t^k, and the hidden state, h_t^k, at timestep t.

$$f_t^k = \sigma(\mathbf{X}_t^k \cdot W^{f^k} + h_{t-1}^k \cdot U^{f^k} + b^{f^k}) \tag{3}$$

$$i_t^k = \sigma(\mathbf{X}_t^k \cdot W^{i^k} + h_{t-1}^k \cdot U^{i^k} + b^{i^k}) \tag{4}$$

$$\widehat{c}_t^k = \tanh(\mathbf{X}_t^k \cdot W^{\widehat{c}^k} + h_{t-1}^k \cdot U^{\widehat{c}^k} + b^{\widehat{c}^k}) \tag{5}$$

$$o_t^k = \sigma(\mathbf{X}_t^k \cdot W^{o^k} + h_{t-1}^k \cdot U^{o^k} + b^{o^k}) \tag{6}$$

$$c_t^k = \sigma(f_t^k \cdot c_{t-1}^k + i_t^k \cdot \widehat{c}_t^k) \tag{7}$$

$$h_t^k = tanh(c_t^k) \cdot o_t^k \tag{8}$$

III. Energy-optimized LSTM

A simpler NN may lead to a lower energy consumption hardware implementation since it will need fewer logic blocks (LB) and operations. Thus, according to the conclusions in our previous work [6], we select the NN in [14]. We design two hardware implementations to verify if the energy optimization approaches are feasible. The first is the most faithful interpretation of the NN, which we call `Model HW1`. The second one, called `Model HW2`, is the hardware implementation after the energy optimizations are applied.

Both hardware models are implemented in an FPGA. Taking advantage of FPGA's flexibility, our goal is to design a hardware LSTM to be as customizable and parametrizable as possible. Customizable means that weights and biases are stored in built-in memories and updated without redesigning or resynthesizing the circuit. Whereas parametrizable means that the architectonic parameters of the NN (i.e., number and type of neurons and layers, number of features, and timesteps of the input data) are parameters of the RTL description.

The hardware design is based on hierarchical and distributed control. Each module has its control unit (CU), which is a Moore finite-state machine. The circuit is coordinated so any data consumer module waits for the data generator module to indicate that data is available so it can start its processes.

At last, we use fixed-point format values, which reduce circuit complexity, lowering the computational time and, therefore, energy consumption [15].

A. Memory modules

We use three types of memories: non-volatile read-only memories (ROMs), random-access memories (RAMs), and first in, first out memories (FIFOs). Weights and biases are stored in ROMs since these values are not changed during the prediction process. RAMs store input data, acting as the input layer, and temporally store data between modules. We use them because of their reading and writing nature and accessibility since we can access specific memory positions, which is needed by the characteristics of our design. Finally, FIFOs are used for communication between modules with sequential reading or writing and communication between the LSTM and the output layer .

B. Activation functions

Equations (1) and (2) are exponential functions that require many hardware resources. The most common approaches to implement them are lookup tables, polynomial curve fitting, and piecewise linear approximation [16], to fit the curve as much as possible with small segments. But for this work, we use a combination of an Intelectual Property (IP) module with a polynomial curve fitting for `Model HW1` and the new proposals of linear low-energy activation functions for `Model HW2`.

Figure 2 shows the activation function module for `Model HW1`. It combines Xilinx's CORDIC IP v6.0 and the Taylor activation module that implements a five-order Taylor series approximation of hyperbolic functions. The CORDIC IP only admits input values bounded within the range $[-\pi/4, \pi/4]$, but the activation function module has to handle input values out of this range. The Taylor activation module is designed to compute values out of the previous range with minimal error. The comparator selects the output of the activation function module depending on whether the input data is within or out of $[-\pi/4, \pi/4]$.

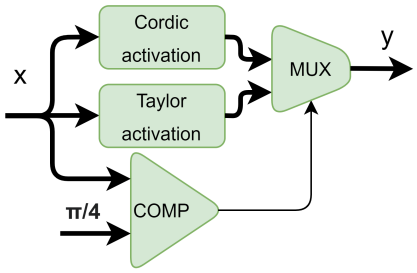


Fig. 2: Block diagram of the activation function module.

Figure 3 is the block diagram of the Taylor activation module. The input raise section is in charge of powering the input; the input can enter the next module directly or multiply itself as many times as necessary. It sends each power to the next module; therefore, we need five loops to raise the input to the fifth power. The constant multiplication section multiplies each input power by its corresponding constant that is hard-wired in the FPGA. Next, the term addition section sums each element and the bound section limits the output between the range [0,1] for Equation (2) and the range [-1,1] for Equation (1).

This work is licensed under a Creative Commons Attribution 4.0 License. For more information, see https://creativecommons.org/licenses/by/4.0/

This article has been accepted for publication in IEEE Journal of Biomedical and Health Informatics. This is the author's version which has not been fully edited and content may change prior to final publication. Citation information: DOI 10.1109/JBHI.2023.3300511

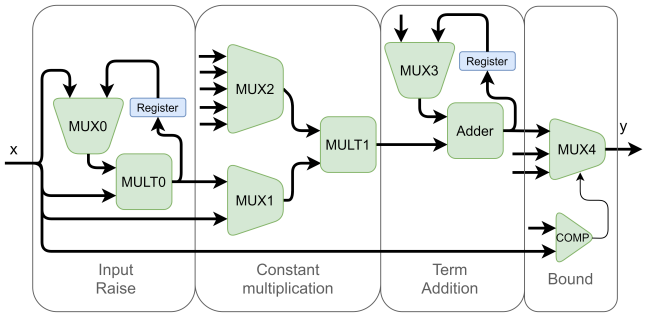


Fig. 3: Taylor activation function module.

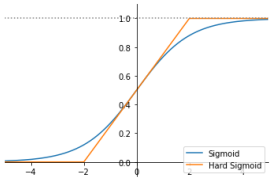
(9)

The first optimization proposal is to replace the two activation functions , Equations (1) and (2), by linear activation functions with similar behaviors but focused on computation speed so the hardware complexity is lower, leading to lower energy consumption. Thus, we substitute the sigmoid function with our modification of the hard sigmoid function, Equation (10). The hard sigmoid function is proposed in the Keras library [17], and we change the slope from 0.20 to 0.25, so it is a power of two. This change will simplify the multiplication of 0.20 by a two-bit right shift, consuming fewer resources. Likewise, we replace the hyperbolic tangent by Equation (11), called by analogy, hard hyperbolic tangent. Thus, we replace hardware description of exponential calculations that requires a high computational cost by hardware associated with a much simpler linear representation. Hard activation functions have two advantages. First, the hard activation functions use only three segments, while the exponential function approximations use as many segments as possible. Secondly, the hard activation functions can be used in the training phase, so there is no accuracy loss when implemented in hardware.

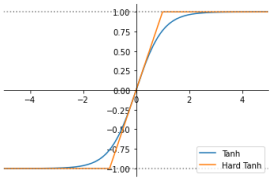
$$g(x) \equiv h\sigma(x) = \begin{cases} 0 & x < -2 \\ 0.25x + 0.5 & -2 \leq x \leq 2 \\ 1 & x > 2 \end{cases} \tag{10}$$

$$g(x) \equiv htanh(x) = \begin{cases} -1 & x < -1 \\ x & -1 \leq x \leq 1 \\ 1 & x > 1 \end{cases} \tag{11}$$

Figure 4 illustrates the similarities between the original activation functions and our proposals. They have the same boundaries and the same y-axis intersection. Since the architecture of an LSTM neuron achieves the non-linearity of the NN, these activation functions are suitable for reducing computational costs.



