Multiplierless MP-Kernel Machine For Energy-efficient Edge Devices

Abhishek Ramdas Nair*, *Member, IEEE,* Pallab Kumar Nath*, *Member, IEEE,* Shantanu Chakrabartty, *Senior Member, IEEE,* and Chetan Singh Thakur, *Senior Member, IEEE*

***Abstract*—We present a novel framework for designing multiplierless kernel machines that can be used on resource-constrained platforms like intelligent edge devices. The framework uses a piecewise linear (PWL) approximation based on a margin propagation (MP) technique and uses only addition/subtraction, shift, comparison, and register underflow/overflow operations. We propose a hardware-friendly MP-based inference and online training algorithm that has been optimized for a Field Programmable Gate Array (FPGA) platform. Our FPGA implementation eliminates the need for DSP units and reduces the number of LUTs. By reusing the same hardware for inference and training, we show that the platform can overcome classification errors and local minima artifacts that result from the MP approximation. The implementation of this proposed multiplierless MP-kernel machine on FPGA results in an estimated energy consumption of 13.4 pJ and power consumption of 107 mW with ~9k LUTs and FFs each for a 256×32 sized kernel making it superior in terms of power, performance, and area compared to other comparable implementations.**

***Index Terms*—Support Vector Machines, Margin Propagation, Online Learning, FPGA, Kernel Machines.**

I. Introduction

EDGE computing is transforming the way data is being handled, processed, and delivered in various applications [1] [2]. At the core of edge computing platforms are embedded machine learning (ML) algorithms that make decisions in real-time, which endows these platforms with greater autonomy [3] [4]. Common edge-ML architectures reported in the literature are based on a deep neural network [5] or support vector machines [6], and one of the active areas of research is to be able to improve the energy efficiency of these ML architectures, both for inference and learning [7]. To achieve this, the hardware models are first trained offline, and the trained models are then optimized for energy efficiency using pruning or sparsification techniques [8] before being deployed on the edge platform.

An example of such design-flows is the binary or the quaternary neural networks, which are compressed and energy-efficient variants of deep-neural network inference engines [9] [10]. However, robust training of quantized deep learning architectures still requires full precision training [11]. Also, deep neural networks require a large amount of training data [12] which is generally unavailable for many applications. On the other hand, Support Vector Machines (SVMs) can provide good classification results with significantly less training data. The convex nature of SVM optimization makes its training more robust and more interpretable [13]. SVMs have also been shown to outperform deep learning systems for detecting rare events [14] [15], which is generally the case for many IoT applications. One such IoT-based edge device architecture is depicted in Fig. 1. Data from video surveillance, auditory event, or motion sensor can be analyzed, and the system can be trained on the device to produce robust classification models.

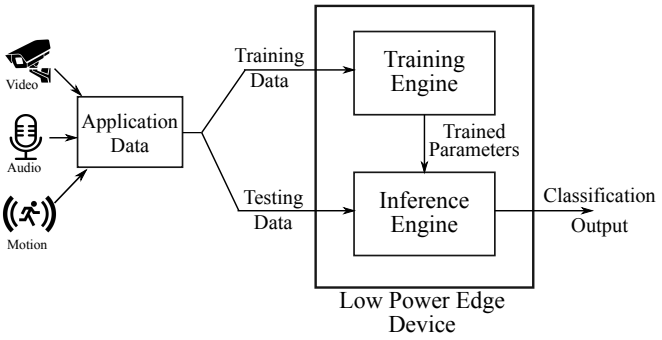

Fig. 1: Edge Device with Online Learning Capability

At a fundamental level, SVMs and other ML architectures extensively use Matrix-Vector-Multiplication (MVM) operations. One way to improve the overall system energy efficiency is to reduce the complexity or minimize MVM operations. In literature, many approximations and reduced precision MVM techniques have been proposed [16] and have produced improvements in energy efficiency without significantly sacrificing classification accuracy. Kernel machines have similar inference engine as SVMs, sharing the execution characteristics with SVM [17]. This paper proposes a kernel machine architecture that eliminates the need for multipliers. Instead, it uses more fundamental but energy efficient and optimal computational primitives like addition/subtraction, shift, and overflow/underflow operations. For example, in a 45 nm CMOS technology, it has been shown that an 8-bit multiplication operation consumes 0.2 pJ of energy, whereas an 8-bit addition/subtraction operation consumes only 0.03 pJ of energy [18]. Shift and comparison operations consume even less energy than additions and subtractions, and underflow/overflow operations do not consume any additional

Abhishek Ramdas Nair, Pallab Kumar Nath and Chetan Singh Thakur are with the Department of Electronic Systems Engineering, Indian Institute of Science, Bangalore, KA, 560012 INDIA e-mail: (abhisheknair@iisc.ac.in, pkniitkgp@gmail.com, csthakur@iisc.ac.in).

Shantanu Chakrabartty is with Department of Electrical and Systems Engineering, Washington University in St. Louis,USA, 63130. e-mail:shantanu@wustl.edu

* Both Abhishek Ramdas Nair and Pallab Kumar Nath have contributed equally to this paper.

energy at all. To achieve this multiplierless mapping, the proposed architecture uses a margin propagation (MP) approximation technique originally proposed for analog computing [19]. MP approximation has been optimized for a digital edge computing platform like field-programmable gate arrays (FPGA) in this work. We show that for inference, the MP approximation is computed as a part of learning, and all the computational steps can be pipelined and parallelized for other MP approximation operations. In addition to reporting an MP approximation-based inference engine, we also report an online training algorithm that uses a gradient-descent technique in conjunction with hyper-parameter annealing. We show that the same hardware can be reused for both training and inference for the proposed MP kernel machine. As a result, the MP approximation errors can be effectively reduced. Since kernel machine inference and SVM inference have similar equations, we compare our system with traditional SVMs. We show that MP-based kernel machines can achieve similar classification accuracy as floating-point SVMs without using multipliers or equivalently any MVMs.

The main contributions of this paper are as follows:

- Design and optimization of MP-approximation using iterative binary addition, shift, comparison, and underflow/overflow operations.
- Implementation of energy-efficient MP-based inference on an FPGA-based edge platform with multiplierless architecture that eliminates the need for DSP units.
- Online training of MP-based kernel machine that reuses the inference hardware.

The rest of this paper is organized as follows. Section II briefly discusses related work, followed by section III, where we explain the concept of multiplierless inner product computation and the related MP-based approximation. Section IV presents the kernel machine formulation based on MP theory. Section V details the online training of the system. Section VI provides the FPGA implementation details and contrasts with other hardware implementations of MP-based kernel machine algorithm. Section VII discusses results obtained with few classification datasets and compares our multiplierless system with other SVM implementations, and Section VIII concludes this paper and discusses potential use cases.

II. Related Work

Energy-efficient SVM implementations have been reported for both digital [20] and analog hardware [21], which also exploit the inherent parallelism and regularity in MVM computation. In [22], an analog circuit architecture of Gaussian-kernel SVM having on-chip training capability has been developed. Even though it has a scalable processor configuration, the circuit size increases in proportion to the number of learning samples. Such designs are not scalable as we increase the dataset size for edge devices due to hardware constraints. Analog domain architectures tend to have lower classification latency [23], but they support simple classification models with small feature dimensions. In [24], an SVM architecture has been reported using an array of processing elements in a systolic chain configuration implemented on an FPGA. By exploiting resource sharing capability among the processing units and efficient memory management, such an architecture permitted a higher throughput and a more scalable design.

Digital and optimized FPGA implementation of SVM has been reported in [25] using a cascaded architecture. A hardware reduction method is used to reduce the overheads due to the implementation of additional stages in the cascade, leading to significant resource and power savings for embedded applications. The use of cascaded SVMs increase the classification speeds, but the improved performance comes at the cost of additional hardware resources.

Ideally, a single-layered SVM should be enough for classification at the edge. In [26], SVM prediction on an FPGA platform has been explored for ultrasonic flaw detection. A similar system has been implemented in [27] using Hamming distance for onboard classification of hyperspectral images. Since the SVM training phase needs a large amount of computation power and memory space, and often retraining is not required, the SVM training was realized offline. Therefore, the authors chose to accelerate only the classifier's decision function. In [28], the authors use stochastic computing as a low hardware cost and low power consumption SVM classifier implementation for real-time EEG based gesture recognition. Even though, this novel design has merits of energy efficient inference, the training framework had to be handled offline on a power hungry system.

Often, training offline is not ideal for edge devices, primarily when the device is operating in dynamic environments, i.e., ever-changing parameters. In such cases, retraining of the ML architecture becomes important. One such system is discussed in [29], where sequential minimal optimization learning algorithm is implemented as an IP on FPGA. This can be leveraged for online learning systems. However, this system standalone cannot provide an end-to-end training and inference system. Another online training capable system was reported in [30] and used an FPGA implementation of a sparse neural network capable of online training and inference. The platform could be reconfigured to trade-off resource utilization with training time while keeping the network architecture the same. Even though the same platform can be used for training and inference, memory management and varying resource utilization based on training time make it less conducive to deploy on the edge device. Reconfiguration of the device would require additional usage of a microcontroller, increasing the system's overall power consumption.

As hardware complexity and power reduction are a major concern in these designs, the authors in [31] implement a multiplierless SVM kernel for classification in hardware instead of using a conventional vector product kernel. The data-flow amongst processing elements is handled using a parallel pipelined systolic array system. In the multiplierless block the authors use Canonic Signed Digit (CSD) to reduce the maximum number of adders. CSD is a number system by which a floating-point number can be represented in two's complement form. The representation uses only -1, 0, +1 (or -, 0, +) symbols with each position denoting the addition and subtraction of power of 2. Despite being multiplierless, the system consumes many Digital Signal Processors (DSPs) due

to the usage of polynomial kernel having exponent operation. The usage of DSPs increases the power consumption of the design.

Another system that uses multiplierless online training was reported in [32]. The authors use a logarithm function based on a look-up table, and a float-to-fixed point transform to simplify the calculations in a Naive Bayes classification algorithm. A novel format of a logarithm look-up table and shift operations are used to replace multiplication and division operations. Such systems incur an overhead of generating the logarithmic look-up table for most operations. The authors chose to calculate logarithmic values offline and store them in memory, contributing to increased memory access for every operation.

There have been various neural network implementations that eliminate the usage of multipliers in their algorithms. In [33], Convolutional Neural Networks (CNNs) are optimized using a new similarity measure by taking L_1 norm distance between the filters, and the input features as the output response. Even though it eliminated the multipliers, this implementation requires batch normalization after each convolution operation, resulting in usage of multipliers for this operation. It also requires a full precision gradient-descent training scheme to achieve reasonable accuracy.

Similarly, in [34], convolutional shifts and fully connected shifts are introduced to replace multiplications with bitwise shifts and sign flipping. The authors use powers of 2 to represent weights, bias, and activations and use compression logic for storing these values. There is a significant overhead for compression and decompression logic for the implementation of an online system. In [35], the authors leverage the idea of using additions and logical bit-shifts instead of multiplications to explicitly parameterize deep networks that involve only bit-shift and additive weight layers. This network has limitations for implementing activation functions in the shift-add technique. In all these neural network systems, the training algorithm and activation implementations involve multipliers. Hence these systems cannot be termed as a complete multiplierless system.

