Enabling Printed Multilayer Perceptrons Realization via Area-Aware Neural Minimization

Argyris Kokkinis[1], Georgios Zervakis[1], Kostas Siozios[1], Mehdi B. Tahoori[1], and Jörg Henkel[1]

[1]Affiliation not available

January 20, 2025

Abstract

Printed Electronics (PE) set up a new path for the realization of ultra low-cost circuits that can be deployed in everyday consumer goods and disposables. In addition, PE satisfy requirements such as porosity, flexibility, and conformity. However, the large feature sizes in PE and limited device counts incur high restrictions and increased area and power overheads, prohibiting the realization of complex circuits. As a result, although printed Machine Learning (ML) circuits could open new horizons and bring "intelligence" in such domains, the implementation of complex classifiers, as required in target applications, is hardly feasible. In this paper, we aim to address this and focus on the design of battery powered printed Multilayer Perceptrons (MLPs). To that end, we exploit fully-customized circuit (bespoke) implementations, enabled in PE, and propose a hardware-aware neural minimization framework dedicated for such customized MLP circuits. Our evaluation demonstrates that, for up to 3% accuracy loss, our co-design methodology enables, for the first time, battery-powered operation of complex printed MLPs.

Accepted for publication by IEEE Transactions on Computers, December 2024 10.1109/TC.2024.3524076

Enabling Printed Multilayer Perceptrons Realization via Area-Aware Neural Minimization

Argyris Kokkinis, Georgios Zervakis,
Kostas Siozios, *Senior Member, IEEE,*
Mehdi B. Tahoori, *Fellow, IEEE,*
Jörg Henkel, *Fellow, IEEE*

Abstract—Printed Electronics (PE) set up a new path for the realization of ultra low-cost circuits that can be deployed in every-day consumer goods and disposables. In addition, PE satisfy requirements such as porosity, flexibility, and conformity. However, the large feature sizes in PE and limited device counts incur high restrictions and increased area and power overheads, prohibiting the realization of complex circuits. As a result, although printed Machine Learning (ML) circuits could open new horizons and bring "intelligence" in such domains, the implementation of complex classifiers, as required in target applications, is hardly feasible. In this paper, we aim to address this and focus on the design of battery-powered printed Multilayer Perceptrons (MLPs). To that end, we exploit fully-customized circuit (bespoke) implementations, enabled in PE, and propose a hardware-aware neural minimization framework dedicated for such customized MLP circuits. Our evaluation demonstrates that, for up to 3% accuracy loss, our co-design methodology enables, for the first time, battery-powered operation of complex printed MLPs.

Index Terms—Approximate Computing, Co-design, Multilayer Perceptrons, Neural Minimization, Printed Electronics

1 Introduction

Printed electronics (PE) technology has attracted significant research interest and shows promise in various domains unreachable by traditional rigid silicon systems. These include smart packaging, disposables, the ten trillion market of fast-moving consumer goods (FMCG), in-situ monitoring, and low-end healthcare products [1], [2], [3]. These domains require ultra-low-cost fabrication, which is impractical for silicon-based technologies, and demand stretchability, porosity, flexibility, and conformality—features inherent to printed electronics but unachievable with silicon [2].

Printing technologies often rely on mask-less, portable, and additive manufacturing methods which can greatly reduce costs and production timelines. PE require low-cost equipment and refer to the fabrication technology that relies on printing processes, such as jet, screen or gravure printing [3]. In such printed technologies, functional solution-based materials are additively printed to form active (transistors, diodes) and passive (resistors, capacitors, interconnects) components for fully-printed electronics. However, this comes at much higher feature sizes (micrometers), low device counts (a few thousands gates), and lower frequencies (few Hz to~KHz) compared to silicon VLSI [4]. In this work, we consider the Electrolyte-Gated FET (EGFET) technology [3] due to its inexpensive additive-only manufacturing processes. Moreover, EGFETs boast favorable mobility characteristics and operate at low supply voltages, as they are capable of operating well below 1V [5]. This aligns well with battery-powered IoT application scenarios.

Despite the benefits of PE, their large feature sizes may lead to a unsustainable area and power overheads, making the realization of complex circuits, e.g., for Machine Learning (ML), extremely challenging and often unrealistic [2], [6], [7]. Still, they attract vast research and industrial interest and flexible systems have been successfully fabricated. In 2020, Ozer et al. [8] fabricated a natively flexible processing engine for odor recognition, and Weller et al. [9] printed an artificial neuron using EGFET technology. In 2021, ARM developed an extremely low-cost 32-bit processor using Thin Film Transistor (TFT) technology on a flexible plastic substrate for deployment on everyday objects, while 4- and 8-bit flexible microprocessors are fabricated with high yield in [10]. Leveraging the extremely low fabrication and non-recursive engineering (NRE) costs of PE that allow on-demand and in-situ printing, [2] proposed the design of bespoke printed ML circuits. These circuits are highly customized to a specific ML model, enabling extremely optimized and resource-efficient implementations [3], [11], which are mostly infeasible in lithography-based silicon systems due to their high associated costs [3]. Even FPGA and CGRA systems, that offer some customization, are limited by the architecture, multiplexing, and routing of the underlying fabric.

Printed ML represents an extreme subset within the TinyML domain. TinyML focuses on achieving low-power ML inference on resource-constrained devices, such as tiny microcontrollers, FPGAs, etc [12]. However, resource limitations in PE are orders of magnitude stricter compared to silicon-based TinyML systems, as PE requires all computations to fit within a few thousand gates. A customized approach to shrink the size of neural networks for FPGA deployment at the deep edge are presented in [13] However, the computational power of FPGAs far exceeds the capabilities of PE. Targeting a 16nm ASIC accelerator, [14] divides a CNN into two parts and runs the first part on a fully-customized accelerator but runs the second part on a on a generic programmable CNN hardware accelerator (e.g., NVDLA) inducing a large hardware complexity and overhead. In [15], a 28nm low-power, error-tolerant system

- *A. Kokkinis and K. Siozios are with the Department of Physics, Aristotle University of Thessaloniki, Thessaloniki 54124, Greece.*
- *G. Zervakis is with the Computer Engineering & Informatics Dept., University of Patras, Patras 26504, Greece.*
- *M. B. Tahoori and J. Henkel are with the Department of Computer Science, Karlsruhe Institute of Technology, Karlsruhe 76131, Germany.*

This work is partially supported by the European Union (ERC) and co-funded by the H.F.R.I call "Basic research Financing (Horizontal support of all Sciences)" under the National Recovery and Resilience Plan "Greece 2.0" (H.F.R.I. Project Number: 17048). Experiments run on the High Performance Computing Infrastructure of Aristotle University of Thessaloniki.
Corresponding Author: Argyris Kokkinis (arkokkin@auth.gr)
Manuscript received Jan. 17, 2023, revised July 11, 2024.

based on the ARM Cortex-M0 is presented for DNN acceleration. [15] saves power by reducing the supply voltage to 0.7V and incorporates specialized hardware for detecting timing violations. Still, such a system is very complex to be implemented in PE due to its increased area requirements.