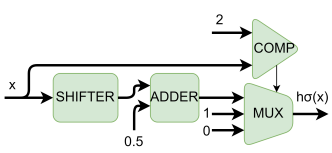
(a) Hard Sigmoid Function



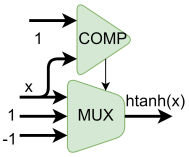
(b) Hard Hyperbolic Tangent Function

Fig. 4: Comparison of common LSTM activation functions (in blue) and their low-energy counterparts (in red).

Figure 5a and Figure 5b show the block diagrams of the modules for Equation (10) and Equation (11), respectively. They are a simple structure of comparators, shift registers, and multiplexers and do not use the CORDIC IP. In addition, there are no data loops which reduces not only the number of modules but also the processing time, leading to a reduction in energy consumption.



(a) Hard Sigmoid Module



(b) Hard Tanh Module

Fig. 5: Hard sigmoid and hard hyperbolic tangent modules

C. Hardware Dense Module

Figure 6 presents the block diagram of the processing elements (PE) on which all the hardware neurons are based. Each PE is built with a gate module, a FIFO, and an activation function module. The gate module is a matrix multiplier and a vector adder that implements the affine transformation of the input vector X by its corresponding weights and bias. The FIFO module has two assignments: first, to sort the gate output; second, to store the information until the next module is ready . Finally, the activation function module composes the affine transformation and the corresponding activation function, $g(\cdot)$. These modules are synchronized through a CU, not shown for the sake of clarity.

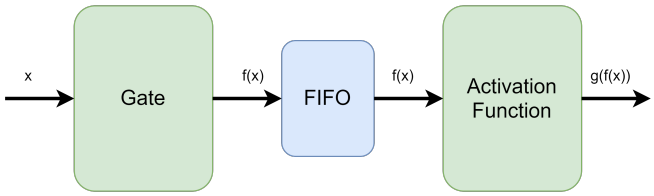


Fig. 6: Block diagram of a processing element.

D. Hardware LSTM Module

Figure 7 shows the hardware implementation of an LSTM neuron. The four PEs are dense neurons that conform the input, forget, new candidate and output gates, respectively. Each gate implements Equations (3) to (6), respectively. The cell state module implements Equation (7). It contains an activation function module, adders and multipliers The hidden state module comprises multipliers and adders to implement Equation (8). After every module, a FIFO interfaces a data generator module with its data consumer module. Finally, we use RAM to store the hidden state for time propagation because the access to the hidden state does not follow a first-in first-out sequencing. Each module also comprises a control unit.

E. Hardware Neural Network Architecture

Figure 8 illustrates the top module of the hardware LSTM. The input layer is a RAM; it stores data so that the next module can access the information when it is ready. The hidden layer consists of one LSTM layer. When the LSTM layer finishes all the operations, it stores data in a FIFO memory and informs the CU that the next module can start. Then, the output layer consists of a dense layer. Finally, the predictions are stored in a RAM.

This work is licensed under a Creative Commons Attribution 4.0 License. For more information, see https://creativecommons.org/licenses/by/4.0/

This article has been accepted for publication in IEEE Journal of Biomedical and Health Informatics. This is the author's version which has not been fully edited and content may change prior to final publication. Citation information: DOI 10.1109/JBHI.2023.3300511

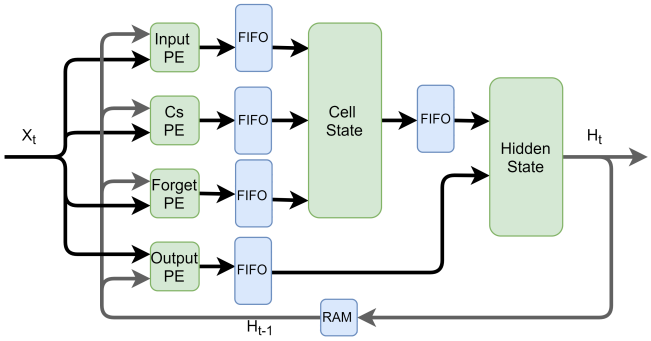


Fig. 7: Block diagram of an LSTM neuron. Blue shadowed blocks are memories whereas green ones are computing modules.

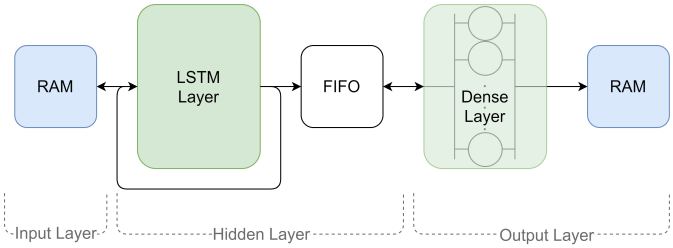


Fig. 8: Top module hardware LSTM. Blue shadowed blocks are memories whereas green ones are computing modules.

Two approaches could be used to design the LSTM network. The first implements each neuron within a layer working in parallel; the design latency is low since all neurons work simultaneously, although with a higher area consumption. The second implements the neurons within the layer sequentially; in this way, the circuit area is lower, but the latency is higher. For the LSTM layer, we use the sequential approach since the FPGA area consumption is lower, which reduces power consumption since fewer LBs need to be powered. Although this affects the prediction speed, it is not a critical issue because CGM readings arrive every five minutes. On the other hand, we use the parallel approach to design the dense layer since we can avoid using memory elements that are not needed by dense neurons. Note that even though we have one dense neuron for this NN, we devise a parametrizable design.

IV. Experimental Results

Figure 9 illustrates the design process. Before designing both hardware models, we define the LSTM architecture and extract the trainable parameters using Python 3.7 and TensorFlow 2.2.0. First, we preprocess the glucose datasets (Section IV-A). Secondly, we recreate the NN proposed in [14] as the benchmark for the different experiments, according to the conclusions in [6], and explore LSTM architectural hyperparameters using 5-minute backpropagation time windows. Once set, we design the NN from scratch in Python so that we can access every operation. This allows us to simulate the implementation of every equation, and, in addition, we can easily test different fixed-point formats as we have full control of the data flow (Section IV-B). From this model, on the one hand, we design the `Model HW1` as a reference from which to optimize the energy consumption. On the other hand, we evaluate at the architectural level the prediction accuracy of the software model after applying the power optimizations. Once the power optimizations have been defined at the architectural level, we design `Model HW2` (Section IV-C).

We design `Model HW1` and `Model HW2` using VHDL 2008, Vivado 2021.1 for synthesis, and QuestaSim 2020.4 for the simulation. We implement both designs in the Xilinx VC707 Evaluation board with a Virtex-7 XC7VX485T-2FFG1761C FPGA Both hardware LSTMs are designed in VHDL and implemented in a Xilinx Virtex-7 XC7VX485T-2FFG1761C FPGA built in TSMC's high performance, low power $28\,\mathrm{nm}$ process technology (28HPL) [18].

Table I shows the characteristics of the proposed NN. The input data has four features, with 25 timesteps per feature per prediction. The hidden layer consists of one LSTM layer with five neurons, and the output layer has one dense neuron to combine all LSTM outputs and make the prediction. The activation function is the hyperbolic tangent, and the recurrent activation function is the sigmoid function.

TABLE I: Neural network architecture hyperparameters [14].

Layer	Hyperarameter	Value
Input	Features	4
	Backpropagation timesteps	25
Hidden	Type	LSTM
	Layers	1
	Neurons per layer	5
	Recurrent activation function	sigmoid
	Activation function	tanh
Out	Type	Dense
	Neurons	1
	Activation function	linear
	Number of parameters	206

The data used for both training and testing are provided by the OHIOT1DM dataset [19] that has recently been used for the "Blood Glucose Prediction Challenge" of the "Workshop on Knowledge Discovery in Healthcare Data," an in vivo database reference in this research area. We use the second cohort of the OHIOT1DM dataset, which contains five males and one female aged between 20 and 80 who participated in an IRB-approved study for eight weeks each. OHIOT1DM dataset provides between 14,943 and 16,547 samples for each patient. To take the different feature measurements, they used Medtronic Enlite CGM sensors, reported life event data via an app, and provided physiological data using the Empatica Embrace fitness band.