In this work, we propose to use an MP approximation to implement multiplierless kernel machine. MP-based approximation was first reported in [19], and in [36], an MP-based SVM was reported for analog hardware. The main objective of this work is to build scalable digital hardware using an optimized MP approximation. Also, the previous work in MP-based SVM has used offline training. Our system is a one-of-a-kind digital hardware system using a multiplierless approximation technique in conjunction with online training and inference on the same platform.

III. Multiplierless Inner-product Computation

We first show that an exact multiplier or an inner product can be computed using the difference between two quadratic functions. Then, we relax some of the properties of the quadratic function and show how to approximate inner products without multipliers using MP-approximation. First, consider the following mathematical expression

$$y = \frac{1}{2}\Big[f(\mathbf{w}+\mathbf{x}, -\mathbf{w}-\mathbf{x}) - f(\mathbf{w}-\mathbf{x}, -\mathbf{w}+\mathbf{x})\Big]. \tag{1}$$

where $f : \mathbb{R}^D \times \mathbb{R}^D \to \mathbb{R}$ is a Lipschitz continuous function, $y \in \mathbb{R}$ is a scalar variable, $\mathbf{w} \in \mathbb{R}^D$ and $\mathbf{x} \in \mathbb{R}^D$ are D dimensional real vectors. Note that if we choose f to be a quadratic equation as $f(\mathbf{x}, -\mathbf{x}) = \frac{1}{2}\mathbf{x}^T\mathbf{x} + c$, where $c \in \mathbb{R}$ is a constant, then we get

$$y = \mathbf{w}^T\mathbf{x}. \tag{2}$$

which is the exact inner product between the vectors $\mathbf{w}$ and $\mathbf{x}$. As a special case when $w \in \mathbb{R}$ and $x \in \mathbb{R}$ are scalars, then using a one dimensional quadratic function

$$y_{QP}(x) = \frac{1}{4}[(w+x)^2 - (w-x)^2] = w \times x \tag{3}$$

as shown in Fig.2a leads to an exact multiplier. However, implementing multiplication and inner products using this approach on digital hardware would require computing a quadratic function, which would require using a look-up table or other numerical techniques [37]. Also, this approach does not consider the finite dynamic range if the operands are represented using fixed precision. While the effect of finite precision might not be evident for 16-bit operands when the precision is reduced down to 8-bits or lower, $y_{QP}(x)$ will saturate due to overflow or underflow. Next, we consider a form of $y_{QP}(x)$ that captures the saturation effect.

Let f be a log-sum-exponential (LSE) function defined over the elements of $\mathbf{x} = [x_1, x_2, ..., x_D]$ as

$$f(\mathbf{x}, -\mathbf{x}) = \log\left(\sum_{i=1}^{D} e^{x_i} + \sum_{i=1}^{D} e^{-x_i}\right), \tag{4}$$

then using a first-order Taylor-Series approximation [38], we get

$$f(\mathbf{w}+\mathbf{x}, -\mathbf{w}-\mathbf{x}) \approx f(\mathbf{x}, -\mathbf{x}) + \mathbf{w}^T\nabla f(\mathbf{x}, -\mathbf{x}) \tag{5}$$

$$f(-\mathbf{w}+\mathbf{x}, \mathbf{w}-\mathbf{x}) \approx f(\mathbf{x}, -\mathbf{x}) - \mathbf{w}^T\nabla f(\mathbf{x}, -\mathbf{x}). \tag{6}$$

where $\nabla f(\mathbf{x}, -\mathbf{x})$ refers to first-order gradient with respect to $\mathbf{w}$. Substituting in eq.(1) leads to

$$y = \mathbf{w}^T\nabla f(\mathbf{x}, -\mathbf{x}) \approx \mathbf{w}^T\mathbf{x}$$

or

$$\approx \mathbf{w}^T\mathbf{x} \approx \sum_{k=1}^{D} w_k \frac{\sum_{i=1}^{D}[e^{x_i} - e^{-x_i}]}{\sum_{i=1}^{D}[e^{x_i} + e^{-x_i}]} \tag{7}$$

The effect of the approximation in eq. (7) can be visualized in Fig.2b for scalars $w \in \mathbb{R}$ and $x \in \mathbb{R}$ and for a one-dimensional LSE function

$$y_{LSE}(x) = \log\left(e^{w+x} + e^{-w-x}\right) - \log\left(e^{w-x} + e^{-w+x}\right). \tag{8}$$

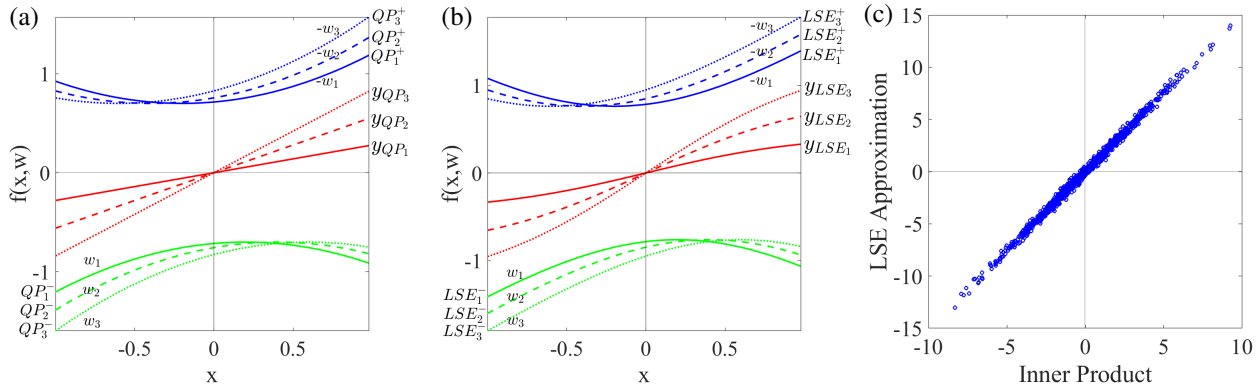

Fig. 2: (a) Scalar Inner Product expressed in quadratic equation for 3 different w, i.e.,(w_1, w_2, w_3). Here, $QP^+ = (w+x)^2$ and $QP^- = -(w-x)^2$. (b) Scalar Inner Product approximation expressed using log-sum-exponential terms for 3 different w, i.e.,(w_1, w_2, w_3). Here, $LSE^+ = \log\left(e^{w+x} + e^{-w-x}\right)$ and $LSE^- = -\log\left(e^{w-x} + e^{-w+x}\right)$. (c) Inner Product and Log-Sum-Exponential Approximation scatter plot of 64 dimensional vectors with each input randomly varied between -1 and +1

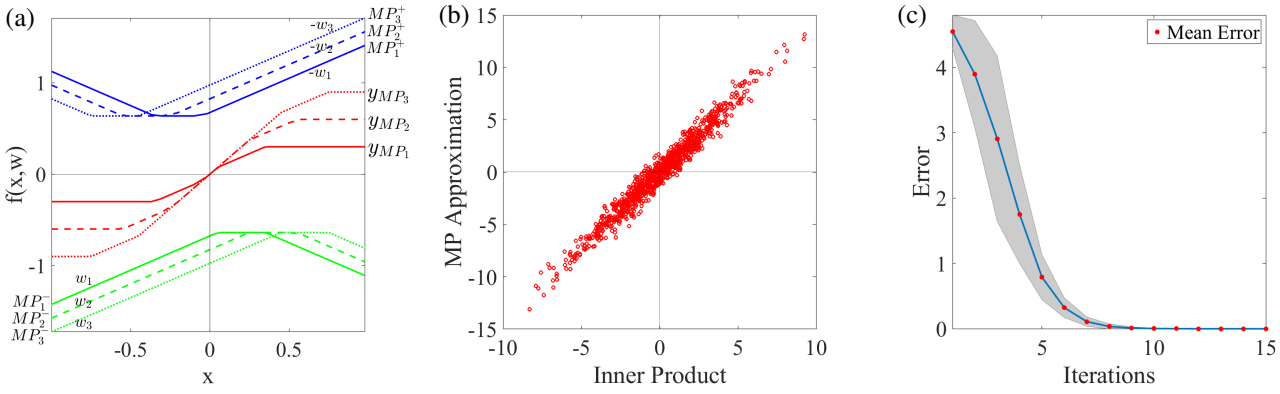

Fig. 3: (a) Scalar Inner Product approximation expressed in MP domain for 3 different w, i.e.,(w_1, w_2, w_3). Here, $MP^+ = MP\left([w+x, -w-x], \gamma\right)$ and $MP^- = -MP\left([w-x, -w+x], \gamma\right)$. (b) Inner Product and MP Approximation scatter plot of 64 dimensional vectors with each input randomly varied between -1 and +1 (c) Variance in error of z value due to shift approximation of $|S|$ becomes zero after 10 iterations

From Fig.2b, we see that for smaller values of the operands, the $y_{LSE}(x)$ approximates the multiplication operation, whereas for larger values $y_{LSE}(x)$ saturates. This effect also applies to general inner products using multi-dimensional vectors, as described by eq.(7). Fig.2c shows the scatter plot that compares the values of y computed using eq.(2) and its log-sum-exponential approximation given by eq.(7), for randomly generated $D = 64$ dimensional vectors $\mathbf{w}$ and $\mathbf{x}$. The plot clearly shows a strong correlation between the two functions, particularly for a smaller magnitude of $||\mathbf{x}||$. Like the quadratic function, implementing the LSE function on digital hardware would also require look-up tables. Other choices of $f(.)$ could also lead to similar multiplier-less approximations of the inner products. However, we are interested in finding the f that can be easily implemented using simple digital hardware primitives.