Bespoke ML classifiers are designed in [2] by hardwiring the model's coefficients in the circuit implementation itself. Despite the high gains, [2] deduced that circuits such as Multilayer Perceptrons (MLPs) are very complex and focused on simple ML circuits (e.g., Decision Trees). Alternative computing paradigms such as approximate [16] and stochastic [17] computing could form promising solutions to mitigate the elevated hardware overheads of printed MLPs. Approximate and stochastic printed MLPs are designed in [6] and [7], respectively, delivering high area gains.

ML arguably forms a perfect match with approximate computing due to many factors [16]. In our work, we exploit this relation and combine it with the unmatched hardware-efficiency of bespoke implementations targeting battery-powered printed MLPs. Our holistic neural minimization framework integrates hardware-aware Neural Architecture Search (NAS), pruning, quantization, and weight sharing, along with voltage scaling at the hardware level. Our framework leverages the unique features and irregularities of bespoke circuits, tailoring these techniques specifically for printed MLPs. With our bespoke MLP area estimator, we apply these techniques in a printed-driven manner, addressing the implications of model-specific bespoke hardware on all optimization steps. While such simplification techniques are used in Deep Neural Networks (DNNs), our optimization objectives differ fundamentally. Our goal is to minimize hardware costs (area and power) to meet physical constraints such as printed battery limitations and end-product area requirements, while maintaining acceptable accuracy. Given the limited device counts and large feature sizes in PE, our primary objective is the feasibility of such systems, where even realizing simple MLPs with few neurons is challenging due to large on-chip area and power expenses [2]. This contrasts with DNN optimization for generic platforms (e.g., GPUs, TPUs), where customization is far more constrained compared to printed circuits.

The MLPs generated by our framework exhibit on average 6x less area and 5.2x lower power consumption than the state-of-the-art exact (un-minimized) baseline [2] for no accuracy loss and at the same operating conditions, while for only up to 3% accuracy loss we enable battery-powered operation of all the examined MLPs.

The novel contributions of our work are as follows:

1) We propose an efficient area estimator that abstracts the unique features of fully customized printed MLPs, enabling thus their high-level optimization.
2) We introduce a hardware-aware weight sharing methodology, dedicated to printed bespoke MLP implementations, for limiting the number of the required multipliers.
3) This is the first automated hardware-aware neural minimization framework for printed MLPs [1].
4) Our co-design methodology enables the realization of high-accuracy battery-powered complex MLPs in PE.

1. https://github.com/ArgyKokk/Printed_MLPs

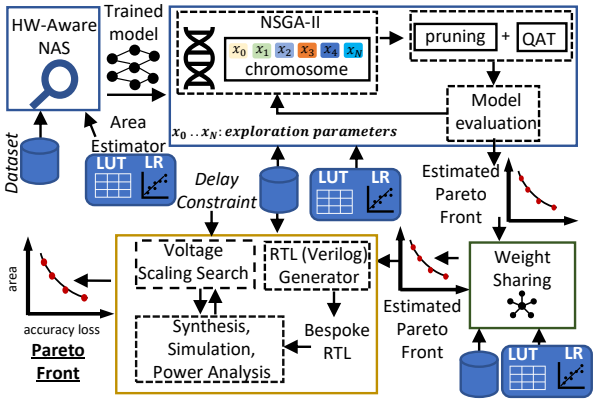

Fig. 1: Our framework for hardware-efficient printed MLPs. Our co-design combines bespoke circuit implementation with hardware-aware neural minimization, leveraging the unique features of these customized circuits.

2 Co-design for Printed MLPs

This section presents the proposed co-design methodology (Fig. 1) for printed MLP circuits that combines the hardware-efficiency of bespoke circuits with a hardware-aware neural minimization that is dedicated to such customized implementations.

For the neural minimization we employ a hardware-aware NAS, quantization, unstructured pruning, and weight sharing. Voltage scaling is additionally applied to boost the power savings. In the remainder of this section we discuss the application of these techniques and emphasize on their expected hardware impact on printed MLPs. Finally, we describe our hardware-aware clustering methodology for the weight sharing, our area estimator and the flow of our proposed methodology.

2.1 Bespoke MLP Circuits

Leveraging the efficiency of bespoke implementations, we design a fully customized MLP per dataset. Integrating the coefficients in the MLP description uses bespoke multipliers, achieving area gains from the multiplications. However, each neuron's product accumulation still requires a conventional adder. Fig. 2 presents the area of EGFET neurons for varying weight precision values. Conventional neurons feature conventional multipliers (i.e., multiplying two inputs of known size), while bespoke neurons use bespoke multipliers (i.e., by constant multipliers that multiply an input of known size by a fixed constant). Given the correlation between the area of bespoke multipliers and weight values [6], a 200-point Monte Carlo analysis is performed for each case. As shown, bespoke neurons consistently have lower area requirements compared to conventional ones.

We consider fully parallel bespoke MLP implementations, where each neuron has dedicated hardware, each weight corresponds to a separate bespoke multiplier, and all neurons operate in parallel. This approach eliminates the need for memories or other sequential elements, which are prohibitively costly in PE [2]. Folded architectures are also avoided due to costly conventional multipliers and registers

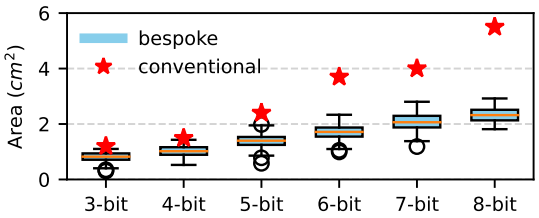

Fig. 2: Area of bespoke and conventional EGFET neurons for 3-bit to 8-bit weights. The number of inputs (and weights) is 6, with input size being 4-bit. For bespoke neurons, a 200-point Monte Carlo analysis is performed for each case.

