



Efficient Hardware Architectures for 1D- and MD-LSTM Networks

Vladimir Rybalkin¹ · Chirag Sudarshan¹ · Christian Weis¹ · Jan Lappas¹ · Norbert Wehn¹ · Li Cheng²

Received: 18 September 2019 / Revised: 8 May 2020 / Accepted: 20 May 2020
© The Author(s) 2020, corrected publication 2021

Abstract

Recurrent Neural Networks, in particular One-dimensional and Multidimensional Long Short-Term Memory (1D-LSTM and MD-LSTM) have achieved state-of-the-art classification accuracy in many applications such as machine translation, image caption generation, handwritten text recognition, medical imaging and many more. However, high classification accuracy comes at high compute, storage, and memory bandwidth requirements, which make their deployment challenging, especially for energy-constrained platforms such as portable devices. In comparison to CNNs, not so many investigations exist on efficient hardware implementations for 1D-LSTM especially under energy constraints, and there is no research publication on hardware architecture for MD-LSTM. In this article, we present two novel architectures for LSTM inference: a hardware architecture for MD-LSTM, and a DRAM-based Processing-in-Memory (DRAM-PIM) hardware architecture for 1D-LSTM. We present for the first time a hardware architecture for MD-LSTM, and show a trade-off analysis for accuracy and hardware cost for various precisions. We implement the new architecture as an FPGA-based accelerator that outperforms NVIDIA K80 GPU implementation in terms of runtime by up to 84× and energy efficiency by up to 1238× for a challenging dataset for historical document image binarization from DIBCO 2017 contest, and a well known MNIST dataset for handwritten digits recognition. Our accelerator demonstrates highest accuracy and comparable throughput in comparison to state-of-the-art FPGA-based implementations of multilayer perceptron for MNIST dataset. Furthermore, we present a new DRAM-PIM architecture for 1D-LSTM targeting energy efficient compute platforms such as portable devices. The DRAM-PIM architecture integrates the computation units in a close proximity to the DRAM cells in order to maximize the data parallelism and energy efficiency. The proposed DRAM-PIM design is 16.19× more energy efficient as compared to FPGA implementation. The total chip area overhead of this design is 18 % compared to a commodity 8 Gb DRAM chip. Our experiments show that the DRAM-PIM implementation delivers a throughput of 1309.16 GOP/s for an optical character recognition application.

Keywords Long short-term memory · LSTM · MD-LSTM · 2D-LSTM · FPGA · DRAM · Processing-in-memory · PIM · Optical character recognition · OCR · MNIST · DIBCO · Zynq · Image binarization · Hardware architecture · Deep learning

1 Introduction

In recent years, a wide variety of neural network accelerators [9, 26, 53] have been published that achieve higher performance and higher energy efficiency compared to general purpose computing platforms. Many of these accelerators target feed-forward networks such as CNN, however far

fewer investigations exist on efficient hardware implementations for Recurrent Neural Networks (RNNs), and Long Short-Term Memory (LSTM) in particular, especially under energy constraints.

Compared to feed-forward Neural Networks (NNs), efficient implementation of LSTM networks are especially challenging for the following reasons: first, LSTM, namely One-dimensional Long Short-Term Memory (1D-LSTM) has a recurrence along a time-axis in speech recognition or writing direction in Optical Character Recognition (OCR) that creates data dependence to previous steps, which forces sequentialization of the processing and affects the overall throughput of the compute architecture; second, the LSTM algorithm includes multiple non-linear activation

✉ Vladimir Rybalkin
rybalkin@eit.uni-kl.de

Extended author information available on the last page of the article.

functions in contrast to low-complexity rectified linear units common for CNN layers; third, LSTM networks have a large memory footprint, and a peak performance that is not computation bounded but limited by the external memory bandwidth [26]. The challenges are further augmented for Multidimensional Long Short-Term Memory (MD-LSTM) that has recurrent connections along multiple axes, e.g. two in image processing or three in volumetric imaging. Multidimensional recurrences further enlarge the memory footprint and required memory bandwidth, significantly reduce the potential for parallelization that extremely slows down the inference and training. Although, it has been shown that a size of a model based on MD-LSTM can be 3-11 times smaller than a CNN model with similar accuracy [38, 45], the best parallel implementation of a single layer of MD-LSTM has a computational complexity that is typically by one or two orders of magnitude higher than of a naive implementation of a convolutional layer [45]. These aforementioned reasons challenge the design of efficient hardware accelerators for LSTM networks.

In this work, we systematically address these challenges and propose efficient hardware accelerators for the inference of LSTM networks. We present two novel hardware architectures: first, an FPGA-based hardware architecture for MD-LSTM network targeting image segmentation and recognition problems. Second, a DRAM-based Processing-in-Memory (PIM) architecture for 1D-LSTM network targeting energy efficient compute platforms. To the best of our knowledge, our research is the first to propose a DRAM-based Processing-in-Memory (PIM) architecture for 1D-LSTM network and the first ever to propose a hardware architecture for MD-LSTM.