A. *Margin Propagation based Multiplier-less Approximation*

Margin Propagation (MP) is a piece-wise linear approach for approximating LSE functions [39]. At its core, MP approximation uses only addition, subtraction and thresholding operations and the approach has been previously used to approximate different linear and nonlinear functions [39]. Here we use MP to approximate inner products without using multipliers and propose efficient implementations that can be mapped onto digital hardware. We first express the LSE equation from eq. (4),

$$z_{log} = f(\mathbf{x}, -\mathbf{x}) = \gamma \log\left(\sum_{i=1}^{D} e^{\frac{x_i}{\gamma}} + e^{\frac{-x_i}{\gamma}}\right). \quad (9)$$

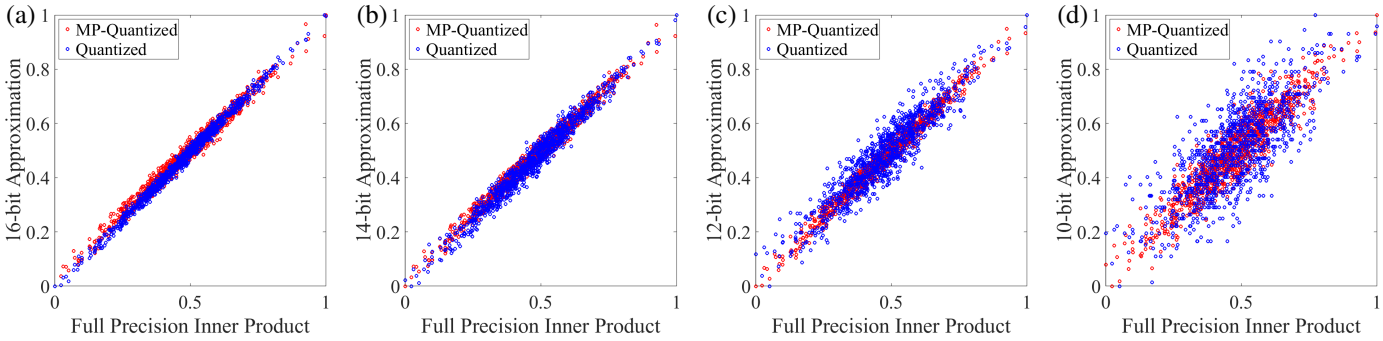

Fig. 4: Impact of quantization and approximation on inner product normalized between 0 and 1 for 1000 pairs of 64 dimensional vector randomly sampled between -1 and 1.

as a constraint

$$\sum_{i=1}^{D} e^{\frac{[x_i - z_{log}]}{\gamma}} + \sum_{i=1}^{D} e^{\frac{[-x_i - z_{log}]}{\gamma}} = 1. \tag{10}$$

Note that without any loss of generality we have introduced $\gamma > 0$ as an additional hyper-parameter. In the MP-based formulation proposed in [39], the exponential function was approximated using a simple piecewise linear model as

$$e^{\frac{x - z_{log}}{\gamma}} \approx \left[\frac{x - z_{MP}}{\gamma} + 1\right]_+. \tag{11}$$

where $[\cdot]_+ = max(\cdot, 0)$ is a rectifying linear operation and z_{MP} is a threshold. Inserting eq. (11) in the constraint eq. (10) leads to

$$\sum_{i=1}^{D} [x_i - z_{MP} + \gamma]_+ + \sum_{i=1}^{D} [-x_i - z_{MP} + \gamma]_+ = \gamma. \tag{12}$$

Thus, for a given input vector $\mathbf{x}$, z_{MP} is computed as a solution to the constraint eq. (12). We will refer

$$z_{MP} \cong MP([\mathbf{x}, -\mathbf{x}], \gamma) \tag{13}$$

as the MP function that will be used in place of $f(\mathbf{x}, -\mathbf{x})$ in eq. (1). Thus,

$$\begin{aligned} y = \mathbf{w}^T\mathbf{x} \approx\ & MP([\mathbf{w} + \mathbf{x}, -\mathbf{w} - \mathbf{x}], \gamma) \\ & - MP([-\mathbf{w} + \mathbf{x}, \mathbf{w} - \mathbf{x}], \gamma). \end{aligned} \tag{14}$$

In addition to being multiplier-less, the inner product approximation in eq. (14) has several properties that make the mapping to digital hardware attractive. First, the formulation is invariant to offsets in inputs as

$$\begin{aligned} & MP([\mathbf{w} + \mathbf{x}, -\mathbf{w} - \mathbf{x}], \gamma) - MP([-\mathbf{w} + \mathbf{x}, \mathbf{w} - \mathbf{x}], \gamma) \\ & = MP([(C + \mathbf{w}) + (C + \mathbf{x}), (C - \mathbf{w}) + (C - \mathbf{x})], \gamma) \\ & - MP([(C - \mathbf{w}) + (C + \mathbf{x}), (C + \mathbf{w}) + (C - \mathbf{x})], \gamma). \end{aligned} \tag{15}$$

where $C > 0$ is some positive quantity. Thus, all operations in eq.(15) requires only unsigned arithmetic, and $[\cdot]_+$ operations. Note that $[\cdot]_+$ operation can be easily implemented on a digital hardware using register underflow. These properties make MP-based approximation suitable for computing inner products and related computation on low-precision digital hardware. Similar to the quadratic and LSE functions, the degree of approximation using MP-function can be visualized for scalars $w \in \mathbb{R}$ and $x \in \mathbb{R}$, as

$$\begin{aligned} y_{MP}(x) = & MP\left([w + x, -w - x], \gamma\right) \\ & - MP\left([w - x, -w + x], \gamma\right). \end{aligned} \tag{16}$$

Fig.3a shows that the MP-function computes a piecewise linear approximation of the LSE function and exhibits the saturation due to register overflow/underflow. Fig.3b shows the scatter plot that compares the true inner product with the MP-based inner product, showing that the approximation error is similar to that of the log-sum-exp approximation in Fig.2c. Thus, MP-function serves as a good approximation to the inner product computation.

B. Implementation of MP-function on Digital Hardware

For the sake of brevity in notation, we will denote the composite differential vector $[\mathbf{x}, -\mathbf{x}]$ as a single vector $\mathbf{x}$ and hence $MP([\mathbf{x}, -\mathbf{x}], \gamma) \equiv MP(\mathbf{x}, \gamma)$. In [39] an algorithm was presented to compute the function $z = MP(\mathbf{x}, \gamma)$ as a solution to the constraint

$$\sum_{i=1}^{D} [x_i - z + \gamma]_+ = \gamma. \tag{17}$$

The approach was based on an iterative identification of the set $S = \{x_i; x_i + \gamma > z\}$ using the reverse water-filling approach. From eq.(17), we get the expression of $MP(\mathbf{x}, \gamma)$ as

$$\sum_{i \in S} (x_i - z) = \gamma \tag{18}$$

and

$$z = MP(\mathbf{x}, \gamma) = -\frac{\gamma}{|S|} + \frac{\sum\limits_{i \in S} x_i}{|S|}. \tag{19}$$

where $|S|$ is the size of the set S. Note that $|S|$ is a function of $MP(\mathbf{x}, \gamma)$ and the expression in eq.(19) requires dividers.

We now report an implementation of MP-function that uses only basic digital hardware primitives like addition, subtraction, shift, and comparison operations. The implementation

TABLE I: Bit width comparison for both types of quantization used in Fig. 4. We use higher precision MP inputs for the same bit width inner product output, resulting in better outputs.

Inner product bit width (64-dimensions)	Input bit width	
	Quantized	MP-Quantized
16-bit	5-bit	9-bit
14-bit	4-bit	7-bit
12-bit	3-bit	5-bit
10-bit	2-bit	3-bit

poses the constraint in eq.(17) as a root-finding problem for the variable z. Then, applying Newton-Raphson recursion [40], the MP function can be iteratively computed as

$$z_n \leftarrow z_{n-1} + \frac{\sum_{i=1}^{D} [x_i - z_{n-1} + \gamma]_+ - \gamma}{|S_{n-1}|}. \tag{20}$$

Here, $S_n = \{x_i; x_i + \gamma > z_n\}$ and $z_n \xrightarrow{n\to\infty} MP(\mathbf{x}, \gamma)$. The Newton-Raphson step can be made hardware-friendly by quantizing S_{n-1} to the nearest upper bound of power of two as $S_{n-1} \approx 2^{P_{n-1}}$ or by choosing a fixed power of 2 in terms of P. Here, $P = floor(log_2(count)) + 1$ as shown in Fig 5. This is implemented in hardware using a priority encoder that checks the highest bit location whose bit value is 1 for the variable $count$. Then, the division operation in eq.(20) can be replaced by a right-shift operation. In Fig. 8, we implement the proposed Newton-Raphson mapping of the MP-function on digital hardware. Since the Netwon-Raphson procedure is an online learning algorithm, any approximation errors that occur due to flooring operations can be compensated by subsequent iterations. Also, note that the Newton-Raphson iteration converges quickly to the desired solution, and hence only a limited number of iterations are required in practice. Fig.3c shows several examples of the MP-function converging to the final value within 10 Newton-Raphson iterations for a 100-dimensional input $\mathbf{x}$. Thus, in our proposed algorithm in Fig 5 used for computing the MP-function we limit the number of Newton-Raphson iterations to 10.

In order to make any algorithm hardware-friendly, the conventional approach is to express it in minimum possible bit width precision with minimal loss in functionality. This would help reduce area and power when implementing digital hardware. In Fig. 4, we see the impact of reducing the bit precision on the approximation of the inner product for a 64-dimensional input vector sampled between -1 and 1. We see the variance in inner product for MP approximation exists, which can be termed as the MP approximation error, compared to the standard quantization of the inner product even for higher bit precision (Fig. 4(a) and 4(b)). We use the online learning approach, detailed in Section V, for MP approximation in our system with minimal hardware increase to mitigate this approximation error. However, as we reduce the bit precision further, we see the quantization error increases and overlaps the MP approximation error (Fig. 4(c) and 4(d)). We see in Table I, for an n-bit inner product output, the standard quantized version requires $\frac{(n-6)}{2}$ - bit input vector, whereas the quantized MP inner product requires $(n-6-1)$ - bit input vector due to addition operation instead of multiplication. Here, we use 6-bits for the 64-dimensional input vector. This shows that MP-function approximates the inner product at a better precision for the same output bit width. Note that when MP-based functions are used in machine learning architectures, the approximation errors can be further compensated by using precision-aware training [41] or by using online learning that computes using MP-based error gradients. In the later sections, we present an online training algorithm that uses the exact form of error gradients computed using MP-based inner products.

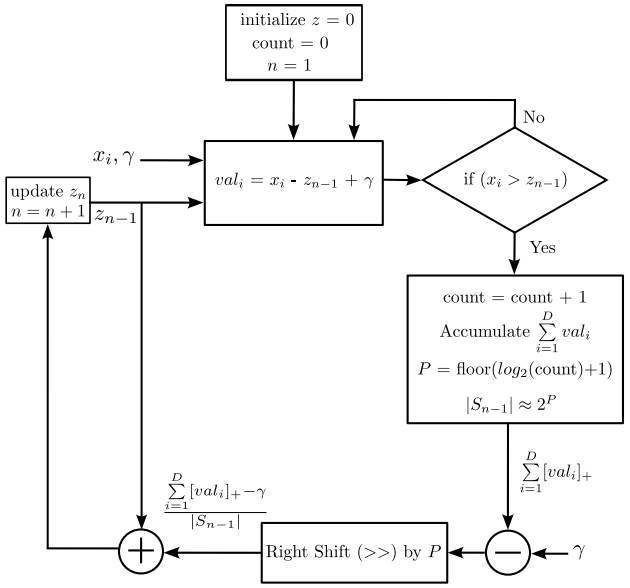

Fig. 5: Newton Raphson method based MP formulation. $|S|$ is the approximated value of variable *count* and represented as the nearest upper bound power of 2.