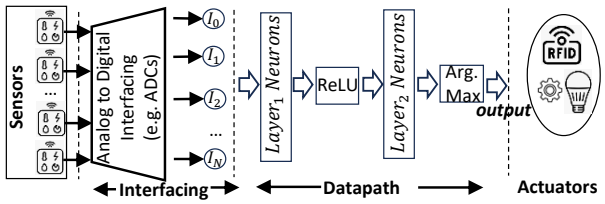

Fig. 3: Overview of a printed MLP classification system. Printed neuron examples are found in Fig. 4.

required. Eventually, the bespoke multipliers, with hard-coded MLP weight values, serve not only as processing elements but also hold the network's parameters. Fig. 3 depicts an abstract overview of a printed MLP-based classification system while hardware neuron examples are illustrated in Fig. 4. Printed ML applications are primarily sensor-based. Sensors capture inputs and provide the classification input features to the MLP through interfacing. When new inputs features arrive, they flow through the fully parallel MLP circuitry directly generating new outputs. Due to the low sampling rate in printed applications (a few Hz) [16], if the MLP's performance is fast enough, buffering of input features is unnecessary. Finally, the classification output is fed to actuators, commonly RFID/NFC. The design and analysis of printed analog-to-digital interfacing and associated hardware overheads are outside the scope of this work.

As shown in Fig. 3, bespoke neurons become less effective at lower bit-widths as the area of the accumulators starts to become the main bottleneck. Therefore, optimizing the accumulators is crucial for achieving high area and power efficiency. In our work, we reduce the size of the accumulators by constraining the size (Section 2.3) and number (Section 2.4) of its summands.

2.2 Hardware-Aware Neural Architecture Search

Given an input dataset, our hardware-aware NAS extracts the most area-efficient model with the highest accuracy. We implement a simplified macro NAS to identify the optimal MLP architecture and training hyperparameters. In NAS, we limit all MLPs to one hidden layer with up to five neurons to keep area requirements low and Relu is used for the activation function due to its hardware efficiency.

For each hidden neuron size, we run a Bayesian optimization to maximize inference accuracy within 100 tries. We define the learning rate range between 0.001 and 0.01, testing various rates and training 100 models with different hyperparameters. The model with the highest accuracy is selected. If multiple models have similar accuracy (within 0.1%), the one with the fewest hidden neurons is chosen. If there's still a tie, our hardware area estimator (see Section 2.7) selects the most area-efficient model. Due to the small size of targeted MLPs, more complex NAS frameworks, e.g., [18], do not need to be examined.

2.3 Low-Bit Quantization for Bespoke MLPs

Quantizing weights and activations to low-precision fixed-point (integer) values leads to simpler arithmetic and more hardware-efficient circuits. In traditional silicon systems, the lower precision used for arithmetic operations the lower the hardware requirements are [19]. In bespoke MLP circuits the impact of quantization is not that straightforward as in silicon systems. In the latter, quantizing a DNN primarily results in memory savings and potential performance gains if the platform supports low-precision computations. However, quantizing below the hardware's capabilities does not yield significant benefits and can degrade accuracy.

For printed bespoke MLPs, hardware gains from quantization are less about multiplier savings—e.g., multiplying by 27 and 111 both occupy 0.63cm² despite needing 5 and 7 bits respectively. Similarly, multiplying an input A by 3 will always require the same area even if 3 or 8 bits are used to represent the weights. As shown in Fig. 2, reducing weight size in conventional neurons from 8 bits to 3 bits results in a 4.5x area reduction, while bespoke neurons see a 2.7x reduction. The area of bespoke multipliers depends more on the weight value than the precision used for quantization, leading to overlaps in area among bespoke neurons with varying bit-widths in Fig. 2. Note that Fig. 2 assumes a random distribution for the weights. In MLPs, significant portions of weights might be close to zero, making bespoke multipliers less affected by quantization precision, thereby amplifying the findings in Fig. 2. Nevertheless, significant gains come from constraining the largest possible product, which results in smaller adders for product accumulation, and increasing weight sharing probability since weights are concentrated in smaller segments (see Section 2.5).

We also explore low-bitwidth activation quantization. Low activation precision yields high area savings by reducing both multiplier and accumulator sizes. MLP inputs are truncated to the required precision, and a quantized ReLU (QRelu) is used in the hidden layer. For weight and bias quantization, stochastic rounding is employed for higher accuracy and because values are determined offline during training. Activation truncation is hardware-friendly, and linear quantization is used for implementation simplicity. Models are quantized using Qkeras [20], and quantization-aware training (QAT) is performed to improve accuracy. Additionally, L1-regularization in QAT forces weights to small values, resulting in more efficient accumulators.

2.4 Unstructured Pruning

Another popular approach to minimize neural networks is pruning. In typical silicon-based MLPs, pruning reduces the network's memory requirements. Structured pruning is preferred for easier hardware exploitation, while unstructured pruning achieves higher sparsity and/or accuracy. However, in bespoke printed MLP circuits, unstructured

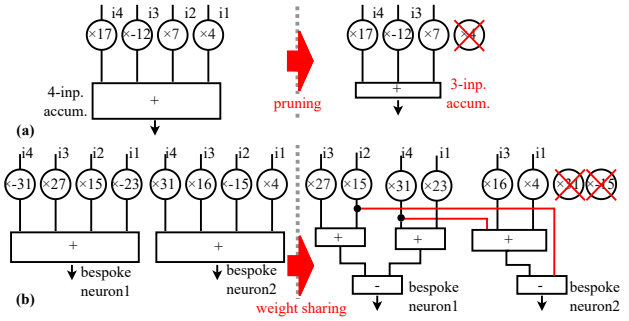

Fig. 4: Examples of the hardware impact of (a) pruning and (b) weight sharing on our bespoke MLPs.

pruning directly translates to hardware savings. Removing a connection eliminates the respective bespoke multiplier, leading to area and power reductions.

Since weights with smaller absolute values are mainly pruned (e.g., low magnitude unstructured pruning [21]), the area gains from eliminating bespoke multipliers may be modest, especially if the multipliers are for weights like 1, 2, 4, etc., which require only wiring. Nonetheless, pruning in bespoke MLPs reduces also the area of each neuron's accumulator, as fewer products need to be summed. An illustrative example is shown in Fig. 4a. This differs from typical silicon systems, where accumulators must accommodate the largest possible sum in any neuron. For example, in the RedWine MLP (see Section 3), 20% sparsity reduces the MLP area by 19%, with only 7% due to multiplier area reduction. At 50% sparsity, the area decreases by 60%, but only 28% is from multiplier savings.