A. Data Preprocessing

First of all, we preprocess the data.: select the features required, complete missing values to maintain a continuous database within each patient, and transform features to improve NN performance.

For these experiments, the input features are BG levels (bg), basal insulin (bas), insulin boluses (bol), and CH intakes (ch), so that $\mathbf{x_t} = (\mathrm{bg}(t), \mathrm{bas}(t), \mathrm{bol}(t), \mathrm{ch}(t))$ is the input vector at time t. These features are chosen because they have more impact on BG dynamics.

To complete missing samples of blood glucose, we use a cubic spline. This is a process in which a series of cubic polynomials are fitted between data points, obtaining a continuous and smooth curve [20]. Insulin is smoothed using the Berger's model [21] to model the glucose dynamics in the patient's body.

To input the data into the LSTM, BG levels are multiplied by a factor of 0.01, so the LSTM can reach BG prediction faster according to the algorithm learning rate and submit them on a similar scale to the remaining three features. The rest of the features have been normalized within the range [0,1], which generates a suitable distance between the different values of each feature to facilitate the neural networks appreciating changes for pattern identification. As a starting point, we use 5-minute timesteps for data acquisition with a history of 120 minutes.

This work is licensed under a Creative Commons Attribution 4.0 License. For more information, see https://creativecommons.org/licenses/by/4.0/

This article has been accepted for publication in IEEE Journal of Biomedical and Health Informatics. This is the author's version which has not been fully edited and content may change prior to final publication. Citation information: DOI 10.1109/JBHI.2023.3300511

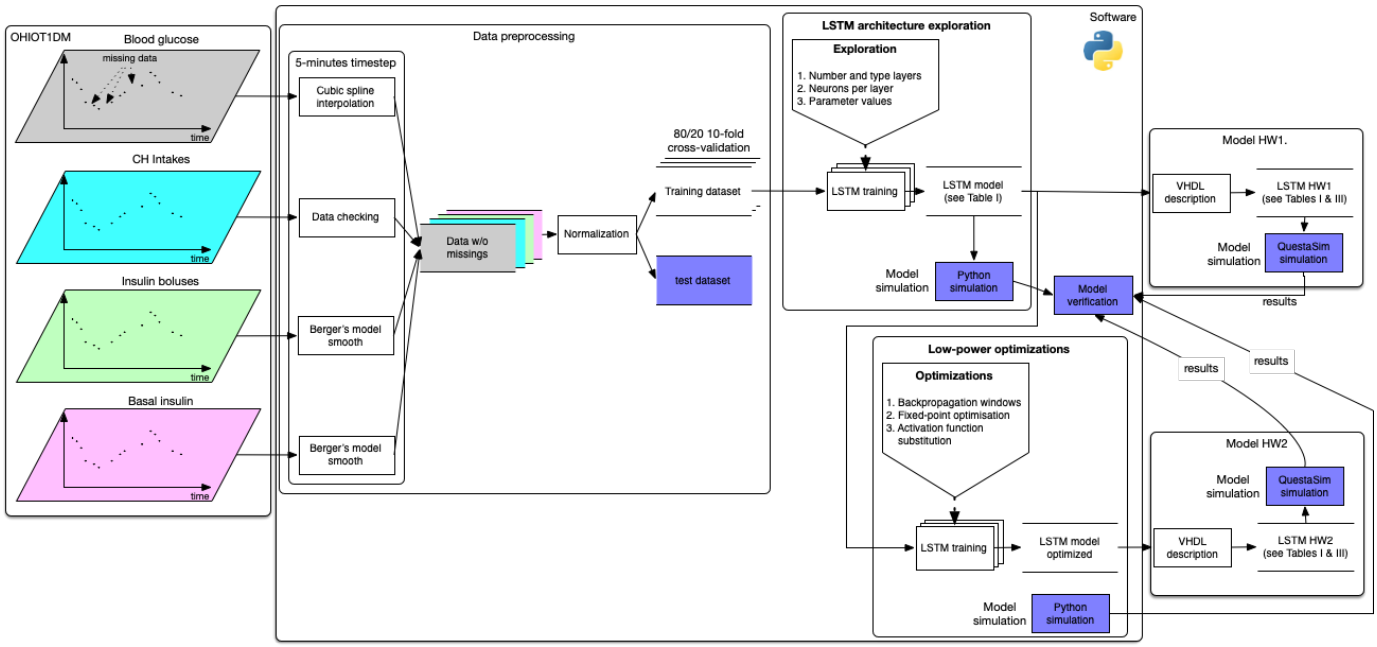


Fig. 9: Flowchart describing the process to define and implement the hardware LSTM models. In violet, those tasks that require test dataset.

B. LSTM Parameters

We split the dataset into training and test data. Training is then performed using an 80/20 10-fold cross-validation approach. The training consists of 100 epochs with an early stopping of 10 epochs' patience.

The training algorithm is the Adam algorithm, which is computationally efficient, has little memory requirement, and is well suited for problems that are large in terms of data and parameters [22]. The learning rate is 0.01 as it fits the scale of the input features and the loss function is the mean squared error Equation (12) . The training is done using floating-point (FP) arithmetic to extract the trainable parameters at maximum accuracy.

BG level rises after 10 to 15 minutes. Hence, the minimum prediction horizon (ph) to take corrective actions is 30 minutes. For this study, BG is forecasted at a 30 minutes prediction horizon.

The first parameter to set is the time window for the propagation through time. The aim is to find the biggest time window to maintain the accuracy of the prediction while reducing the loops of the LSTMs. We start with a 5-minute window and increase it in 5-minute steps up to a 45-minute window.

Figure 10 shows the root mean squared error (RMSE) of every time window for backpropagation, maintaining the 2 hours of backpropagation for every window. The best performance is achieved at 10-minute backpropagation with an RMSE equal to $19.82\,\mathrm{mg\,dL^{-1}}$. We decided to maintain this 10-minute timestep because it improves the error concerning the other time windows. The computation time is reduced by half compared to a 5-minute backpropagation as the LSTM cycles are halved. Hence, there are 12 timesteps for backpropagation plus the current timestep for each dataset feature.

Once the timestep is defined, we compare the prediction accuracy using both activation functions approaches. Figure 10 depicts that this comparison is performed using two LSTMs: `Model HW1` uses the original activation functions whereas `Model HW2` was trained using our hard activation function. Predictions are evaluated on a per-patient basis using the most common error metrics, Equations (12)–(18), respectively: mean squared error (MSE), RMSE, mean absolute error (MAE), R-squared (R^2), correlation coefficient (CC), fit (FIT), and

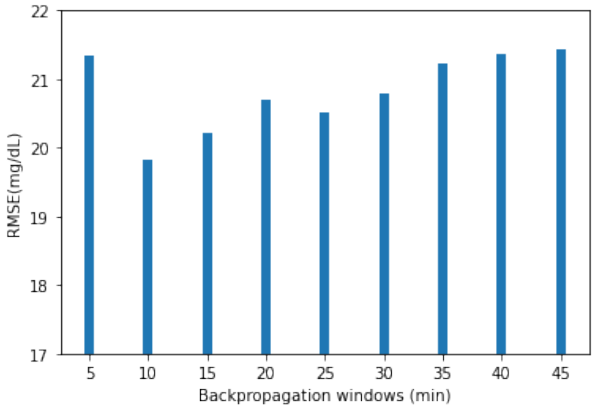


Fig. 10: Error of backpropagation windows, from 5 to 45 minutes

mean absolute relative difference (MARD), respectively. We denote the actual BG value at time t as $\mathrm{bg}(t)$, the actual future BG value ph minutes ahead of time t as $\mathrm{bg}_{ph}(t) = \mathrm{bg}(t+ph)$, and the predicted BG ph minutes ahead of time t as $\widehat{\mathrm{bg}}_{ph}(t)$. In them, n is the number of predictions per patient, and $\overline{\mathrm{bg}}_{ph} = \frac{1}{n}\sum_{t=1}^{n}\mathrm{bg}_{ph}(t)$ and $\overline{\widehat{\mathrm{bg}}}_{ph} = \frac{1}{n}\sum_{t=1}^{n}\widehat{\mathrm{bg}}_{ph}(t)$ are the mean values. On the other hand, in clinical practice, physicians usually plot predictions versus actual values using the Parkes error grid (PEG) [23]. This graph has five zones (A to E) to bound prediction accuracy. These zones are set by taking into account the treatment applied for a corresponding BG level. While zone A will always correspond to correct treatment, zone E corresponds to a hyperglycemia treatment while the patient will suffer from hypoglycemia, or vice versa.