C. Energy-cost of MP-based Computing

Let C, C_M, and C_A denote the total energy-cost corresponding to an MVM operation, a single multiplication operation, and a single addition operation, respectively. Then, MVM of a $1 \times M$ vector and a $M \times M$ matrix would incur an energy-cost of

$$C = M^2 \times C_M + (M^2 - M) \times C_A. \tag{21}$$

For an MP-based approximation of the MVM, the energy cost incurred is

$$C_{MP} = (M^2 + (M \times F \times R)) \times C_A + M \times R \times C_C. \tag{22}$$

Here, F is the sparsity factor determined by γ, typically having a value less than 1, and C_C is the energy-cost for a comparison operation having $\mathcal{O}(1)$ complexity. Also, note that R is the number of Newton-Raphson iterations, which is 10 in our case. Thus, as inputs increase beyond R, MP approximation complexity reduces further compared to that of MVM. Multiplication requires n^2 full adders for n-bit system, i.e., $C_M = n^2 \times C_A$ [42]. C_A has linear complexity, i.e., $\mathcal{O}(n)$. Hence, the MP approximation technique becomes ideal for digital systems as the implementation of adder logic is less complex and low on resource usage than the equivalent multiplier.

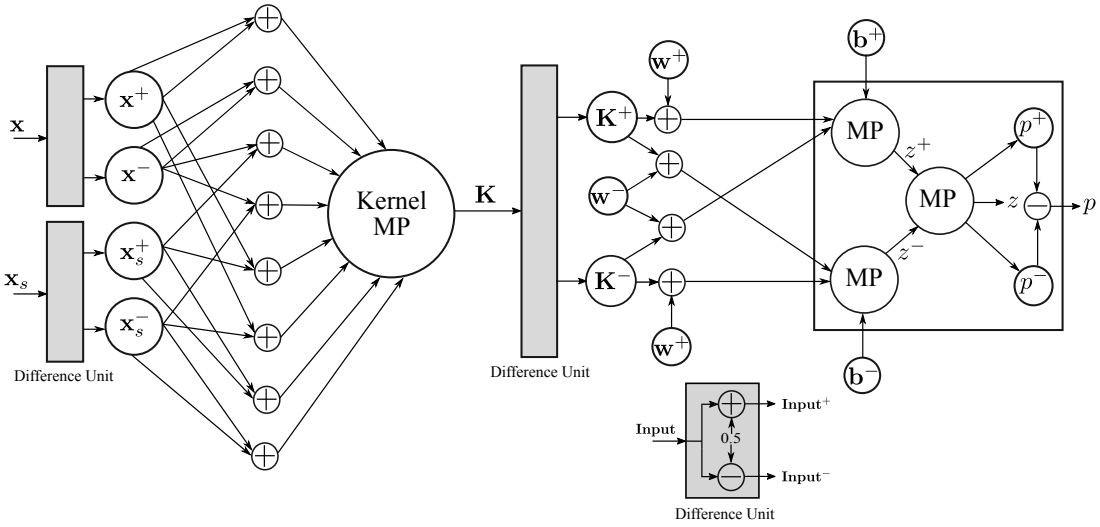

Fig. 6: MP kernel machine architecture. Kernel MP is based on eq. (32) and the other MP functions are described from eq. (27) to (29).

IV. MP Kernel Machine Inference

We now use the MP-based inner product approximation to design an MP kernel machine. Consider a vector $\mathbf{x} \in \mathbb{R}^d$, the decision function for kernel machines [43] is given as,

$$f(\mathbf{x}) = \mathbf{w}^T\mathbf{K} + \mathbf{b}. \tag{23}$$

where $f : \mathbb{R}^d \to \mathbb{R}$, $\mathbf{K} : \mathbb{R}^{d\times d} \to \mathbb{R}^d$ is the kernel which is a function of $\mathbf{x}$, and $\mathbf{w} \in \mathbb{R}^d$, $\mathbf{b} \in \mathbb{R}$ is the corresponding trained weight vector and bias, respectively. As, MP approximation, as shown in eq. (16), is in differential format, we express the variables as $\mathbf{w} = \mathbf{w}^+ - \mathbf{w}^-$, $\mathbf{b} = \mathbf{b}^+ - \mathbf{b}^-$ and $\mathbf{K} = \mathbf{K}^+ - \mathbf{K}^-$.

$$\begin{aligned} f(\mathbf{x}) &= (\mathbf{w}^+ - \mathbf{w}^-)^T(\mathbf{K}^+ - \mathbf{K}^-) + (\mathbf{b}^+ - \mathbf{b}^-). \\ f(\mathbf{x}) &= \left[(\mathbf{w}^+)^T\mathbf{K}^+ + (\mathbf{w}^-)^T\mathbf{K}^- + \mathbf{b}^+\right] \\ &\quad - \left[(\mathbf{w}^+)^T\mathbf{K}^- + (\mathbf{w}^-)^T\mathbf{K}^+ + \mathbf{b}^-\right]. \end{aligned} \tag{24}$$

Using eq.(2), and applying MP approximation based on eq.(16), we can express eq.(24) as,

$$\begin{aligned} f_{MP}(\mathbf{x}) = MP([\mathbf{w}^+ + \mathbf{K}^+, \mathbf{w}^- + \mathbf{K}^-, \mathbf{b}^+], \gamma) \\ -MP([\mathbf{w}^+ + \mathbf{K}^-, \mathbf{w}^- + \mathbf{K}^+, \mathbf{b}^-], \gamma). \end{aligned} \tag{25}$$

Fig.6 describes the kernel machine architecture using MP approximation. The input is provided to a difference unit to generate $\mathbf{x}^+, \mathbf{x}^-, \mathbf{x}_s^+$ and $\mathbf{x}_s^-$ vectors. The kernel MP generates the kernel output with inputs as a combination of $\mathbf{x}^+, \mathbf{x}^-, \mathbf{x}_s^+, \mathbf{x}_s^-$ based on the kernel used. In our case, we use the kernel mentioned in next section IV-A. This kernel $\mathbf{K}$ is used to produce $\mathbf{K}^+$ and $\mathbf{K}^-$ with the help of the difference unit. The weights and bias generated, described in section V, are used with the kernel combination to generate MP approximation output as below. We can express eq.(25) as,

$$f_{MP}(\mathbf{x}) = z^+ - z^-. \tag{26}$$

where,

$$z^+ = MP([\mathbf{w}^+ + \mathbf{K}^+, \mathbf{w}^- + \mathbf{K}^-, \mathbf{b}^+], \gamma_1). \tag{27}$$

$$z^- = MP([\mathbf{w}^+ + \mathbf{K}^-, \mathbf{w}^- + \mathbf{K}^+, \mathbf{b}^-], \gamma_1). \tag{28}$$

γ_1 is a hyper-parameter that is learned using gamma annealing described in Algorithm 1. We normalize the values for z^+ and z^- for better stability of the system using MP,

$$z = MP([z^+, z^-], \gamma_n). \tag{29}$$

Here, γ_n is the hyper-parameter used for normalization. In this case, $\gamma_n = 1$. The output of the system can be expressed in differential form,

$$p = p^+ - p^-. \tag{30}$$

Here, $p \in \mathbb{R}$, $p^+ + p^- = 1$ and $p^+, p^- \geq 0$. As z is the normalizing factor for z^+ and z^-, we can estimate the output using reverse water filling algorithm [39], which is generated by the MP function in eq.(29) for each class,

$$\begin{aligned} p^+ &= [z^+ - z]_+. \\ p^- &= [z^- - z]_+. \end{aligned} \tag{31}$$

A. MP Kernel Function

We use a similarity measure function, which has similar property as a Gaussian function, between the vectors $\mathbf{x}$ and $\mathbf{x}_s$ for defining Kernel MP approximation as,

$$\begin{aligned} \mathbf{K}^- = MP\,([2\mathbf{x}_s^+, 2\mathbf{x}_s^-, 2\mathbf{x}^+, 2\mathbf{x}^-, \\ \mathbf{x}_s^+ + \mathbf{x}^- + 2, \\ \mathbf{x}_s^- + \mathbf{x}^+ + 2], \gamma_2). \end{aligned} \tag{32}$$

γ_2 is the MP hyper-parameter for kernel. We define $\mathbf{K}^+ = -\mathbf{K}^-$. The kernel function is derived in detail in Section A of the supplementary document.

Complexity of Kernel Machine, based on eq.(21) and (23), can be expressed as,

$$C_{KM} = (M^2 + M) \times C_M + (M^2 + M - 1) \times C_A. \quad (33)$$

and similarly, complexity of kernel machine in MP domain based on eq. (22) can be expressed as,

$$C_{MPKM} = (F \times R \times M + 2 \times M^2 + 1) \times C_A + R \times M \times C_c. \quad (34)$$

The complexity equations show that the MP kernel machine complexity is a fraction of that of traditional SVM. This can be leveraged to reduce the power and hardware resources and increase the speed of operation of the MP kernel machine system over traditional SVM.

V. Online training of MP Kernel Machine

The training of our system requires cost calculation and parameter updates to be done over multiple iterations (τ) using the gradient descent approach, which is described below.

Consider a two-class problem, and the cost function can be written as,

$$E = \sum_{n=1}^{M} |y_n^+ - p_n^+| + |y_n^- - p_n^-|. \quad (35)$$

where M is the number of input samples, y_n^+ and y_n^- are the class labels where $y_n^+, y_n^- \in 0, 1$ p_n^+ and p_n^- are the respective predicted values. $p_n^+, p_n^- \geq 0$. We have selected the absolute cost function as it is easier to implement on hardware, as we require fewer bits to represent this cost function than the squared error function.

The weights and biases are updated during each iteration using gradient descent optimization,

$$\mathbf{w}^+ \leftarrow \mathbf{w}^+ - \eta \Bigg[\sum_{n=1}^{M} \Bigg((\mathrm{sgn}^+) \left(1 - \frac{1}{|S|}\right) \left(I_{(z^+)} \frac{1}{|S_p|} I_{(w^+K^+)} - \frac{1}{|S_n|} I_{(w^+K^-)} \right) + (\mathrm{sgn}^-) \left(1 - \frac{1}{|S|}\right) \left(I_{(z^-)} \frac{1}{|S_n|} I_{(w^+K^-)} - \frac{1}{|S_p|} I_{(w^+K^+)} \right) \Bigg) \Bigg]. \quad (36)$$

$$\mathbf{w}^- \leftarrow \mathbf{w}^- - \eta \Bigg[\sum_{n=1}^{M} \Bigg((\mathrm{sgn}^+) \left(1 - \frac{1}{|S|}\right) \left(I_{(z^+)} \frac{1}{|S_p|} I_{(w^-K^-)} - \frac{1}{|S_n|} I_{(w^-K^+)} \right) + (\mathrm{sgn}^-) \left(1 - \frac{1}{|S|}\right) \left(I_{(z^-)} \frac{1}{|S_n|} I_{(w^-K^+)} - \frac{1}{|S_p|} I_{(w^-K^-)} \right) \Bigg) \Bigg]. \quad (37)$$

$$\mathbf{b}^+ \leftarrow \mathbf{b}^+ - \eta \Bigg[\sum_{n=1}^{M} (\mathrm{sgn}^+) \left(1 - \frac{1}{|S|}\right) I_{(z^+)} \frac{1}{|S_p|} I_{(\mathbf{b}^+)} \Bigg]. \quad (38)$$

$$\mathbf{b}^- \leftarrow \mathbf{b}^- - \eta \Bigg[\sum_{n=1}^{M} (\mathrm{sgn}^-) \left(1 - \frac{1}{|S|}\right) I_{(z^-)} \frac{1}{|S_n|} I_{(\mathbf{b}^-)} \Bigg]. \quad (39)$$

where η is the learning rate, $\mathrm{sgn}^+ = \mathrm{sgn}(p_n^+ - y_n^+)$, $\mathrm{sgn}^- = \mathrm{sgn}(p_n^- - y_n^-)$, $I_{(z^+)} = \mathbb{1}(z^+ > z)$, $I_{(z^-)} = \mathbb{1}(z^- > z)$, $I_{(w^+K^+)} = \mathbb{1}(\mathbf{w}^+ + \mathbf{K}^+ > z^+)$, $I_{(w^+K^-)} = \mathbb{1}(\mathbf{w}^+ + \mathbf{K}^- > z^-)$, $I_{(w^-K^+)} = \mathbb{1}(\mathbf{w}^- + \mathbf{K}^+ > z^+)$, $I_{(w^-K^-)} = \mathbb{1}(\mathbf{w}^- + \mathbf{K}^- > z^-)$, $I_{(\mathbf{b}^+)} = \mathbb{1}(\mathbf{b}^+ > z^+)$ and $I_{(\mathbf{b}^-)} = \mathbb{1}(\mathbf{b}^- > z^-)$. $\mathbb{1}$ is the indicator function. An indicator function is defined on a set X indicating membership of an element in a subset A of X, having the value 1 for all elements of X in A and value 0 for all elements of X not in A. The gradient descent steps have been derived in detail in Section B of the supplementary document