2.5 Weight Sharing in Bespoke MLPs

Weight sharing is a compression technique initially designed to reduce a network's memory footprint [22]. However, we employ weight sharing to achieve hardware gains. When weights of different neurons in the same layer, which are multiplied by the same input, share the same value, only one bespoke multiplier is needed, and its product can be reused across multiple neurons (see Fig. 4b). This reduces the number of required multipliers while maintaining the same number of summands in each neuron. When optimizing for area, EDA synthesis tools can auto-detect subexpression sharing, allowing for seamless area and power-efficient implementations if a model exhibits high weight-sharing potential.

In our neural minimization process, after pruning and quantizing the MLP, we further approximate the model by enforcing weight sharing. We introduce a hardware-aware weight clustering methodology tailored for printed bespoke MLP implementations. This involves clustering same-input weights and fine-tuning the centroids' values to generate fewer and more area-efficient bespoke multipliers. Unlike typical state-of-the-art weight clustering approaches [22], which group all weights in each layer without considering their relative positions, our approach focuses on clustering same-input weights to maximize multiplier sharing. Additionally, we select centroid values in an area-aware manner,

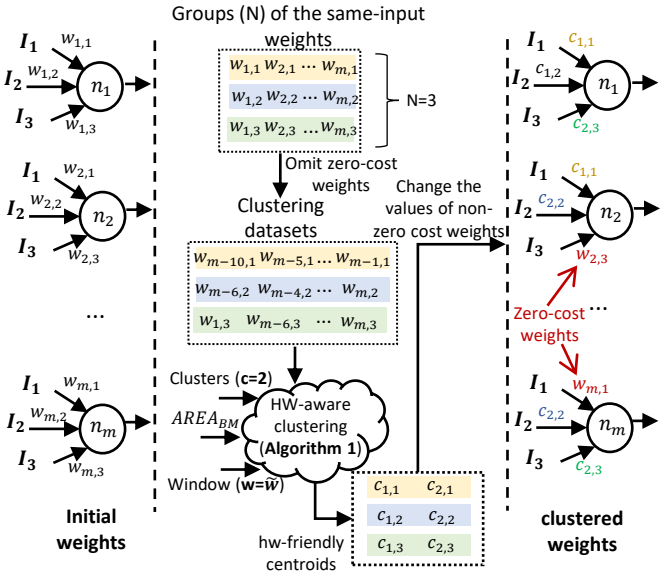

Fig. 5: Example of our proposed hardware-aware weight clustering algorithm applied for m neurons with three inputs. The number of clusters (c) is two and the size of the exploration window (w) is set to a value $\tilde{w}$.

leading to more area-efficient results. Traditional memory-oriented clustering methods are unsuitable for printed applications due to their hardware-unawareness.

Additionally, to increase the probability of multiplier sharing, we split the weights of each neuron into two groups that comprise the positives and negatives ones. An example of our weight sharing implementation is shown in Fig. 4b. The absolute value of the weights is used for the bespoke multiplier. Considering that the inputs are always positive (e.g., due to Relu), the sign of the weight determines the product's sign. By accumulating separately the products of each group and then subtracting the sums, we obtain the expected result. The benefit of "artificially" removing the negative weights is twofold. Firstly, the area of a positive bespoke multiplier is mainly smaller than its negative counterpart [6], and splitting might result in shorter carry-chains and smaller adders. A 200-point Monte Carlo analysis on neurons with varying configurations (4 to 12 inputs of 4 or 8 bits and random weights) shows that splitting reduces area by an average of 16.7%. Secondly, the probability for multiplier sharing is theoretically doubled since the possible values for the bespoke multipliers are halved, increasing the possibility of re-appearing values.

An example of our proposed hardware-aware weight clustering is illustrated in Fig. 5. In our methodology two user-defined parameters are set before the algorithm's execution: i) the number of clusters c and ii) the algorithm's exploration window w. Initially, for each of the network's layers the same-input weights are split into N groups, where N is the number of different inputs. For each group, we identify the weights that do not lead to a zero-area bespoke multiplier (see AREA(BM_w) in Section 2.7) and we execute the procedure described in Algorithm 1.

Specifically, in Algorithm 1, first we cluster the same-input weights using the K-means algorithm for c number of clusters. Then, for each cluster we replace the values

Algorithm 1 Hardware-Aware Clustering

```
// AREA(BM_W): See Section 2.7; c: number of clusters; w: window size
procedure CLUSTERING
    for d in clust_datasets do
        centroids, clustered_points = Kmeans(d, c)
        for K in centroids do
            low_cost_cent = K
            cost = AREA(BM_K)
            for window in range( −w/2, w/2 ) do
                K̃ = K + window
                new_cost = AREA(BM_K̃)
                if new_cost < cost then
                    low_cost_cent = K̃
                    cost = new_cost
            new_centroids.append(low_cost_cent)
    return new_centroids
```

of the weights with a value in the close proximity of the centroid value of the respective cluster. More precisely, for each cluster, assuming K the centroid value, we replace the weights of that cluster with $\tilde{K}, \forall \tilde{K} \in [K - w/2, K + w/2]$. Finally, among all the obtained configurations, we select the one that minimizes $\mathrm{AREA}(BM_{\tilde{K}})$.

2.6 Voltage Scaling

Approximating a circuit usually makes it faster [19]. This delay gain can be leveraged to lower the voltage value and further improve the power-efficiency at the same performance as the exact baseline [19]. Though, voltage scaling does not deliver any area gains. In our case, using low-precision operands (input activations and weights) makes the bespoke multipliers smaller since less partial products need to be accumulated. Moreover, the products are also smaller leading to smaller carry-chains and faster accumulators. Furthermore, pruning weights removes products, resulting in accumulators of smaller depth and potentially faster. Leveraging the performance boost due to our neural minimization, we perform a binary search to find the minimum voltage value that satisfies a user's specified delay constraint. Hence, additional power savings are achieved without any timing errors due to the voltage drop. Considering that EGT printed circuits (as in our case) may be operated at only 0.6V [5], a high performance gain from minimizing the network may lead to high additional power gains. Note that printed batteries are customizable in voltage, shape, etc., [23]. In our work, we avoid timing errors by not considering voltage scaling beyond the minimum supported voltage value that ensures no timing violations.