This work is licensed under a Creative Commons Attribution 4.0 License. For more information, see https://creativecommons.org/licenses/by/4.0/

This article has been accepted for publication in IEEE Journal of Biomedical and Health Informatics. This is the author's version which has not been fully edited and content may change prior to final publication. Citation information: DOI 10.1109/JBHI.2023.3300511

$$\mathrm{MSE}_{ph} = \frac{1}{n}\sum_{t=1}^{n}\left(\widehat{\mathrm{bg}}_{ph}(t) - \mathrm{bg}_{ph}(t)\right)^2 \tag{12}$$

$$\mathrm{RMSE}_{ph} = \sqrt{\mathrm{MSE}_{ph}} \tag{13}$$

$$\mathrm{MAE}_{ph} = \frac{1}{n}\sum_{t=1}^{n}|\widehat{\mathrm{bg}}_{ph}(t) - \mathrm{bg}_{ph}(t)| \tag{14}$$

$$\mathrm{R}^2_{ph} = 1 - \frac{\sum_{t=1}^{n}\left(\widehat{\mathrm{bg}}_{ph}(t) - \mathrm{bg}_{ph}(t)\right)^2}{\sum_{t=1}^{n}\left(\mathrm{bg}_{ph}(t) - \overline{\mathrm{bg}}_{ph}\right)^2} \tag{15}$$

$$\mathrm{CC}_{ph} = \frac{\sum_{t=1}^{n}\left(\widehat{\mathrm{bg}}_{ph}(t) - \overline{\widehat{\mathrm{bg}}}_{ph}\right)\left(\mathrm{bg}_{ph}(t) - \overline{\mathrm{bg}}_{ph}\right)}{\sqrt{\sum_{t=1}^{n}\left(\widehat{\mathrm{bg}}_{ph}(t) - \overline{\widehat{\mathrm{bg}}}_{ph}\right)^2 \sum_{t=1}^{n}\left(\mathrm{bg}_{ph}(t) - \overline{\mathrm{bg}}_{ph}\right)^2}} \tag{16}$$

$$\mathrm{FIT}_{ph} = 1 - \frac{\frac{1}{n}\sum_{t=1}^{n}\left|\widehat{\mathrm{bg}}_{ph}(t) - \overline{\mathrm{bg}}_{ph}\right|}{\frac{1}{n}\sum_{t=1}^{n}\left|\mathrm{bg}_{ph}(t) - \overline{\mathrm{bg}}_{ph}\right|} \tag{17}$$

$$\mathrm{MARD}_{ph} = \frac{1}{n}\cdot\sum_{t=1}^{n}\frac{|\widehat{\mathrm{bg}}_{ph}(t) - \mathrm{bg}_{ph}(t)|}{\mathrm{bg}_{ph}(t)} \tag{18}$$

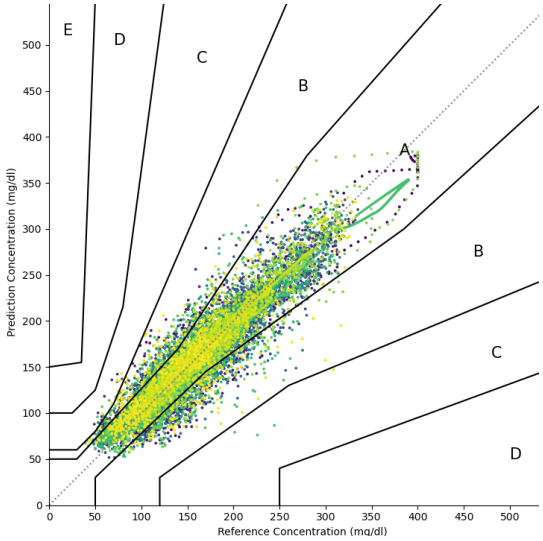
Table II shows the error metrics to compare both activation function approaches' impact on the prediction accuracy. We can extract similar conclusions from the RMSE, the MSE, and the MAE. With an RMSE of $19.92\,\mathrm{mg\,dL^{-1}}$ and $20.02\,\mathrm{mg\,dL^{-1}}$, both approaches predict well enough to be part of a treatment. R^2 can be understood as the explainability of the NN. Therefore, we can say that, in both approaches, the NN explains 90% of the data variability.

Regarding CC and FIT, they compare predicted values against actual values, so the highest performance corresponds to a value of 1 for both metrics; once again, they present identical values for both models. Finally, MARD is a well-known metric for CGM systems accuracy. It measures the difference between actual and predicted values; therefore, lower MARD values correspond to more accurate predictions. For both approaches, the MARD values indcate that the predictions are sufficiently accurate. When we compare both activation function approaches, all metrics overlap; thus, we can say that they are so close that the differences are due to the random initialization of the learning algorithm.

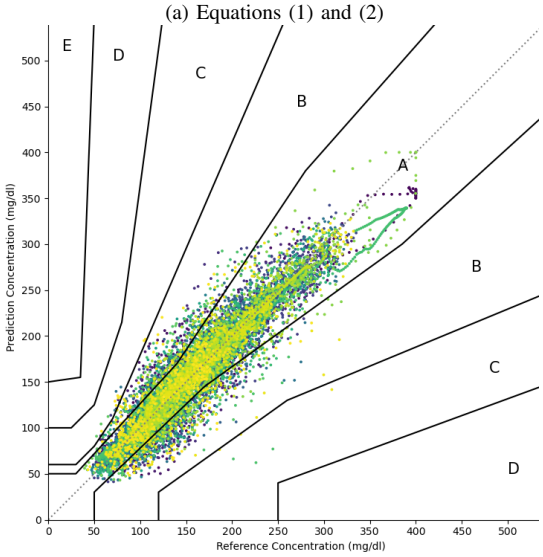
TABLE II: Results of the NN with Equations (1) and (2), and the NN with Equations (10) and (11).

Metrics	Activation functions	
	Equations (1) and (2)	Equations (10) and (11)
RMSE ($\mathrm{mg\,dL^{-1}}$)	19.82 ± 2.90	20.02 ± 3.19
MSE ($\mathrm{mg^2\,dL^{-1}}$)	392.78 ± 115.00	400.60 ± 127.83
MAE ($\mathrm{mg\,dL^{-1}}$)	14.24 ± 2.59	13.55 ± 2.38
R2	0.90 ± 0.02	0.90 ± 0.03
CC	0.95 ± 0.01	0.95 ± 0.01
FIT	0.72 ± 0.04	0.73 ± 0.05
MARD	0.10 ± 0.01	0.09 ± 0.01

Figure 11 shows that from the clinical point of view, there is no difference between both activation function approaches. Both PEGs are very similar, with a $97.38\,\%$ points within zones A and B for the NN with non-linear activation functions and a $99.84\,\%$ of points for the model with hard activation functions. Because of all the previous results, we conclude that there is no difference in the accuracy of both activation function approaches; hence, the rest of the experiments are done with the hard activation functions since the results are very similar.