Algorithm 1: Gamma Annealing. E_τ is the value of cost function (35) estimated at iteration τ. Here, ϵ and δ are empirical values based on the input dataset. $iter$ is the number of iteration for the MP gradient update.

Input: E_τ,δ,ϵ,$iter$
Output: γ_1
Intialize $\gamma_1 = c$; `// Initalize to a value c based on input for MP function`
for $\tau = 2$ *to* $iter$ **do**
  if $(E_{\tau-1} - E_\tau) > \delta$ **then**
    $\gamma_1 = \gamma_1 - \epsilon$;
  end
end

The parameter γ in eq.(19) impacts the output of MP function and hence can be used as a hyper-parameter in the learning algorithm for the 2nd layer as per Fig.6, i.e., γ_1. This value is adjusted each iteration based on the change in the cost between two consecutive iterations as described in Algorithm 1.

Since the gradient is obtained using the MP approximation technique, the training of the system is more tuned to minimizing the error rather than mitigating the approximation itself and hence improves the overall accuracy of the system.

As we see in all these equations, we require only basic primitives such as comparator, shift operators, counters, and adders to implement this training algorithm, making it very hardware-friendly in terms of implementation.

VI. FPGA Implementation of MP Kernel Machine

The FPGA implementation of the proposed multiplierless kernel machine is described in this section. It is a straight forward hardware implementation of the MP-based mathematical formulation proposed in section IV. The salient features of the proposed design are as follows:

- The proposed architecture is designed to support 32 input features and a total of 256 kernels. It can easily be

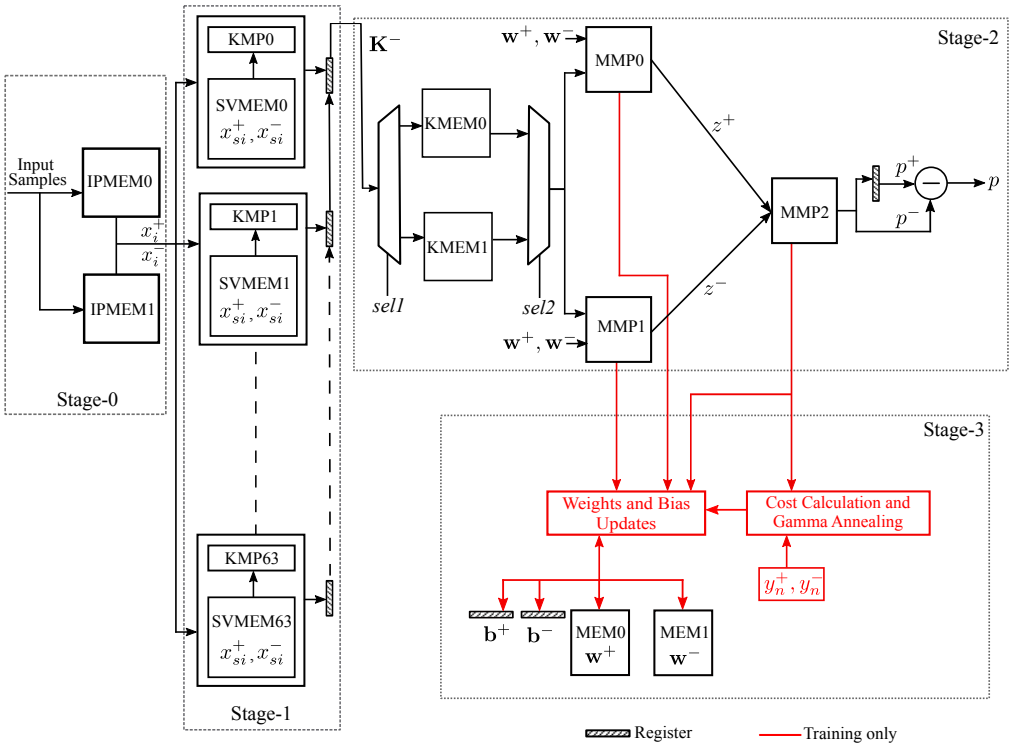

Fig. 7: High-level block diagram of MP kernel machine. The proposed design shares hardware resources during both the training and inference phase. The red blocks are used only during training.

modified to support any number of features and kernel sizes by increasing or decreasing the MP processing blocks.

- The design parameters are set to support 32 input features (in differential mode $x_i^+ \in \mathbb{R}^{32}$ and $x_i^- \in \mathbb{R}^{32}$) and 256 stored vectors (in differential form $x_{si}^+ \in \mathbb{R}^{32}$ and $x_{si}^- \in \mathbb{R}^{32}$). Here, $x_i^+, x_i^- \in \mathbf{x}$, and $x_{si}^+, x_{si}^- \in \mathbf{x}_s$.
- In this design, the width of the data path is set to 12-bits.
- The proposed design includes inference and training and does not consume any complex arithmetic modules like multiplier or divider, only uses adder, subtractor, and shifter modules.
- The resource sharing between training and inference modules saves a significant amount of hardware resources. The weight vectors (also in differential form $\mathbf{w}^+$ and $\mathbf{w}^-$) calculation, as shown in red in Fig.7, are the additional blocks required for training.

A. Description of the proposed design:

The high-level block diagram of the proposed MP-kernel machine, which can support both online training and inference, is shown in Fig.7. The architecture has four stages. Stage 0 shows a memory management unit for storing input samples from the external world to input memory blocks. The MP-based kernel function computation is described in Stage 1. Stage 2 provides a forward pass to generate necessary outputs for inference and training based on the kernel function output obtained from Stage 1. Stage 3 is used only for online training and calculates weight vectors ($\mathbf{w}^+$,$\mathbf{w}^-$) and bias values ($\mathbf{b}^+$, $\mathbf{b}^-$). Here all the stages execute in a parallel manner after receiving the desired data and control signals.

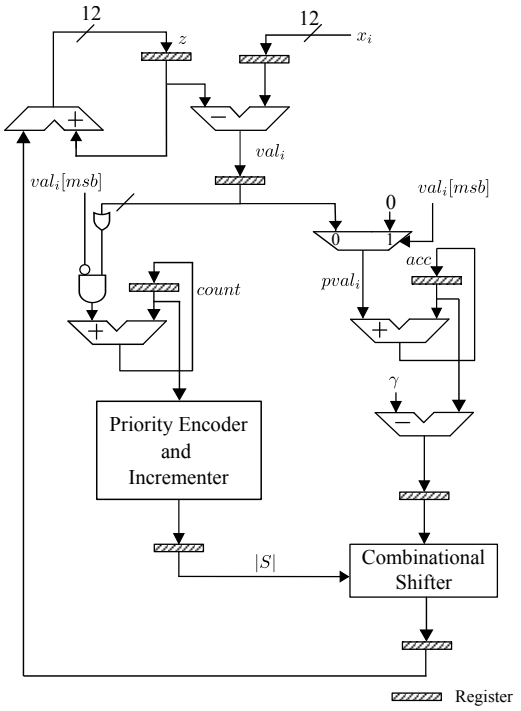

Fig. 8: Architecture of an MP processing block.

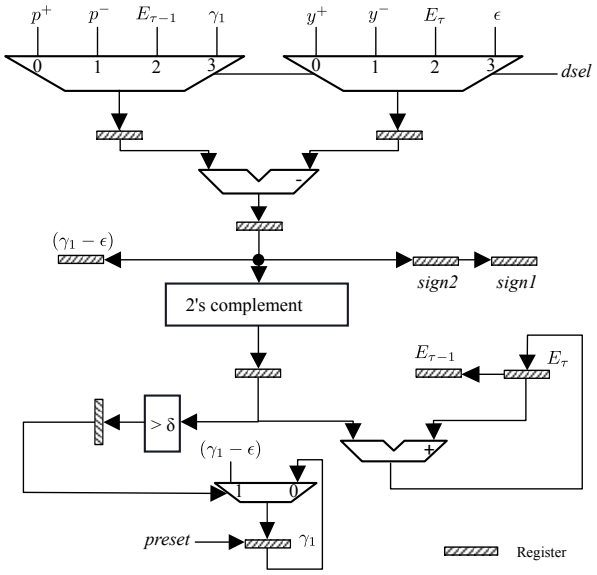

Fig. 9: Architecture for cost function calculation and gamma annealing.

Stage 0 (Accessing inputs from the external world): The input features (x_i^+, x_i^- $\in \mathbf{x}$) of a sample are stored in Block RAM (BRAM), either IPMEM0 or IPMEM1. These BRAMs act as a ping-pong buffer. When an incoming sample is being written into IPMEM0, the kernel computation is carried out on the previously acquired sample in IPMEM1 and vice-versa. The dimension of each input memory block is 64×9-bit.

Stage 1 (Kernel function computation): The architectural design of an MP processing block is shown in Fig.8, the straight forward implementation of the algorithm from Fig. 5. The inputs x_i appear in the MP unit serially i.e. one input in each clock cycle, and calculates val_i. The val_i having positive value is getting accumulated in acc register, where as number of positive terms is counted on $count$ register. The msb of val_i is used to detect positive terms. The val_i is passed through the OR gate to perform bit-wise OR and the output is used to discard val_i with value zero. After accessing all the inputs, the $count$ value is approximated to nearest upper bound of power of two, i.e. $|S| \approx 2^P$ and $P = floor(log_2(count)) + 1$. This is implemented using a simple priority encoder followed by an incrementer. The priority encoder checks the highest bit location whose bit value is 1 for the variable $count$.