2.7 Proposed Bespoke Area Estimator

Our introduced area estimator lies at the core of our neural minimization methodology, abstracting the unique features of bespoke printed MLP circuits and enabling high level exploration of the neural minimization parameters (see Section 2.8). Using our estimator, we can run our hardware-aware neural minimization without requiring circuit synthesis to evaluate each design solution. Time-efficiency is critical considering the in-situ, on demand, and low-volume fabrication of printed circuits. The main elements that contribute to a printed bespoke MLP's area are i) the bespoke multipliers, ii) the conventional accumulators, and iii) QRelu of each neuron. A circuit's area can be estimated by the sum of the area of its sub-components. Denoting a neuron as N, a weight as w, a bespoke multiplier by w as BM_w, we express the MLP area as follows:

$$\mathrm{AREA}(MLP) = \sum_{\forall N \in MLP} \mathrm{AREA}(N), \tag{1}$$
$$\mathrm{AREA}(N) = \sum_{\substack{\forall w \in N| \\ w \text{ not shared}}} \mathrm{AREA}(BM_w) + \mathrm{AREA}(Add) + \mathrm{AREA}(QRelu)$$

$\mathrm{AREA}(BM_w)$: As aforementioned, the area of a bespoke multiplier depends on the coefficient value and the input precision. Hence, $\forall w \in [0, 128]$ (we use the absolute values in the circuit description and weights are up to 8 bits) we run an offline process in which we synthesize all the BM_w for all the supported input precisions (see Section 2.8) and store the area values in a look-up table (LUT). Hence, if z is the input precision, $\mathrm{LUT}(z, w)$ gives the area of the respective bespoke multiplier. Each evaluation requires less than a second and all the evaluations are fully parallelizable. To identify if a weight w is shared or not, we just need to check if it has the same absolute value with any of the same-input weights of the previous neurons of the same layer.

$\mathrm{AREA}(QRelu)$: QRelu requires only an inverter (INV) and a few AND and OR gates. The number of the required gates depends only on the QRelu's input and output precision. Assume that the input is in the $\mathrm{Q}I_i.F_i$ format and the output in the $\mathrm{UQ}I_r.F_r$ one. Considering that the printed technology library [3] comprises only 2-input OR gates:

$$\begin{aligned}\mathrm{AREA}(QRelu) =& \mathrm{AREA}(\mathrm{INV}) + (I_r + F_r) \cdot \mathrm{AREA}(\mathrm{AND}) \\ &+ max(I_i - I_r - 1, 0) \cdot \mathrm{AREA}(\mathrm{OR}),\end{aligned} \tag{2}$$

where the area of the AND, OR, and INV is constant and obtained directly from the technology library.

$AREA(Add)$: The area of the accumulator depends on the number and precision of the summands (products). Overall, the area of an accumulator is proportional to the number of the required full-adders. However, since in bespoke architectures the coefficients are hardwired, the EDA tools will do constant propagation to further optimize the logic. In addition, the summands (products) are proportional to the weight values. Hence, we express the area of the accumulator as a linear regression (LR) of the neuron's weights. For all possible input sizes, we randomly generate 100 accumulator samples and synthesize them to obtain their area and train our LR model. We train a LR per input precision. Each synthesis requires a few seconds and they can run fully in parallel. Assuming W the list of neuron's weights and z the input precision, the area of the accumulator is estimated by $AREA(Add) = LR_z(W)$.

2.8 Proposed Co-design Framework for Bespoke MLPs

Our neural minimization methodology is implemented in a framework (illustrated in Fig. 1) that operates as follows:

2.8.1 First step

In the first step of our framework, we run our hardware-aware NAS (Section 2.2) on the input dataset to optimize the size of the single hidden layer. This process yields an MLP with the highest accuracy and minimum size. Subsequently, we apply the remaining optimizations serially.

2.8.2 Second step

The obtained model is pruned (Section 2.4) with the target sparsity level and then the weights, activations, and biases are quantized (Section 2.3) using the desired precision values. QAT is performed using the Adam optimizer and the learning rate is set to $1 \cdot e^{-4}$ to fine-tune the quantized weights. To identify the optimal value for the sparsity level and the coefficients/inputs precision, we employ the elitist NSGA-II genetic algorithm from the open-source framework Pymoo to explore the design space and find the solutions that minimize the design objectives of printed bespoke MLPs:

$$\begin{aligned} &\min_{w,b,r,i,s} \text{AccuracyLoss}(MLP_N, c, b, r, i, s) \\ &\min_{w,b,r,i,s} \text{AreaEstimation}(MLP_N, c, b, r, i, s), \end{aligned} \tag{3}$$

where MLP_N is the model obtained from NAS, s is the target sparsity level, and c, b, r, and i are respectively the precision levels for the weights, biases, activations of the output layer (i.e., QRelu precision), and input precision. As in QKeras, the precision levels are in the form (P, I) where P specifies the number of bits for the quantization, and I specifies how many bits of P are used for the integer part. The number of bits (P) for c and b is ≤ 8, for i is ≤ 4, and for $r \leq 8$. Finally, the examined sparsity levels are $\leq 80\%$ with a 10% step. w, b, r, i, and s are encoded in the chromosomes' genes. Each chromosome generates an MLP with a unique set of parameters. The accuracy of each MLP is evaluated and its hardware efficiency is assessed using our area estimator (Section 2.7). The output of our genetic optimization is the Pareto front of the solutions that minimize the optimization objectives (3).

2.8.3 Third step

We apply our weight sharing (Section 2.5) methodology on the MLPs extracted in step 2. Our clustering algorithm is executed at each of the network's layers for all possible number of clusters (c). Though, note that c is well bound by the number of neurons on the corresponding layer (e.g for a layer that consists of 10 neurons up to $c = 10$ clusters can be generated). For the exploration windows sizes (w) of $0, 2, 4, 8, 10$, and 20 are considered. The K-means algorithm is executed for 10 iterations. Again, our area estimator is used to obtain the new accuracy-estimated area Pareto front after weight sharing.

2.8.4 Fourth step

The final step of our framework is to generate the bespoke hardware descriptions (Section 2.1), in Verilog, of the MLPs obtained from the previous step. The obtained circuits are synthesized at the minimum voltage value that satisfies a user-defined delay constraint (Section 2.6).
Then, circuit simulation and power analysis with switching activity annotation are performed to obtain the circuit's power and verify that its accuracy meets the software (QKeras) extracted one. Finally, among the analyzed designs the power-accuracy and/or area-accuracy Pareto front is the output of our framework. Note that at this stage, the area is not estimated but it is calculated from the EDA tools.