(a) Equations (1) and (2)



(b) Equations (10) and (11)

Fig. 11: Parkes Error Grid of the NN with Equations (1) and (2), and the NN with Equations (10) and (11).

Before designing the hardware implementation, we apply quantization [24] to define the fixed-point format with the minimum length that ensures a reasonable accuracy. As a starting fixed-point format, in `Model HW1`, we use the 3Q12 fixed-point format to implement all the operations, where three bits are reserved for the integer part, 12 bits for the decimal part, and 1 bit for the sign, with a total of 16 bits. This format meets the requirements of numerical range word precision. Figure 12 shows the RMSE results for the different word lengths with different bits for the integer part, where each color defines the number of integer bits without the sign; i.e., a red point in the 13-bit length column corresponds to the 4Q8 format. For `Model HW2`, we select 2Q7, marked with a black arrow. Even though the fixed-point format adds error to the prediction, $25.26\,\mathrm{mg\,dL^{-1}}$, Figure 13 shows that there are few differences from the clinical point of view. The PEG of the NN using the fixed-point format is similar to those illustrated in Figure 11 with a $98.21\,\%$ of predictions within

This work is licensed under a Creative Commons Attribution 4.0 License. For more information, see https://creativecommons.org/licenses/by/4.0/

This article has been accepted for publication in IEEE Journal of Biomedical and Health Informatics. This is the author's version which has not been fully edited and content may change prior to final publication. Citation information: DOI 10.1109/JBHI.2023.3300511

zones A and B. Thus, we can use the 2Q7 fixed-point format for `Model HW2` since there are no differences from the clinical point of view. These results also serve to inspect the numerical stability of the LSTM. LSTM parameters are estimated using floating-point models during the training stage and then quantized, inducing a modification (quantization error) on the original value. Note this quantization error affects all the parameters of the LSTM. Figure 12 shows that the LSTM errors increase significantly when the quantization noise is high (for those fixed formats with few fractional bits –left points in figure— or with only 1 bit in the integer part –format 1Q). However, under low quantization noise, the prediction error remains low. Indeed, it is negligible from a clinical perspective.

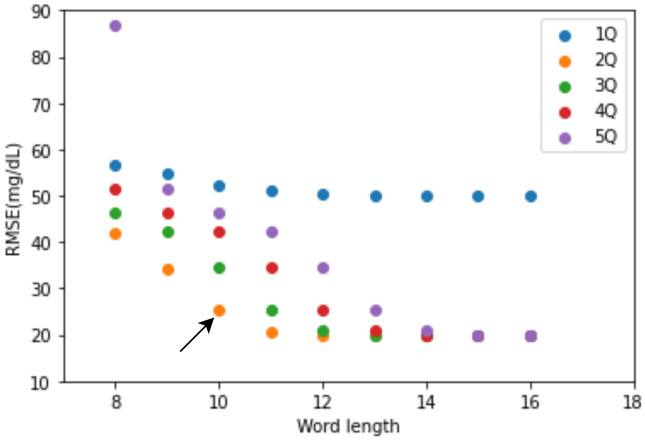


Fig. 12: Error comparison for different fixed point formats.

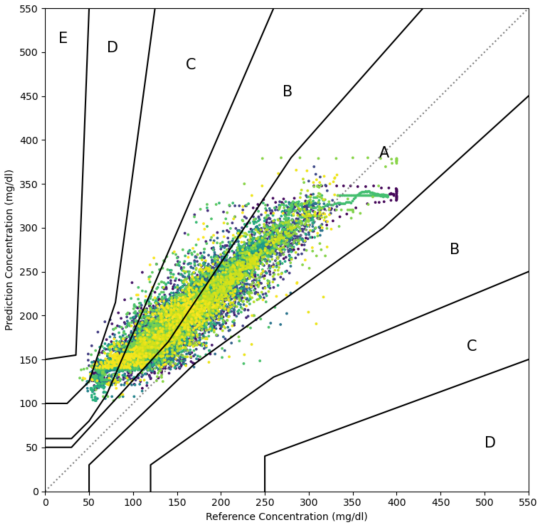


Fig. 13: Fixed-point 2Q7 format Parkes Error Grid.

Other techniques also reduce energy consumption and complexity, such as weight pruning. This optimization technique is meant to be used in large NNs with thousands of trainable parameters in which more than 50% of the weights are pruned [25]. As the size of the neural network increases, the accuracy is less affected. The NN for blood glucose prediction is too small for weight pruning . In our experiments, with 206 trainable parameters, weight pruning decreases the prediction accuracy: pruning 10% of the weights increases error beyond acceptable for a clinical device. For this reason, we discard the application of this optimization technique.

Table III summarizes the optimization techniques used by showing the key differences between `Model HW1` and `Model HW2`.

TABLE III: Key differences between `Model HW1` and `Model HW2`.

	`Model HW1`	`Model HW2`
Activation function	tanh	hard tanh
Recurrent activation function	sigmoid	hard sigmoid
Arithmetic	3Q12	2Q7
Time window	5 min	10 min

C. Hardware Implementation LSTM

Finally, we implement `Model HW1` and `Model HW2` to know the impact of all energy optimization methods applied to `Model HW2`, and compare with the same model running on smartphones.

First, we compare their area, indicating the model's complexity in terms of LBs. Table IV shows the LB usage compared with the total amount of blocks within the Virtex-7 FPGA. Moreover, `Model HW2`'s area, with 3573 LBs, is one-third of `Model HW1`'s area, with 11034 LBs. In addition, for `Model HW1`, we need digital signal processing elements (DSP), while for the `Model HW2`, they are not required. These DSPs are used for complex arithmetic operations. For all these reasons, we conclude that `Model HW2` circuit is less complex, which may lead to less energy consumption. Virtex-7 XC7VX485T-2FFG1761C device fits into a package of 35x35 mm. This package contains all the logic elements plus 350 I/O pins, 56x 12.5 Gb/s Low-Power Gigabit Transceivers with their ports, and 4x Integrated blocks for PCI Express with their ports. `Model HW1` requires less than 3% of the combinational elements and less than 1% of the FFs, and neither Transceivers nor PCIe blocks. So, a chip in 28nm technology with a size lower than 3x3 mm could contain all the logic of this design. In summary, both hardware LSTMs have a small area usage of the whole FPGA area allowing their implementation on a smaller and cheaper FPGA that could be carried as a complement to any wearable device without additional costs.

TABLE IV: Logic block usage of the hardware LSTMs.

Logic Blocks	`Model HW1`	`Model HW2`	Virtex-7
LUT	8486	2384	303600
LUTRAM	213	108	130800
FF	2281	1043	607200
BRAM	5	5	1030
DSP	22	0	2800
IO	44	32	700
BUFG	1	1	32

Secondly, we compare their performance by analyzing their critical path delay and the number of clock cycles needed for one prediction. The critical path is the path with the maximum delay between synchronous memory elements, and its delay defines the minimum period of the clock. In addition to the number of clock cycles needed for one prediction, this period determines the total time taken by the hardware LSTM to make a prediction. Table V shows the minimum period (T_{min}), the number of cycles needed for one prediction, and the total time taken for a single prediction. Here is where the hard activation functions have the greatest impact; T_{min} is nearly reduced by a factor of 6.40; while `Model HW1` requires 56.22 ns, `Model HW2` requires 8.78 ns. Also, the number of cycles per prediction is reduced; in this case, with 1602 cycles, the number of cycles is divided by 2.87, as the `Model HW1` needs 4592 cycles. Hence, the total prediction time for the `Model HW1` is 260 µs and the time needed by `Model HW2` is 14 µs, ×18.57 faster.