The combinational shifter right shifts the $(acc - \gamma)$ value by P number of bits. After that, z is updated with the summation of the previous value of z and shifter output. This process iterates 10 times to get the final value of z. approximation is an iterative method and generates optimum z value after 10 iterations. The high-level architecture for computing MP kernel function ($\mathbf{K}^-$) represented in eq.(32), is shown in Fig.7. Here, 64 MP processing blocks (KMP0-63) are executed parallely and reuses these blocks 4 times to generate a kernel vector ($\mathbf{K}^-$) of dimension 256×1. Each MP processing block is associated with a dual-port BRAM named SVMEM for storing 4 support vectors. Here, SVMEM0 for KMP0 stores support vectors with indices 0, 64, 128, and 192. Similarly, SVMEM1 stores the vectors x_{si} with indices 1, 65, 129, and 193. The rest of the support vectors are stored similarly in the respective SVMEM. An MP processing block receives the inputs serially as mentioned in eq.(32), i.e., $2x_i^+$, $2x_{si}^-$, $x_{si}^- + x_i^+ + 2$, $2x_i^-$, $2x_{si}^+$, $x_{si}^+ + x_i^- + 2$ and generates output z after 10 iterations. Here, all the 64 MP processing blocks receives inputs and generates outputs simultaneously. The generated kernel vector $\mathbf{K}^-$ is stored in BRAM either KMEM0 or KMEM1 based on the select signal $sel1$ (working as a ping-pong buffer). Here, $\mathbf{K}^+$=$-\mathbf{K}^-$.

Stage 2 (Merging of Kernel function output):

In this stage, three MP processing blocks (MMP0-2) are arranged in a layered manner. In the first layer, two MP processing blocks, i.e., MMP0 and MMP1, execute simultaneously, and in the second layer, MMP2 starts processing after receiving outputs from the first layer. The MMP0 implements the eq.(27) and generates output z^+. The inputs arrive at the MP block serially in the following order ($\mathbf{w}^+$+ $\mathbf{K}^+$), ($\mathbf{w}^-$+ $\mathbf{K}^-$), $\mathbf{b}^+$. Similarly, MMP1 takes ($\mathbf{w}^+$+ $\mathbf{K}^-$), ($\mathbf{w}^-$+ $\mathbf{K}^+$), $\mathbf{b}^-$, and γ_1 as inputs and produces z^- as the output according to eq.(28).

In this stage, $\mathbf{K}^-$ is accessed from either KMEM0 or KMEM1 based on $sel2$ signal, $\mathbf{w}^+$ and $\mathbf{w}^-$ are accessed from MEM0 and MEM1, respectively.

The two outputs (z^+ and z^-) generated from the first layer are provided as inputs to MMP2 along with γ_1. This module generates outputs p^+ and p^- according to the eq.(31). The final prediction value p is computed based on the eq.(30). In the Fig.8 $pval_1$ = p^+ and $pval_2$ = p^- at 10^{th} round.

Stage 3 (Weights and bias update module): This stage executes only during the training cycle. The detailed hardware architecture for updating weights and bias is shown in Fig.10. The proposed design performs error gradients update ($\frac{\partial E}{\partial \mathbf{w}^+}$, $\frac{\partial E}{\partial \mathbf{w}^-}$, $\frac{\partial E}{\partial \mathbf{b}^+}$, $\frac{\partial E}{\partial \mathbf{b}^-}$) followed by weights and bias update at the end of each iteration (τ) according to eq.(36)-(39). The hardware resources of stages 0, 1 and 2 are shared by both the training and inference cycles. This stage updates weight vectors ($\mathbf{w}^+$, $\mathbf{w}^-$) and bias values ($\mathbf{b}^+$, $\mathbf{b}^-$) based on the parameters generated at Stage 2.

At the training phase, outputs $|S_p|$, $|S_n|$ and $|S|$ are generated from MMP0, MMP1, and MMP2, respectively, and stored in three different registers. After that, two parameters $q = \left(1 - \frac{1}{|S|}\right)\frac{1}{|S_p|}$ and $r = \left(1 - \frac{1}{|S|}\right)\frac{1}{|S_n|}$ are calculated using the adder and combinational shifter modules. The precomputed values q and r are used to update the weights and bias values according to eq.(36)-(39).

The sign bit, i.e., $val_i^{(0)}[msb]$ and $val_i^{(1)}[msb]$, originated from MMP0 and MMP1 respectively at the 10^{th} round are passed through two separate 8-bit serial in parallel out (s2p) registers to generate 8-bit data which is stored in two separate BRAMs (MEM2 and MEM3, respectively). The memory contents are accessed through an 8-bit parallel in serial out (p2s) register during processing. Similarly, the sign bits $val_1^{(2)}[msb]$ and $val_2^{(2)}[msb]$ from MMP2 are also stored in two separate registers. The sign bits are used to implement indicator function ($\mathbb{1}$) represented in eq.(36)-(39). An architecture that calculates both cost function (E) and gamma annealing is shown in Fig. 9. The architecture works in two phases, in

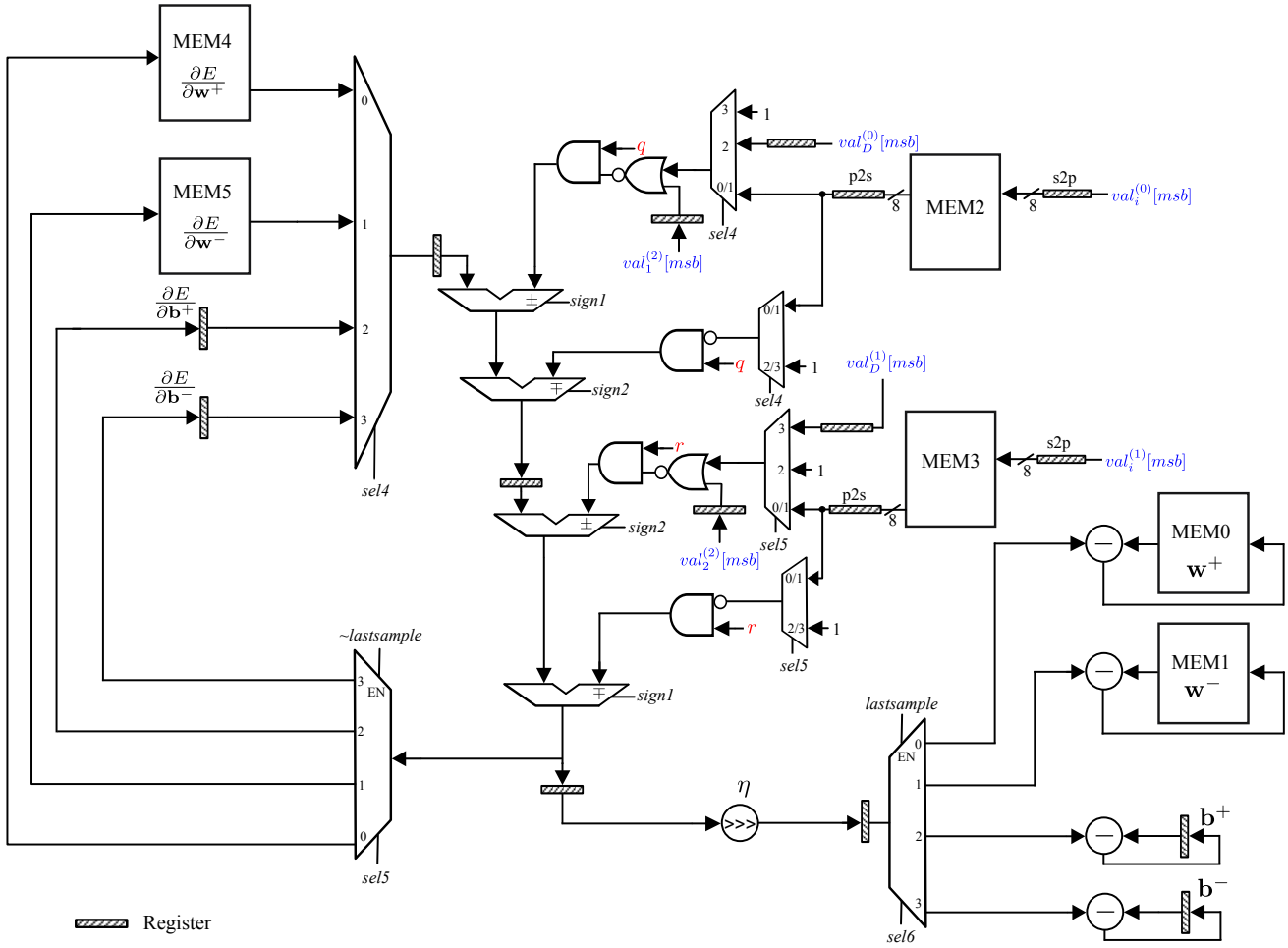

Fig. 10: Architecture for weights and bias update module. The final weights ($\mathbf{w}^+$, $\mathbf{w}^-$) and bias values ($\mathbf{b}^+$, $\mathbf{b}^-$) are stored in the corresponding memory blocks (MEM0 and MEM1) and registers, respectively. The parameters $q = \left(1 - \frac{1}{|S|}\right)\frac{1}{|S_p|}$ and $r = \left(1 - \frac{1}{|S|}\right)\frac{1}{|S_n|}$ (shown in red) are calculated using the adder and combinational shifter modules. The val_i (shown in blue) values are provided by the MP processing block Fig.8

the first phase ($dsel[msb] = 0$), the cost function is calculated according to eq.(35), and in the second phase ($dsel[msb] = 1$), gamma annealing is performed as per the Algorithm 1 mentioned in Section V. The $preset$ signal initializes γ_1 register and the register content getting updated in each iteration (τ) according to the Algorithm 1. In the first phase, the module also calculates $\text{sgn}(p^+ - y^+)$, $\text{sgn}(p^- - y^-)$ and stores the outputs in two registers ($sign1$ and $sign2$), respectively, which are used as inputs to the weights and bias update module.

The datapath for updating error gradients as well as weights and bias updates are shown in Fig. 10. The values of $\frac{\partial E}{\partial \mathbf{w}^+}$ and $\frac{\partial E}{\partial \mathbf{w}^-}$ are stored in MEM4 and MEM5 BRAMs (256×9-bit each), respectively, and the values of $\frac{\partial E}{\partial \mathbf{b}^+}$ and $\frac{\partial E}{\partial \mathbf{b}^-}$ are stored in registers. At the beginning of each iteration (τ), these are initialized with zero. The error gradients are accumulated for each sample, and the respective storage is updated. The mathematical formulation for the error gradient update is explained in Section B of the supplementary document.

In Fig. 10, the MUXes are used to select appropriate inputs for the adder-subtractor modules. The signals $sign1$ and $sign2$ are used to select the appropriate operations i.e., either addition or subtraction. The $sel5$ signal is generated by delaying $sel4$ by one clock cycle. When the msb of $sel4$ is 0, it accesses and updates the MEM4 and MEM5 alternatively in each clock cycle. After that, the msb of $sel4$ becomes high and updates the registers. While processing the last sample of an iteration (τ), an active high signal $lastsample$ along with $sel6$ starts updating the MEM0 and MEM1 for $\mathbf{w}^+$ and $\mathbf{w}^-$, and the registers for $\mathbf{b}^+$ and $\mathbf{b}^-$. Here, the learning rate (η) is expressed in powers of 2 and can be implemented using a combinational shifter module.