TABLE 1: Evaluated classification MLPs after NAS

Classifier	A^{+}	W*	T†	NAS‡
WhiteWine	0.54	72	(11,4,7)	19min.
RedWine	0.56	34	(11,2,6)	22min.
Cardio	0.88	72	(21,3,3)	20min.
Vertebral (3C)	0.83	27	(6,3,3)	16min.
Pendigits	0.94	130	(16,5,10)	35min.
Seeds	0.94	30	(7,3,3)	12min.

+ The MLPs classification inference accuracy using 8-bit coefficients and 4-bit inputs.
⋆ Number of weights per MLP.
† The MLPs topology (input, hidden layer, output).
‡ Single-threaded NAS exploration time on Xeon 5218R.

3 Results and Analysis

In this section, we evaluate the efficiency of our framework in generating area- and power- efficient printed MLPs and assess its effectiveness in enabling their battery powered operation. To that end, we consider six classification MLPs trained on the RedWine, WhiteWine, Cardiotocography, Vertebral (3 column), Pendigits, and Seed datasets of the UCI ML repository [24], similarly to [6], [7]. Synopsys Design Compiler and PrimeTime are used for synthesis and power analysis, respectively, while QuestSim is used for circuit simulation. The open source EGFET library [3] is used. The delay constraint for Pendigits is 250ms and for the rest MLPs 200ms. Since we consider fully parallel architectures, this is also the latency of a single inference, aligning with typical few-Hz-performance of PE [3] and comparable to state-of-the-art latencies [6], [7]. In our genetic algorithm we set the population size to 80 and the number of generations to 60.

First, we evaluate the area-efficiency of the MLPs produced by our co-design methodology. As baseline printed MLPs we consider the un-simplified bespoke implementation [2] of the models obtained from NAS. Table 1 shows the inference accuracy, topology, and the NAS exploration time for the six examined classification MLPs. All MLPs achieve the same inference accuracy as in [2]. For the NAS, Scikit-learn and the randomized parameter optimization with 5-fold cross validation are used. The input dataset is randomly split to a $70\%/30\%$ train/test set and inputs are normalized to $[0, 1]$. Limited-memory BFGS, Stochastic Gradient Descent, and Adam are considered for the solver. As shown in Table 1, NAS converges quickly due to the simplicity of the examined models, averaging only 20 min.

Fig. 6 presents the true area-accuracy Pareto front of our printed MLPs. The values are normalized w.r.t. the area and accuracy of the respective baseline [2]. The voltage value is set to 1V for all the designs in Fig. 6. Moreover, Fig. 6 depicts the same analysis for each technique when applied in isolation, i.e., bespoke MLP circuits with only quantization ($\times$), pruning ($\triangledown$), or weight sharing ($\square$) using our weight clustering methodology. In addition, we also include designs that apply only the state-of-the-art weight clustering (but hardware-unaware) algorithm [22] ($\triangle$).

Compared to the baseline [2], our MLPs feature very high area reduction for minimal accuracy loss. Indicatively, on average, our MLPs exhibit 6x less area than the baseline [2] for no accuracy loss. Furthermore, our MLPs constitute the most efficient solutions as they always appear on the Pareto-front in Fig. 6. Specifically, for similar accuracy, our

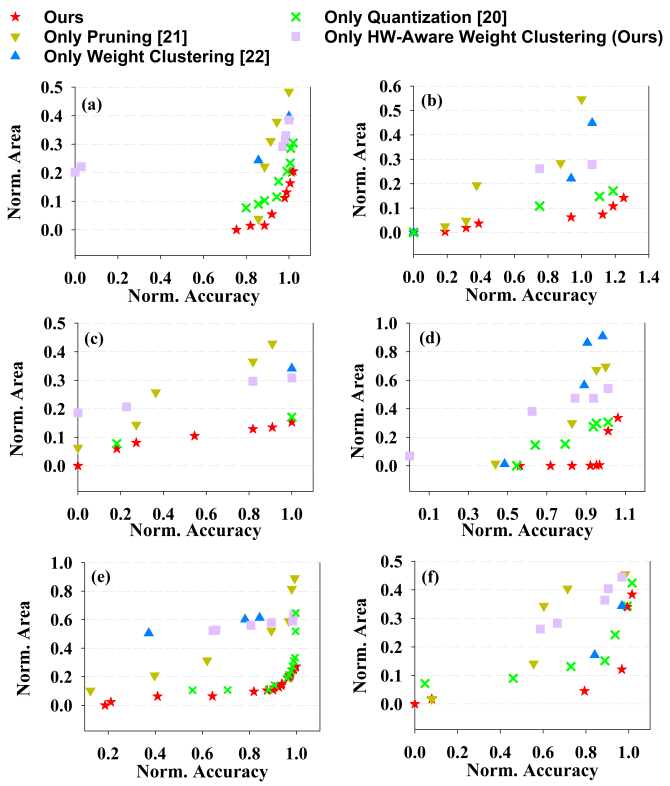

Fig. 6: Area-Accuracy trade-off of the printed MLPs produced by our framework, only pruning [21], only quantization [20], only weight sharing with our clustering, and only weight sharing with [22]. Values are normalized over each baseline MLP [2]. Classifiers: (a) WhiteWine, (b) RedWine, (c) Cardio (d) Vertebral (3C), (e) Pendigits, (f) Seeds.

MLPs achieve on average 2.7x, 3.4x, 2.9x, and 42% lower area compared to only our hardware-aware weight sharing, weight sharing [22], pruning, and quantization respectively.

It is noteworthy that for the Vertebral MLP (Fig. 6d), our framework can produce designs that occupy up to 185x less area than the baseline [2] for a less than 2% accuracy drop. This is because the examined network is highly tolerant to low-bit coefficients and our co-design framework produced MLPs with 2-bit weights that are either zero or power of two. Interestingly, as also observed in many cases in approximate neural networks [25], our framework can produce MLPs with an inference accuracy higher than the baseline exact design. For instance, in the Redwine MLP (Fig. 6b) there is a Pareto-optimal design with 6% higher inference accuracy than [2] and 7x less area.