This work is licensed under a Creative Commons Attribution 4.0 License. For more information, see https://creativecommons.org/licenses/by/4.0/

This article has been accepted for publication in IEEE Journal of Biomedical and Health Informatics. This is the author's version which has not been fully edited and content may change prior to final publication. Citation information: DOI 10.1109/JBHI.2023.3300511

Next, we have analyzed the energy consumption, which is the most critical metric for this system to be wearable. It determines the system's autonomy, size, and weight due to the energy consumption impacts on the battery needed by the wearable device. Table V shows the energy consumption for both hardware models. The energy consumption is the sum of the static energy, the energy needed to keep the FPGA on, and the dynamic energy. Thus, it depends on FPGA's size; and the dynamic energy, which is the energy consumed when making a prediction. `Model HW2` needs 4950 nJ per prediction, while `Model HW1` needs 76 594 nJ. This means that our approaches reduces $\times$15.5 the energy consumption overall. The energy-consumption reduction and performance improvement of `Model HW2` is built notably upon the improvements provided by the new activation functions. The hard activation module, which contains the two hard activation functions, consumes 0.72 nJ per operation, whereas the original activation function module consumes 23.20 nJ per operation, a reduction of $\times$32. Regarding performance, the hard activation module requires 6 clock cycles to compute an operation, and it is capable of computing at a maximum clock frequency of 787 MHz, whereas the CORDIC requires up to 29 clock cycles and the maximum clock frequency is 18 MHz.

TABLE V: Comparison of the performance and energy consumption of the four NN: `Model HW1`, `Model HW2`, Android NN and iOS NN. Each NN has been evaluated on a different device implemented on different technologies.

	`Model HW1`	`Model HW2`	Android	iOS
Technology node (nm)	28	28	7	5
T_{min} (ns)	56.22	8.78	—	—
Cycles/prediction	4592	1602	—	—
Prediction time (μs)	260	14	200	193
Power (W)	0.28	0.31	0.83	0.18
Dynamic Power (W)	0.04	0.06	—	—
Energy/prediction (nJ)	76 594	4950	5 048 736	34 763
LB Scaled Energy (nJ)	14 084	1434	—	—

CGMs are commonly integrated with smartphone apps. The computational power of today's smartphones is enormous, there is a vast user base that uses them daily, and, in the case of the iOS devices, they provide a wearable AI accelerator. Hence, smartphones seem to be good candidates to run edge NNs, through software, for BG prediction. Therefore, we compare the energy consumption of the hardware implementations with the proposed NN in Table I running on an Android and an iOS device. The NN models deployed on these devices are built using tools provided by Android and Apple that translate the Python description into a highly optimized code for each hardware. For such a reason, we code the hard activation functions on the mobile implementation but rely on the Android and iOS translation tools optimizers to provide the most efficient ML code for the corresponding smartphone..

On the one hand, the Android NN is tested on a OnePlus Nord smartphone, running Android Oxygen 11.1.10.10.AC01BA operating system, which has a Qualcomm® Snapdragon™ 765G based on the Samsung's 7 nm Low Power Plus (7LPP) process technology at 2.40 GHz and a battery of 4115 mA h with a nominal battery voltage of 3.87 V. This device has no specific ML hardware. The NN is developed with TensorFlow Lite: a set of tools designed to implement ML techniques on smartphones, embedded devices, and the internet of things, optimized for power consumption and other constraints. Hence, we design an app that implements the NN using Android Studio and Java™. To monitor the energy consumption, we use Battery Historian, a Google tool to analyze app performance. It monitors energy consumption, CPU usage, time spent on each of the apps, and other useful parameters for developers [26]. This allows the energy consumption of the NN to be analyzed separately. Even though the Android implementation of the NN is more energy-efficient than the Python version, thanks to TensorFlow-lite, it needs, averaged over 10000 samples, 5 048 736 nJ per prediction. This consumption is around $\times$660 greater than `Model HW1` and $\times$1020 greater than `Model HW2`. Hence, using a portable device with generic hardware greatly impacts energy consumption compared to specialized hardware for NNs.

On the other hand, the iOS NN is tested on an Apple® iPhone 13 Pro™ running under the iOS 15.4.1 operating system, which has an A15 Bionic chipset based on the TSMC's 5 nm (N5P) process technology, which is the current cutting-edge semiconductor technology node, running at 3.23 GHz and a battery of 3095 mA h with a nominal voltage of 3.91 V. This chipset comprises specialized ML cores named Apple's Neural Engines, designed to accelerate AI operations, lowering the processing time without compromising its accuracy. The NN is ported from Python to Swift programming language with the Python library CoreML. The app is designed with XCode, a set of tools that provides user interface design, coding, testing, and debugging. We use the iOS built-in battery usage app to monitor the energy consumption of the iOS app that runs the NN. With an energy consumption, averaged over 10000 samples, of 34 763 nJ per prediction, the AI accelerators designed by Apple lower the energy consumption to the extent that it is better than `Model HW1`. Nevertheless, the `Model HW2` has an energy consumption of $\times$7 lower.

When we compare the energy consumption, the iOS NN is more energy efficient than the `Model HW1`. However, we use the static energy of the whole FPGA while using a small area; Table IV shows that `Model HW1` occupies a 1.06 % and `Model HW2` occupies a 0.34 % of the FPGA area. For this reason, most static energy is used to power LBs that the models will not use.

Even though we use the Xilinx Virtex-7 FPGA to obtain the results, it is not the target FPGA. Our main goal is to make a wearable device, so the target FPGA must be the smallest one whose area is fully utilized, as much as possible, by the hardware implementation. For this reason, we scale the static energy by the area that each hardware NN occupies. The dynamic power of `Model HW1` is 0.04 W, see Table V, so the scaled static energy is 670 nJ and, therefore, the scaled energy consumption of `Model HW1` is 14 084 nJ per prediction. For the `Model HW2`, the dynamic power is 0.06 W, so the scaled static energy is 12 nJ with a total scaled energy consumption of 1434 nJ per prediction. Hence, if we use an FPGA mainly occupied by each hardware NN circuit, they consume less energy per prediction than the other Android and iOS NN approaches.

Note that we are comparing chipsets with different technology nodes. With the information provided by TSMC [27], the energy consumption of the 7N technology is 7.30 % and the energy consumption of the N5P technology is 4.37 % of the 28HPL technology. This fact denotes the impact in energy consumption of the technology leap between the FPGA transistor technology and the smartphones' CPU transistor technologies.

In addition, Figure 14 shows the study of the energy consumption and accuracy concerning the time window size. Our main goal is to design an LSTM-based NN for glucose prediction that, in addition, is wearable. With the optimization approaches that we have made, the error grows from 19.82 mg dL^{-1} to 25.26 mg dL^{-1} and, in combination with other devices that people with diabetes commonly use, such as CGMs or insulin pumps, the different errors accumulate. For this reason, we maintain the 10-minute window since we need

This work is licensed under a Creative Commons Attribution 4.0 License. For more information, see https://creativecommons.org/licenses/by/4.0/

This article has been accepted for publication in IEEE Journal of Biomedical and Health Informatics. This is the author's version which has not been fully edited and content may change prior to final publication. Citation information: DOI 10.1109/JBHI.2023.3300511

that the whole processing chain (from sensor to final prediction) provides good predictions and good performance. If the accuracy were not a critical issue, the 30-minute window could increase the error by $0.92\,\mathrm{mg\,dL^{-1}}$ and reduces the energy consumed by up to two-thirds. In any case, the 10-minute window energy consumption shows promising results.