VII. Results and Discussion

The proposed design has been implemented on Artix-7 (xc7a25tcpg238-3), manufactured at 28 nm technology node, a low-powered, highly efficient FPGA. Artix-7 family devices have been extensively used in edge device applications like automotive sensor processing. This makes it ideal for us to showcase our design capability on this device. Our design is capable of running at a maximum frequency of 200 MHz.

TABLE II: Comparison of Architecture and Resource Utilization of Related Work.

Hardware Comparison	Kuan et.al [29]	Peng et.al [44]	Peng et.al [45]	Feng et.al [46]	Jiang et.al [26]	Mandal et.al [31]	This Work
FPGA	Cyclone II DE2-70	Spartan-3 c3s4000	Cyclone II 2C70	Cyclone II ep2c70f896C6	Zynq 7000	Virtex 7vx485tffg1157	Artix-7 xc7a25tcpg238-3
Operating Frequency	50 MHz	50 MHz	50 MHz	50 MHz	NA+	NA+	62 MHz (Max.200 MHz)*
Multipliers	58	NA+	NA+	41	152	515	0
Resources	6395 (LEs)	5612 (LEs)	5755 (LEs)	6839 (LEs)	21305 (FFs) 14028*(LUTs)	19023 (FFs) 19023 (LUTs)	9735 (Max. 9788)*(FFs) 9535 (Max. 9874)*(LUTs)
RAM (Kbits)	445	432	456	304	NA+	NA+	317
Input Vector Size	16	16	16	8	NA+	12	32
Vector bit length	24-bit	24-bit	24-bit	16-bit	25-bit	NA+	12-bit
Training Cycles	454M	13M	60M	2964K	NA+	NA+	2054K
On-Chip Classification Support	No	No	Yes	No	No	No	Yes
Dynamic Power	216 mW#	206 mW#	195 mW#	45 mW#	NA+	NA+	107 mW
Energy Consumed†	116 pJ	NA+	NA+	45.1 pJ	NA+	NA+	13.4 pJ

+ These works did not report the corresponding values for their designs.
* These values correspond to design changes for operating at a frequency of 200 MHz.
# These work do not state explicitly that the power consumed is dynamic.
† Based on energy consumption analysis in [18] for different operations (Multipliers and Adders) on 45 nm technology node.
Note that LUTs/FFs from Xilinx and LEs from Intel are not equivalent.

The design supports both inference and on-chip training. It consumes about 9874 LUTs, 9788 FFs, along with 35 BRAMs (36 Kb). The design does not utilize any complex arithmetic module such as multiplier. Table **??** summarizes implementation results. For processing a sample, the proposed MP-kernel machine consumes 8024 clock cycles for computing a (256×1) kernel vector $\mathbf{K}^-$ (Stage 1). Stage 2 consumes 5256 and 5710 clock cycles during inference and learning, respectively. Stage 3, which is active during learning, consumes 524 clock cycles. In top level, the proposed design has two sections : section 1, executes stage 1 and section 2, executes stage 2 and stage 3. Two sections work in pipeline fashion. In real time applications, when pipe is full, it can generate one output after 8024 clock cycles.

TABLE III: Comparison of Traditional SVM Inference and MP Kernel Machine (Training and Inference) Resource Utilization at operating frequency of 62 MHz.

Hardware Resources	Traditional SVM (Inference)	MP Kernel Machine (Training and Inference)	
FPGA	xc7z020-1 clg400c	xc7z020-1 clg400c	xc7a25 tcpg238-3
FFs	6148	9734	9735
LUTs	18141	9572	9535
BRAMs (36k)	46	35	35
DSPs	192	0	0
Dynamic Power	320 mW	107 mW	107 mW

We compare our system with similar SVM systems in the literature. From Table II, it is clear that our system consumes the least amount of energy of 13.4 pJ compared to similar SVM systems with online training capability. Based on the type and number of operations used in each design and the energy consumed by each type of operator, using [18] as the reference, we arrive at the energy consumption of each system. Our system can process an input vector size of 32, which is higher than other systems, and at the same time consumes lower RAM bits with no DSP usage making it resource efficient. The amount of training cycles required by our system is lowest, i.e, around 2054K, providing lower latency and higher throughput. A traditional SVM inference algorithm is also implemented on the PYNQ board (xc7z020clg400-1), manufactured at 28 nm technology node, to compare resource utilization and power consumption with the proposed MP kernel machine algorithm. We used linear kernel for this implementation as non-linear kernel was complex for implementation. The hardware design for the traditional SVM algorithm consumed a higher number of resources, and due to this, we were unable to fit the design in the same FPGA part used for the kernel machine design. To obtain a fair comparison, we implemented our design on the same PYNQ board. The design parameters for both designs are the same to make a direct comparison and bring out the efficiency of our system. The maximum operating frequency of the traditional SVM design is limited to 62 MHz. The power and efficiency of our system, which includes training and inference, is compared to this traditional SVM implementation having only an inference engine in Table III. We see that our MP kernel machine consumes a fraction of the power and about half of the LUTs (including the training engine) in comparison to the traditional SVM. Due to the multiplierless design, we see no DSPs consumed compared to 192 DSPs in traditional SVM design. For edge devices, the requirement of a small footprint and low power is fulfilled by our system.

We used datasets from different domains for classification to benchmark our system. We used the occupancy detection dataset from the UCI ML repository [50] [51], which detects whether a particular room is occupied or not based on different sensors like light, temperature, humidity, and CO_2. We also verified our system on the Activity Recognition dataset (AReM) [52]. We used a one versus all approach on the AReM dataset for binary classification using this system. We chose two of the activities as positive cases, i.e., bending and lying activities, to verify the classification the capability of our system. Free Spoken Digits Dataset (FSDD) was used to showcase the capability of our system for speech applications

TABLE IV: Accuracy in % for UCI Datasets in Percent.

Datasets	Full Precision				Fixed Point (12-bit)	
	Traditional SVM*		MP SVM		MP SVM	
	Train	Test	Train	Test	Train	Test
Occupancy Detection	98.6 [47]	97.8 [47]	98	94	97.9	93.8
FSDD Jackson	99 [48]	99 [48]	97.5	96	97	96
AReM Bending	96 [49]	96 [49]	96	94.1	96	93.2
AReM Lying	96 [49]	96 [49]	96	94.7	95.9	94

* These values are reported in different works.

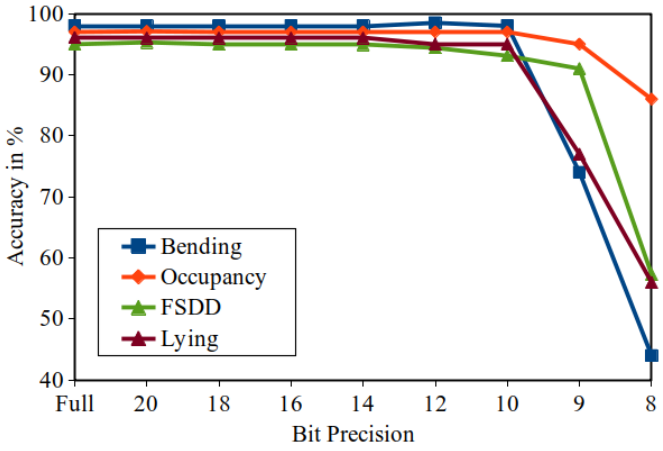

Fig. 11: Bit precision vs. Accuracy for Datasets

[53]. Here, we used this dataset for the speaker verification task. We identified a speaker named Jackson among 3 other speakers. Mel-Frequency Cepstral Coefficients (MFCC) is used as a feature extractor for this classification.

Since our hardware currently supports 256 input samples, we truncated the datasets to 256 inputs. We performed a k-fold cross-validation to generate 256 separate train and test samples per fold from the original dataset to arrive at the results. This was repeated to cover the entire size of the dataset and get the average accuracy results across separate runs.A MATLAB model of the proposed architecture is developed to determine the datapath precision. We can see from Fig.11, that the dataset's accuracy remains more or less constant as we reduce the bit width precision up to 12-bit. Below 12-bit, accuracy starts degrading due to quantization error. Hence, we used 12-bit precision for implementing the datapath. The accuracy degradation between full precision MATLAB model and fixed point (12-bit) RTL versions were minimal, as shown in Table IV. We compared the results of traditional SVM from independent works using the same datasets with our MP-based system. Despite being an approximation, we can get the accuracy results of our system comparable to the traditional SVM systems. From these results, our system demonstrates its capability in classification tasks, and also, with minor changes to the hardware, it can adapt to any application.

VIII. Conclusion

In this paper, we show a novel algorithm used for classification in edge devices using MP function approximation. This algorithm proves to be hardware-friendly since the training and inference algorithm can be implemented using basic primitives like adders, comparators, counters, and threshold operations. The unique training formulation for kernel machines is lightweight and hence enables online training. The same hardware is used for training and inference, leveraging hardware reuse policy to make it a highly efficient system. Also, the system is highly scalable without requiring significant hardware changes. In comparison to traditional SVMs, we were able to achieve low power and computational resource usage, making it ideal for edge device deployment. This algorithm being multiplierless improves the speed of operation when compared to traditional SVMs. Such edge devices can be deployed in remote locations for surveillance and medical applications, where human intervention may be minimal. We can fabricate this system into Application Specific Integrated Chip to make it more power and area efficient. Moreover, this algorithm can be extended to more complex ML systems like deep learning to leverage the low hardware footprint.

Acknowledgment

This research was supported in part by SPARC grant (SPARC/2018-2019/P606/SL) and SRAship grant from CSIR-HRDG, and IMPRINT Grant (IMP/2018/000550) from the Department of Science and Technology, India. The authors would like to acknowledge the joint Memorandum of Understanding (MoU) between Indian Institute of Science, Bangalore and Washington University in St. Louis for supporting this research activity.

References

[1] M. Satyanarayanan, "The emergence of edge computing," *Computer*, vol. 50, no. 1, pp. 30–39, 2017.

[2] W. Shi, J. Cao, Q. Zhang, Y. Li, and L. Xu, "Edge computing: Vision and challenges," *IEEE internet of things journal*, vol. 3, no. 5, pp. 637–646, 2016.

[3] S. Wang, T. Tuor, T. Salonidis, K. K. Leung, C. Makaya, T. He, and K. Chan, "When edge meets learning: Adaptive control for resource-constrained distributed machine learning," in *IEEE INFOCOM 2018-IEEE Conference on Computer Communications*. IEEE, 2018, pp. 63–71.

[4] G. Zhu, D. Liu, Y. Du, C. You, J. Zhang, and K. Huang, "Toward an intelligent edge: wireless communication meets machine learning," *IEEE Communications Magazine*, vol. 58, no. 1, pp. 19–25, 2020.

[5] H. Li, K. Ota, and M. Dong, "Learning iot in edge: Deep learning for the internet of things with edge computing," *IEEE network*, vol. 32, no. 1, pp. 96–101, 2018.