Moreover, as observed in Fig. 6, our hardware-aware weight sharing outperforms [22] for all the examined MLPs but the Seeds. For similar accuracy designs, our weight sharing delivers MLPs with 53.4% less area, on average, than [22], while for Seeds, [22] generates MLPs that are 29% smaller on average. In the case of Seeds, the [22] clustering produced many clusters with zero centroids. Eventually, the latter becomes equivalent to neuron pruning. However, in our work such cases will be captured by the pruning step of our framework (see Section 2.8).

Next, in Fig. 7, we compare our MLPs against the current

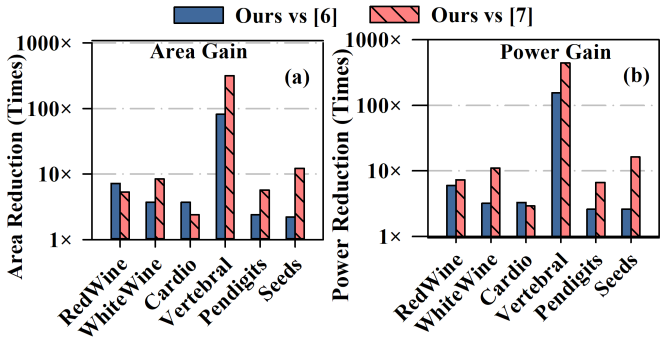

Fig. 7: a) Area and b) power reduction achieved by our printed MLPs vs the state-of-the-art approximate [6] and stochastic [7] ones. Iso-accuracy evaluation is considered.

state-of-the-art [6], [7] that also consider complex printed MLP classifiers. In [6] and [7] approximate and stochastic computing are employed, respectively. In Fig. 7 we consider an iso-accuracy evaluation. When comparing our MLPs with [6] we set the accuracy threshold to 1% for both frameworks. When comparing against [7] we set the accuracy threshold for our MLPs equal to the accuracy achieved by the respective stochastic MLP of [7]. Since [6], [7] consider high voltage operation, we again set (as in Fig. 6) the voltage value of our circuits to 1V. As shown in Fig. 7, our MLPs outperform significantly both the approximate [6] and the stochastic [7] MLPs. Compared to [6], besides the Vertebral, our MLPs feature on average 3.8x and 3.5x lower area and power, respectively, while for the Vertebral we achieve 82x less area and 154x lower power. Similarly, compared to [7], our MLPs achieve 6.8x and 8.8x lower area and power consumption, while for the Vertebral the area and power gains are 316x and 445x higher respectively.

Furthermore, we assess the effectiveness of our framework in generating battery powered printed MLPs. Unlike Fig. 6 and 7, in this evaluation we employ the full-potential of our framework by also applying voltage scaling. We consider three accuracy loss constraints, i.e., up to 0.2%, 1%, and 3% accuracy degradation compared to the baseline MLPs. We present in Table 2 the accuracy and hardware characteristics of our MLPs that feature the least power consumption and satisfy the accuracy constraint. The circuits are synthesized at a reduced voltage value and thus, at the same accuracy, the designs in Table 2 might be different than the ones in Fig. 6 or there might be some slight area differences for the same designs. As shown in Table 2, for up to 0.2% accuracy loss, our MLPs achieve 16.7x lower power consumption on average compared to the baseline MLPs [2]. For an accuracy loss up to 1%, the average power gain increases to 21.4x and for a loss up to 3% the gain in power consumption is 33.7x on average.

Similarly, the area gain is on average 5.8x, 7.5x and 11x for the 0.2%,1% and 3% accuracy loss thresholds, respectively. Moreover, as shown in Table 2, for all the MLPs (except from Pendigits) a voltage value of 0.6V is selected by our voltage scaling. In other words, the delay gain due to our neural minimization enables almost all MLPs to operate at the lowest voltage value without any performance loss.

In Table 2, we use a color code to represent the batteries that could power our MLPs. Orange represents a Molex

TABLE 2: Hardware analysis of the printed MLPs produced by our framework for up to 0.2%, 1% and, 3% accuracy loss.

Classifier	Acc. Loss*	Area (cm^2)	Area Gain*	Power (mW)	Power Gain*	Voltage (V)
WhiteWine+	≤ 0.2%	6.9	4.5x	6.6	14.9x	0.60
WhiteWine	≤ 1%	3.8	8.2x	3.8	25.9x	0.60
WhiteWine	≤ 3%	1.4	22.2x	1.6	61.5x	0.60
RedWine	≤ 0.2%	1.2	14.6x	1.4	38x	0.60
RedWine	≤ 1%	1	17.6x	1.2	44.4x	0.60
Cardio	≤ 0.2%	6.1	5.4x	5.9	16.4x	0.60
Cardio	≤ 1%	5.2	6.4x	5.1	19x	0.60
Cardio	≤ 3%	3.6	9.2x	3.1	31.3x	0.60
Vertebral	≤ 0.2%	2.6	5x	2.8	14.5x	0.60
Vertebral	≤ 3%	0.03	436x	0.04	1020x	0.60
Pendigits	≤ 0.2%	22.9	2.92x	42.5	5x	0.80
Pendigits	≤ 1%	21.9	3x	35.4	6x	0.76
Pendigits	≤ 3%	16.9	3.9x	27.6	7.7x	0.72
Seeds	≤ 0.2%	3.8	2.6x	3.9	11.5x	0.60
Seeds	≤ 1%	3.4	2.6x	3.7	12.1x	0.60
Seeds	≤ 3%	1.1	9x	1.3	34.6x	0.60

* With respect to the respective baseline MLP [2].
\+ Colors denote a printed battery that can power each MLP. Orange: 30mW, light blue: 15mW, dark blue: 3mW, green: harvester (< 2mW).

30mW batery, light blue a Zinergy 15mW, dark blue a BlueSpark 3mW, green a printed energy harvester, while no color means that there is no adequate power supply. As shown, for up to 3% accuracy loss all our MLPs can be battery-powered while for up to 0.2% and up to 1% loss only Pendigits cannot. At the 3% threshold only the Pendigits and the Cardio MLPs require a printed battery (i.e a Molex and a Zinergy) for their operation, all the other examined classifiers can be powered up by an energy harvester. Finally, our MLPs feature mainly a reasonable area to be considered for a printed application. The area and power values in Table 2 are subject to combinational logic only since our classifiers don't use any memories or registers. Moreover, the power consumption is mostly static, since EGFET is a nMOS-resistor technology. For the MLPs listed in Table 2, their static power consumption is on average 86% of their overall power consumption.