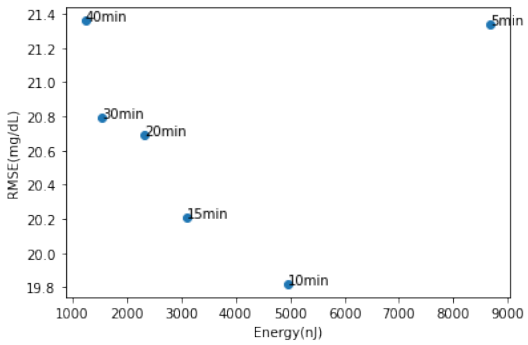


Fig. 14: Prediction accuracy of the proposed model and energy consumption of `Model HW2` for different backpropagation windows, from 5 to 45 minutes.

Finally, in Table VI we compare `Model HW1` and `Model HW2` with three state-of-the-art LSTM-based hardware models to obtain a global view of its efficiency. Two of them use single-precision IEEE754 FP arithmetic [28], [29], and the third one with 7Q8 format [30]. As the models have different architectures, we compare their performance as the number of FP operations per second (GFLOPS) and their energy efficiency as the number of FP operations per joule (GFLOP/J). In this case, we use the whole FPGA consumption as it is done in the rest of the studies. Note that the NNs also have different word formats to implement the arithmetic operations. `Model HW1` is the slower model, with 0.022 GFLOPS. Nevertheless, performance is not an issue for glucose prediction since there are at least five minutes between new data entries. Its efficiency is better than the models of other studies by little, with 0.402 GFLOP/J. `Model HW2` is at mid-range of performance with 0.404 GFLOPS, but, in terms of efficiency, it is, by far, the best model of the comparison; with 1.142 GFLOP/J, it is between $\times 2.84$ and $\times 7.82$ greater than the rest of the models. Theses values prove that using the same integrated circuit technology, very substantial improvements in energy consumption can be obtained in AI accelerators by applying specific optimizations aimed at this purpose, i.e., there is room for improvement for low-power implementations in AI accelerators.

TABLE VI: Comparison of state-of-the-art FPGA-based LSTM models. Technology node refers to the integrated circuit fabrication technology; word format refers to the type of arithmetic, either fixed point or floating point using single-precision 32-bit IEEE754. Power is measured in watts, performance in GFLOPS, and energy efficiency in GFLOP per joule.

Model	HW1	HW2	[28]	[29]	[30]
FPGA	Virtex-7 XC7VX485T	Virtex-7 XC7VX485T	Zynq XC7Z020	Virtex-7 XC7VX485T	Zynq XC7Z020
Technology node (nm)	28	28	28	28	28
Word format	3Q12	2Q7	IEEE754	IEEE754	7Q8
Power (W)	0.28	0.31	1.932	19.63	1.942
Performance (GFLOPS)	0.022	0.404	0.360	7.62	0.284
Efficiency (GFLOP/J)	0.402	1.142	0.186	0.388	0.146

V. Conclusions

We have designed a parametrizable and specialized hardware device that implements a NN hardware model, named `Model HW2`, for BG prediction on a wearable medical device. `Model HW2` is designed after an optimization process aimed to reducing the area and power consumption, and increasing the performance.

Before and after energy optimization, the NN prediction performance is tested using the most common analysis tools and metrics in the literature for BG prediction. It is trained using the OhioT1DM Dataset for a $30\,\mathrm{min}$ prediction horizon. The NN prediction performance, before and after energy optimization, is similar: their metrics' values are very close and have overlapping confidence intervals. Hence, the proposed NN, without optimization, has an RMSE of $19.82\,\mathrm{mg\,dL^{-1}}$ while the NN with energy-optimized activation functions has an RMSE of $20.02\,\mathrm{mg\,dL^{-1}}$ and the NN with energy-optimized activation functions and fixed-point arithmetic has an RMSE of $33.72\,\mathrm{mg\,dL^{-1}}$. Nevertheless, these differences are not significant from a clinical perspective: the PEG analysis shows that the NN has more than $98\,\%$ of the points within zones A and B with each optimization approach. Thus, the energy-optimized NN can be used in clinical practice.

Its energy consumption is compared with an implementation that best mimics the behavior of the originally proposed NN, `Model HW1`, and software counterparts running on smartphones with AI accelerators. Both hardware implementations, implemented in a Xilinx Virtex-7 FPGA VC707 Evaluation Kit, are evaluated and compared in terms of area, performance, and energy consumption with the NN running in an Android smartphone with a general-purpose CPU and in an iOS smartphone with AI application-specific CPU. As expected, `Model HW2` is the implementation with the lowest energy consumption and processing time compared to `Model HW1`and the smartphone implementations. Hence, the approaches that reduce the energy consumption also lower the latency and LB usage without losing accuracy from the clinical point of view.

We also compare the hardware implementation efficiency (GFLOP/J) of `Model HW1` and `Model HW2` regarding other state-of-the-art NNs implemented on FPGAs, even though they are not aimed to be wearable. `Model HW2`, outperforms all other implementations with a higher number of GFLOP per joule. This experiment proves that there is room for improvement in the energy consumption reduction of edge wearable NN by optimizing different hyperparameters of the NN taking into account the hardware implementation target.

Both hardware implementations occupy a minimal portion of the FPGA area, so most of the static energy is wasted to power-on FPGA components that are not used by the models, reducing their energy consumption efficiencies. Therefore, using an FPGA with a size that fits the model area requirements lowers their energy consumption, outperforming smartphone implementations' energy-consumption metrics.

The use of custom AI application-specific hardware is mandatory for designing NNs that can run on battery-powered embedded systems. This proves that the best option is to use ad hoc hardware to implement NNs for wearable medical devices. The iOS version that uses software running on AI-specific hardware trails far behind the ad hoc hardware implementations, and the Android version that uses general-purpose CPUs is the worst option. Regarding the flexibility and facility of deployment of the software implementations, with the usage of parametrizable components, automatic high-level synthesis tools, and reconfigurable logic, the FPGA implementations of NNs can be tailored and deployed on-demand as much as the software implementations.

This work is licensed under a Creative Commons Attribution 4.0 License. For more information, see https://creativecommons.org/licenses/by/4.0/

This article has been accepted for publication in IEEE Journal of Biomedical and Health Informatics. This is the author's version which has not been fully edited and content may change prior to final publication. Citation information: DOI 10.1109/JBHI.2023.3300511

The application of emerging approaches such as compute-in-memory, CIM, [31], and transformers [10] could lead to even more significant improvements in energy efficiency and performance, respectively. Nevertheless, CIM is not straightforwardly applicable to our approach due to: (i) different paradigms in NN modeling (execution of the arithmetic operations that model layer behavior in CIM vs. actual implementation of each cell in our proposal) and (ii) dataset size suitable for each paradigm (medium/big datasets for CIM) vs. little/medium datasets in our case). How to combine both approaches remains open research. Regarding transformers, they have shown a better performance than LSTM. However, it is an open discussion if they would provide such a performance under the architectural constraints imposed by the severe energy consumption constraints of wearable devices.

In summarising, using ad hoc hardware NN implementations would allow taking advantage of the benefits of edge computing, reducing its energy consumption, size, and costs to obtain a wearable medical device for NN-based BG prediction.

REFERENCES

[1] S. Ghosh and A. Collier, "Section 1 - diagnosis, classification, epidemiology and biochemistry," in *Churchill's Pocketbook of Diabetes (Second Edition) (Second Edition)*. Oxford: Churchill Livingstone, 2012, pp. 1–49. DOI: https://doi.org/10.1016/B978-0-443-10081-9.00008-7.

[2] M. Mayo and T. Kounty, "Neural multi-class classification approach to blood glucose level forecasting with prediction uncertainty visualisation," in *5th International Workshop on Knowledge Discovery in Healthcare Data*, Chicago, IL, USA, 2005.