[6] K. Flouri, B. Beferull-Lozano, and P. Tsakalides, "Training a svm-based classifier in distributed sensor networks," in *2006 14th European Signal Processing Conference*. IEEE, 2006, pp. 1–5.

[7] Q. Cui, Z. Gong, W. Ni, Y. Hou, X. Chen, X. Tao, and P. Zhang, "Stochastic online learning for mobile edge computing: Learning from changes," *IEEE Communications Magazine*, vol. 57, no. 3, pp. 63–69, 2019.

[8] S. Han, H. Mao, and W. J. Dally, "Deep compression: Compressing deep neural networks with pruning, trained quantization and huffman coding," *arXiv preprint arXiv:1510.00149*, 2015.

[9] M. Courbariaux, Y. Bengio, and J.-P. David, "Binaryconnect: Training deep neural networks with binary weights during propagations," *Advances in neural information processing systems*, vol. 28, 2015.

[10] M. Rastegari, V. Ordonez, J. Redmon, and A. Farhadi, "Xnor-net: Imagenet classification using binary convolutional neural networks," in *European conference on computer vision*. Springer, 2016, pp. 525–542.

[11] I. Hubara, M. Courbariaux, D. Soudry, R. El-Yaniv, and Y. Bengio, "Quantized neural networks: Training neural networks with low precision weights and activations," *The Journal of Machine Learning Research*, vol. 18, no. 1, pp. 6869–6898, 2017.

[12] S. S. Heydari and G. Mountrakis, "Effect of classifier selection, reference sample size, reference class distribution and scene heterogeneity in per-pixel classification accuracy using 26 landsat sites," *Remote Sensing of Environment*, vol. 204, pp. 648–658, 2018.

[13] C. Cortes and V. Vapnik, "Support-vector networks," *Machine learning*, vol. 20, no. 3, pp. 273–297, 1995.

[14] Y. Tang, Y. Zhang, N. V. Chawla, and S. Krasser, "Svms modeling for highly imbalanced classification," *IEEE Transactions on Systems, Man, and Cybernetics, Part B (Cybernetics)*, vol. 39, no. 1, pp. 281–288, 2008.

[15] A. Palaniappan, R. Bhargavi, and V. Vaidehi, "Abnormal human activity recognition using svm based approach," in *2012 International Conference on Recent Trends in Information Technology*. IEEE, 2012, pp. 97–102.

[16] J. Choi and S. Venkataramani, "Approximate computing techniques for deep neural networks," in *Approximate Circuits*. Springer, 2019, pp. 307–329.

[17] C. K. I. W., "Learning kernel classifiers," *Journal of the American Statistical Association*, vol. 98, no. 462, pp. 489–490, 2003.

[18] M. Horowitz, "1.1 computing's energy problem (and what we can do about it)," in *2014 IEEE International Solid-State Circuits Conference Digest of Technical Papers (ISSCC)*. IEEE, 2014, pp. 10–14.

[19] S. Chakrabartty and G. Cauwenberghs, "Margin propagation and forward decoding in analog vlsi," *A (A)*, vol. 100, p. 5, 2004.

[20] R. Genov and G. Cauwenberghs, "Kerneltron: support vector" machine" in silicon," *IEEE Transactions on Neural Networks*, vol. 14, no. 5, pp. 1426–1434, 2003.

[21] S. Chakrabartty and G. Cauwenberghs, "Forward-decoding kernel-based phone recognition," in *Advances in Neural Information Processing Systems*, 2003, pp. 1189–1196.

[22] K. Kang and T. Shibata, "An on-chip-trainable gaussian-kernel analog support vector machine," *IEEE Transactions on Circuits and Systems I: Regular Papers*, vol. 57, no. 7, pp. 1513–1524, 2009.

[23] S. Chakrabartty and G. Cauwenberghs, "Sub-microwatt analog vlsi trainable pattern classifier," *IEEE Journal of Solid-State Circuits*, vol. 42, no. 5, pp. 1169–1179, 2007.

[24] C. Kyrkou and T. Theocharides, "A parallel hardware architecture for real-time object detection with support vector machines," *IEEE Transactions on Computers*, vol. 61, no. 6, pp. 831–842, 2011.

[25] C. Kyrkou, T. Theocharides, and C. Bouganis, "An embedded hardware-efficient architecture for real-time cascade support vector machine classification," in *2013 International Conference on Embedded Computer Systems: Architectures, Modeling, and Simulation (SAMOS)*. IEEE, 2013, pp. 129–136.

[26] Y. Jiang, K. Virupakshappa, and E. Oruklu, "Fpga implementation of a support vector machine classifier for ultrasonic flaw detection," in *2017 IEEE 60th international midwest symposium on circuits and systems (MWSCAS)*. IEEE, 2017, pp. 180–183.

[27] L. A. Martins, G. A. Sborz, F. Viel, and C. A. Zeferino, "An svm-based hardware accelerator for onboard classification of hyperspectral images," in *Proceedings of the 32nd Symposium on Integrated Circuits and Systems Design*, 2019, pp. 1–6.

[28] K. Han, J. Wang, X. Xiong, Q. Fang, and N. David, "A low complexity svm classifier for eeg based gesture recognition using stochastic computing," in *2020 IEEE International Symposium on Circuits and Systems (ISCAS)*. IEEE, 2020, pp. 1–5.

[29] T.-W. Kuan, J.-F. Wang, J.-C. Wang, P.-C. Lin, and G.-H. Gu, "Vlsi design of an svm learning core on sequential minimal optimization algorithm," *IEEE Transactions on Very Large Scale Integration (VLSI) Systems*, vol. 20, no. 4, pp. 673–683, 2011.

[30] S. Dey, D. Chen, Z. Li, S. Kundu, K. Huang, K. M. Chugg, and P. A. Beerel, "A highly parallel fpga implementation of sparse neural network training," in *2018 International Conference on ReConFigurable Computing and FPGAs (ReConFig)*. IEEE, 2018, pp. 1–4.

[31] B. Mandal, M. P. Sarma, K. K. Sarma, and N. Mastorakis, "Implementation of systolic array based svm classifier using multiplierless kernel," in *2014 International Conference on Signal Processing and Integrated Networks (SPIN)*, 2014, pp. 35–39.

[32] Z. Xue, J. Wei, and W. Guo, "A real-time naive bayes classifier accelerator on fpga," *IEEE Access*, vol. 8, pp. 40 755–40 766, 2020.

[33] H. Chen, Y. Wang, C. Xu, B. Shi, C. Xu, Q. Tian, and C. Xu, "Addernet: Do we really need multiplications in deep learning?" in *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition*, 2020, pp. 1468–1477.

[34] M. Elhoushi, Z. Chen, F. Shafiq, Y. H. Tian, and J. Y. Li, "Deepshift: Towards multiplication-less neural networks," *arXiv preprint arXiv:1905.13298*, 2019.

[35] H. You, X. Chen, Y. Zhang, C. Li, S. Li, Z. Liu, Z. Wang, and Y. Lin, "Shiftaddnet: A hardware-inspired deep network," *Advances in Neural Information Processing Systems*, vol. 33, 2020.

[36] P. Kucher and S. Chakrabartty, "An energy-scalable margin propagation-based analog vlsi support vector machine," in *2007 IEEE International Symposium on Circuits and Systems*. IEEE, 2007, pp. 1289–1292.

[37] M. Sadeghian, J. E. Stine, and E. G. Walters, "Optimized linear, quadratic and cubic interpolators for elementary function hardware implementations," *Electronics*, vol. 5, no. 2, p. 17, 2016.

[38] W. Hlawitschka, "The empirical nature of taylor-series approximations to expected utility," *The American Economic Review*, vol. 84, no. 3, pp. 713–719, 1994.

[39] M. Gu, *Theory, Synthesis and Implementation of Current-mode CMOS Piecewise-linear Circuits Using Margin Propagation*. Michigan State University, Electrical Engineering, 2012.

[40] A. Ben-Israel *et al.*, "A newton-raphson method for the solution of systems of equations," *J. Math. Anal. Appl*, vol. 15, no. 2, pp. 243–252, 1966.

[41] Z. Dong, Z. Yao, A. Gholami, M. W. Mahoney, and K. Keutzer, "Hawq: Hessian aware quantization of neural networks with mixed-precision," in *Proceedings of the IEEE/CVF International Conference on Computer Vision*, 2019, pp. 293–302.

[42] E. A. Vittoz, "Future of analog in the vlsi environment," in *IEEE International Symposium on Circuits and Systems*. IEEE, 1990, pp. 1372–1375.

[43] N. Cristianini, J. Shawe-Taylor *et al.*, *An introduction to support vector machines and other kernel-based learning methods*. Cambridge university press, 2000.

[44] C. Peng, B. Chen, T. Kuan, P. Lin, J. Wang, and N. Shih, "Recsta: Reconfigurable and efficient chip design with smo-based training accelerator," *IEEE Transactions on Very Large Scale Integration (VLSI) Systems*, vol. 22, no. 8, pp. 1791–1802, 2013.

[45] C.-H. Peng, T.-W. Kuan, P.-C. Lin, J.-F. Wang, and G.-J. Wu, "Trainable and low-cost smo pattern classifier implemented via mcmc and sfbs technologies," *IEEE Transactions on Very Large Scale Integration (VLSI) Systems*, vol. 23, no. 10, pp. 2295–2306, 2014.

[46] L. Feng, Z. Li, and Y. Wang, "Vlsi design of svm-based seizure detection system with on-chip learning capability," *IEEE transactions on biomedical circuits and systems*, vol. 12, no. 1, pp. 171–181, 2017.

[47] Z. Liu, J. Zhang, and L. Geng, "An intelligent building occupancy detection system based on sparse auto-encoder," in *2017 IEEE Winter Applications of Computer Vision Workshops (WACVW)*. IEEE, 2017, pp. 17–22.

[48] A. R. Nair, S. Chakrabartty, and C. S. Thakur, "In-filter computing for designing ultra-light acoustic pattern recognizers," *IEEE Internet of Things Journal*, 2021.

[49] D. Anguita, A. Ghio, L. Oneto, X. Parra Perez, and J. L. Reyes Ortiz, "A public domain dataset for human activity recognition using smartphones," in *Proceedings of the 21th international European symposium on artificial neural networks, computational intelligence and machine learning*, 2013, pp. 437–442.

[50] D. Dua and C. Graff, "UCI machine learning repository," 2017, http://archive.ics.uci.edu/ml.

[51] L. M. Candanedo and V. Feldheim, "Accurate occupancy detection of an office room from light, temperature, humidity and co2 measurements using statistical learning models," *Energy and Buildings*, vol. 112, pp. 28–39, 2016.

[52] F. Palumbo, C. Gallicchio, R. Pucci, and A. Micheli, "Human activity recognition using multisensor data fusion based on reservoir computing," *Journal of Ambient Intelligence and Smart Environments*, vol. 8, no. 2, pp. 87–107, 2016.

[53] Z. Jackson, "Free spoken digits dataset," 2016, https://github.com/Jakobovski/free-spoken-digit-dataset.