Finally, we evaluate the time-efficiency of our framework and the accuracy of our area estimator. Our framework's exploration time is on average 220 min ranging from 135 min to 330 min (including NAS time). These values refer to a single-threaded execution on a Xeon 5218R server. To assess the accuracy of our area estimator, we randomly generated $5,000$ MLPs and evaluated their area. The Pearson correlation coefficient of our estimator is 0.89, i.e., very good linear correlation. Hence, our estimator can predict with high confidence if a MLP is more area efficient than another one and, thus, is perfectly capable in guiding our optimization search towards more area-efficient solutions.

4 Conclusions

Printed electronics is a promising solution to bring "intelligence" in application domains characterized by ultra-low-cost fabrication and flexibility requirements. In this paper, we present a co-design methodology for printed MLPs. We propose a hardware-aware weight clustering methodology to achieve sharing of bespoke multipliers among the same-layer neurons and we introduce an area estimator for the time efficient estimation of the MLPs occupied on-chip area. Our hardware-aware neural minimization, combined with fully customized circuit implementations, delivers very high hardware gains (16.7x lower power consumption on average) at only up to 0.2% accuracy loss. Importantly, for only 3% loss, our co-design enables, for the first time, battery-powered operation of complex printed MLPs.

References

[1] B. Shao, "Fully printed chipless rfid tags towards item-level tracking applications," *Doctoral dissertation, KTH Royal Institute of Technology*, 2014.

[2] M. H. Mubarik *et al.*, "Printed machine learning classifiers," in *Annu. Int. Symp. Microarchitecture (MICRO)*, 2020, pp. 73–87.

[3] N. Bleier, M. Mubarik, F. Rasheed, J. Aghassi-Hagmann, M. B. Tahoori, and R. Kumar, "Printed microprocessors," in *Annu. Int. Symp. Computer Architecture (ISCA)*, jun 2020, pp. 213–226.

[4] G. Cadilha Marques *et al.*, "Digital power and performance analysis of inkjet printed ring oscillators based on electrolyte-gated oxide electronics," *Applied Physics Letters*, vol. 111, no. 10, p. 102103, 2017.

[5] C. Marques *et al.*, "Progress Report on "From Printed Electrolyte-Gated Metal-Oxide Devices to Circuits"," *Advanced Materials*, vol. 31, 2019.

[6] G. Armeniakos, G. Zervakis, D. Soudris, M. B. Tahoori, and J. Henkel, "Cross-layer approximation for printed machine learning circuits," in *Design, Automation & Test in Europe Conference & Exhibition*, 2022.

[7] D. D. Weller *et al.*, "Printed stochastic computing neural networks," in *Design, Automation & Test in Europe Conference & Exhibition*, 2021.

[8] E. Ozer *et al.*, "A hardwired machine learning processing engine fabricated with submicron metal-oxide thin-film transistors on a flexible substrate," *Nature Electronics*, vol. 3, no. 7, pp. 419–425, 2020.

[9] D. D. Weller, M. Hefenbrock, M. B. Tahoori, J. Aghassi-Hagmann, and M. Beigl, "Programmable neuromorphic circuit based on printed electrolyte-gated transistors," in *Asia and South Pacific Design Automation Conference (ASP-DAC)*, 2020, pp. 446–451.

[10] N. Bleier *et al.*, "Flexicores: low footprint, high yield, field reprogrammable flexible microprocessors," in *Int. Symp. Computer Architecture (ISCA)*, 2022, pp. 831–846.

[11] E. Ozer *et al.*, "Bespoke machine learning processor development framework on flexible substrates," in *Int. Conf. Flexible and Printable Sensors and Systems (FLEPS)*, 2019, pp. 1–3.

[12] C. Banbury *et al.*, "Micronets: Neural network architectures for deploying tinyml applications on commodity microcontrollers," *SysML Conf.*, 2021.

[13] J. M. Meng, V. Kolala, C. Zhou, P. Hansen, P. Whatmough, and J.-s. Seo, "Fixyfpga: Efficient fpga accelerator for deep neural networks with high element-wise sparsity and without external memory access," *Int. Conf. Field-Programmable Logic and Applications*, 2021.

[14] P. Whatmough, C. Zhou, P. Hansen, K. Venkataramanaiah, J.-s. S. Seo, and M. Mattina, "Fixynn: Efficient hardware for mobile computer vision via transfer learning," *SysML Conf.*, 2019.

[15] P. Whatmough, S. Lee, D. Brooks, and G.-Y. Wei, "Dnn engine: A 28-nm timing-error tolerant sparse deep neural network processor for iot applications," *IEEE J. Solid-State Circuits*, pp. 2722–2731, 2018.

[16] J. Henkel *et al.*, "Approximate computing and the efficient machine learning expedition," in *Int. Conf. Computer-Aided Design (ICCAD)*, 2022.

[17] S. Liu *et al.*, "Stochastic dividers for low latency neural networks," *IEEE Trans. Circuits Syst. I, Reg. Papers*, vol. 68, no. 10, pp. 4102–4115, 2021.

[18] I. Fedorov, R. Matas, H. Tann, C. Zhou, M. Mattina, and P. Whatmough, "Udc: Unified dnas for compressible tinyml models for neural processing units," *Advances in Neural Information Processing Systems 35 (NeurIPS 2022)*, 2022.

[19] G. Zervakis *et al.*, "Thermal-aware design for approximate dnn accelerators," *IEEE Trans. Comp.*, vol. 71, no. 10, pp. 2687–2697, 2022.

[20] C. N. Coelho *et al.*, "Automatic heterogeneous quantization of deep neural networks for low-latency inference on the edge for particle detectors," *arXiv:2006.10159*, 2020.

[21] F. Jonathan and C. Michael, "The lottery ticket hypothesis: Finding sparse, trainable neural networks," *arXiv:1803.03635*, 2018.
[22] S. Han, H. Mao, and W. J. Dally, "Deep Compression: Compressing Deep Neural Network with Pruning, Trained Quantization and Huffman Coding," in *Int. Conf. Learning Representations*, 2016.
[23] S. Lanceros-Méndez and C. M. Costa, *Printed Batteries: Materials, Technologies and Applications*. Wiley, 2018.
[24] D. Dua and C. Graff, "UCI machine learning repository," 2017.
[25] M. S. Ansari, V. Mrazek, B. F. Cockburn, L. Sekanina, Z. Vasicek, and J. Han, "Improving the accuracy and hardware efficiency of neural networks using approximate multipliers," *IEEE Trans. VLSI Syst.*, vol. 28, no. 2, pp. 317–328, 2020.