[3] J. E. Gerich, "Control of glycaemia," *Baillieres Clin Endocrinol Metab*, vol. 7, no. 3, pp. 551–86, 1993. DOI: 10.1016/s0950-351x(05)80207-1.

[4] Mayo Clinic, *Diabetic hypoglycemia*, Web Page, 2020. [Online]. Available: https://www.mayoclinic.org/diseases-conditions/diabetic-hypoglycemia/symptoms-causes/syc-20371525.

[5] I. Hidalgo *et al.*, "Identification of models for glucose blood values in diabetics by grammatical evolution," in 2018, pp. 367–393. DOI: 10.1007/978-3-319-78717-6_15.

[6] F. Tena, O. Garnica, J. Lanchares, and J. I. Hidalgo, "Ensemble models of cutting-edge deep neural networks for blood glucose prediction in patients with diabetes," *Sensors*, vol. 21, no. 21, p. 7090, 2021. DOI: 10.3390/s21217090.

[7] C. Meijner and S. Persson, "Blood glucose prediction for type 1 diabetes using machine learning long short-term memory based models for blood glucose prediction," Thesis, 2017. [Online]. Available: https://hdl.handle.net/20.500.12380/251317.

[8] S. Hochreiter and J. Schmidhuber, "Long short-term memory," *Neural Computation*, vol. 9, no. 8, pp. 1735–1780, 1997. DOI: 10.1162/neco.1997.9.8.1735. [Online]. Available: https://doi.org/10.1162/neco.1997.9.8.1735.

[9] M. Satyanarayanan, "The emergence of edge computing," *Computer*, vol. 50, no. 1, pp. 30–39, 2017. DOI: 10.1109/MC.2017.9.

[10] A. Vaswani *et al.*, *Attention is all you need*, 2017. DOI: 10.48550/ARXIV.1706.03762. [Online]. Available: https://arxiv.org/abs/1706.03762.

[11] Q. Wen *et al.*, *Transformers in time series: A survey*, 2022. DOI: 10.48550/ARXIV.2202.07125. [Online]. Available: https://arxiv.org/abs/2202.07125.

[12] R. Joost and R. Salomon, "Advantages of fpga-based multiprocessor systems in industrial applications," in *31st Annual Conference of IEEE Industrial Electronics Society, 2005. IECON 2005.*, 6 pp. DOI: 10.1109/IECON.2005.1568946.

[13] K. Gurney, *An Introduction to Neural Networks*. USA: Taylor & Francis, Inc., 1997.

[14] S. Mirshekarian, R. Bunescu, C. Marling, and F. Schwartz, "Using lstms to learn physiological models of blood glucose behavior," in *2017 39th Annual International Conference of the IEEE Engineering in Medicine and Biology Society (EMBC)*, pp. 2887–2891. DOI: 10.1109/EMBC.2017.8037460.

[15] D. Lin, S. Talathi, and S. Annapureddy, "Fixed point quantization of deep convolutional networks," in *Proceedings of The 33rd International Conference on Machine Learning*, B. Maria Florina and Q. W. Kilian, Eds., vol. 48, PMLR, pp. 2849–2858. [Online]. Available: https://proceedings.mlr.press/v48/linb16.html.

[16] Y. Zheng, H. Yang, Y. Jia, and Z. Huang, "Permlstm: A high energy-efficiency lstm accelerator architecture," *Electronics (Switzerland)*, vol. 10, no. 8, 2021. DOI: 10.3390/electronics10080882.

[17] F. Chollet *et al.* "Keras." (2015), [Online]. Available: https://github.com/fchollet/keras.

[18] *Lowering power at 28 nm with xilinx 7 series fpgas*, Xilinx WP389 (v1.1.1), Feb. 2012.

[19] C. Marling and R. Bunescu, "The ohiot1dm dataset for blood glucose level prediction: Update 2020," 2020. [Online]. Available: http://smarthealth.cs.ohio.edu/bglp/OhioT1DM-dataset-paper.pdf.

[20] S. McKinley and M. Levine, "Cubic spline interpolation," *College of the Redwoods*, vol. 45, no. 1, pp. 1049–1060, 1998.

[21] M. Berger and D. Rodbard, "Computer Simulation of Plasma Insulin and Glucose Dynamics After Subcutaneous Insulin Injection," *Diabetes Care*, vol. 12, no. 10, pp. 725–736, Nov. 1989. DOI: 10.2337/diacare.12.10.725. eprint: https://diabetesjournals.org/care/article-pdf/12/10/725/438354/12-10-725.pdf. [Online]. Available: https://doi.org/10.2337/diacare.12.10.725.

[22] D. Kingma and J. Ba, "Adam: A method for stochastic optimization," *International Conference on Learning Representations*, 2014.

[23] J. L. Parkes, S. L. Slatin, S. Pardo, and B. H. Ginsberg, "A new consensus error grid to evaluate the clinical significance of inaccuracies in the measurement of blood glucose," *Diabetes Care*, vol. 23, no. 8, p. 1143, 2000. DOI: 10.2337/diacare.23.8.1143.

[24] R. Gray and D. Neuhoff, "Quantization," *IEEE Transactions on Information Theory*, vol. 44, no. 6, pp. 2325–2383, 1998. DOI: 10.1109/18.720541.

[25] M. Zhu and S. Gupta, "To prune, or not to prune: Exploring the efficacy of pruning for model compression," 2017. DOI: 10.48550/ARXIV.1710.01878. [Online]. Available: https://arxiv.org/abs/1710.01878.

[26] W. Oliveira, R. Oliveira, and F. Castor, "A study on the energy consumption of android app development approaches," in *2017 IEEE/ACM 14th International Conference on Mining Software Repositories (MSR)*, pp. 42–52. DOI: 10.1109/MSR.2017.66.

[27] T. S. M. Company, *5nm technology*, Jun. 2021. [Online]. Available: https://www.tsmc.com/english/dedicatedFoundry/technology/logic/l_5nm.

This work is licensed under a Creative Commons Attribution 4.0 License. For more information, see https://creativecommons.org/licenses/by/4.0/

This article has been accepted for publication in IEEE Journal of Biomedical and Health Informatics. This is the author's version which has not been fully edited and content may change prior to final publication. Citation information: DOI 10.1109/JBHI.2023.3300511

[28] U. Yoshimura, T. Inoue, A. Tsuchiya, and K. Kishine, "A method for accelerating the inference process of FPGA-based LSTM for biometric systems," *IEIE Transactions on Smart Processing & Computing*, vol. 10, no. 5, pp. 416–423, Oct. 2021. DOI: 10.5573/ieiespc.2021.10.5.416. [Online]. Available: https://doi.org/10.5573/ieiespc.2021.10.5.416.

[29] Y. Guan, Z. Yuan, G. Sun, and J. Cong, "Fpga-based accelerator for long short-term memory recurrent neural networks," in *2017 22nd Asia and South Pacific Design Automation Conference (ASP-DAC)*, 2017, pp. 629–634. DOI: 10.1109/ASPDAC.2017.7858394.

[30] A. X. M. Chang, B. Martini, and CulurcielloEugenio, *Recurrent neural networks hardware implementation on fpga*, 2015. DOI: 10.48550/ARXIV.1511.05552. [Online]. Available: https://arxiv.org/abs/1511.05552.

[31] S. Yu, X. Sun, X. Peng, and S. Huang, "Compute-in-memory with emerging nonvolatile-memories: Challenges and prospects," in *2020 IEEE Custom Integrated Circuits Conference (CICC)*, 2020, pp. 1–4. DOI: 10.1109/CICC48029.2020.9075887.

This work is licensed under a Creative Commons Attribution 4.0 License. For more information, see https://creativecommons.org/licenses/by/4.0/