Excessive sequential execution of MD-LSTM is a bottleneck that significantly constrains efficiency of using massive parallelism of modern GPUs that prevents intensive research involving MD-LSTM in recent years. In this paper, we show that high flexibility of FPGAs that allows for custom pipelined datapath, and natural support of low bit-width operations provides an efficient solution to complex recurrent dataflow of MD-LSTM, when massive parallelism of GPUs ain't enough to provide the necessary performance required by the application. An efficient implementation for inference is presented that opens a door for efficient training that can boost application of MD-LSTM. We elaborate on two vanilla MD-LSTM [19] based typologies, where one was trained to perform binarization of historical documents from DIBCO 2017 competition [44], and second to perform handwritten digits recognition task on MNIST dataset [30]. The former task was selected because it is a challenging problem that shows that the selected topology has a representative complexity and deals with a real world problem. Additionally, MNIST was chosen as a benchmark to have a fair comparison with existing

FPGA-based implementations with respect to throughput and energy efficiency.

The PIM architecture focuses on integrating a binary weighted 1D-LSTM inference engine into a commodity DRAM architecture. The aim of the PIM architecture is to eliminate the memory bound (bandwidth and energy) issues that are associated with NN data and parameter access. It bridges the memory-computation gap by integrating the computation logic in the memory devices. The tight coupling of the memory and computational logic enables massive data parallelism and minimal data movement cost, which results in a much higher energy efficiency. The challenge in DRAM-based PIM architecture is the physical implementation of the computational logic that is constrained by factors like DRAM sub-array width, fine bit-line pitch and the limited number of available metal layers (e.g. three) in the DRAM process.

The novel contributions of this paper are:

- A novel hardware architecture of MD-LSTM neural network is presented.
- A systematic exploration of accuracy vs. hardware cost trade-off for MD-LSTM using challenging DIBCO 2017 and MNIST datasets is conducted. We investigate the effects of quantization of weights, input, output, recurrent, and in-memory cell activations on accuracy during training. In particular, we explore binarized weights and activation functions.
- We provide the first open source HLS implementation for parameterizable hardware architectures of MD-LSTM layers on FPGAs, which provides full-range precision flexibility, and allows for parameterizable performance scaling.¹
- An open-source implementation of MD-LSTM in Pytorch (<http://pytorch.org/>) for training and inference on GPU is provided.²
- We present a novel DRAM based 1D-LSTM-PIM architecture that is compatible with the existing commodity DRAM technology, strictly not modifying the DRAM sub-array design.

The paper is structured as follows: In Section 2 we review the basics of 1D- LSTM and MD-LSTM algorithms. This is followed by Section 3, which presents the details of the MD-LSTM accelerator. Section 3.1 gives an overview of the existing research on MD-LSTM. We describe the training setup and quantization techniques in Section 3.2 and Section 3.3, respectively. Section 3.4 provides the details of the novel hardware architecture of MD-LSTM. Section 3.5 presents the results with respect to implementation of MD-LSTM. Section 4 presents the details of the 1D-LSTM

¹<https://github.com/tukl-msd/hls-2dlstm>

²<https://github.com/tukl-msd/pytorch-2dlstm>

PIM accelerator. The prior-art on PIM accelerators is reviewed in Section 4.1 and followed by the basics of DRAM in Section 4.2. Section 4.3 presents DRAM based 1D-LSTM PIM architecture in detail. The results of the PIM implementation are discussed in Section 4.5. Finally, Section 5 concludes the paper.

2 Background

2.1 1D-LSTM

In this paper, as a showcase topology for the PIM architecture we selected a Bidirectional LSTM (Bi-LSTM) with Connectionist Temporal Classification (CTC) layer targeting OCR. Bidirectional Recurrent Neural Networks (RNNs) were proposed to take into account impact from both the past and the future status of the input signal by presenting the input signal forward and backward to two separate hidden layers both of which are connected to a common output layer that improves overall accuracy [20]. CTC is the output layer used with LSTM networks designed for sequence labelling tasks. Using the CTC layer, LSTM networks can perform the transcription task on a level of text-lines without requiring pre-segmentation into characters.

For the sake of clarity, we review the basic LSTM algorithm, earlier presented in [18, 22].

$$\begin{aligned}
 a^t &= \tanh(W_a x^t + R_a y^{t-1} + b_a) \\
 i^t &= \sigma(W_i x^t + R_i y^{t-1} + b_i) \\
 f^t &= \sigma(W_f x^t + R_f y^{t-1} + b_f) \\
 o^t &= \sigma(W_o x^t + R_o y^{t-1} + b_o) \\
 c^t &= i^t \odot a^t + f^t \odot c^{t-1} \\
 y^t &= o^t \odot \tanh(c^t)
 \end{aligned} \tag{1}$$

The LSTM architecture is composed of a number of recurrently connected memory cells. Each memory cell is composed of three multiplicative gating connections, namely input, forget, and output gates (i , f , and o , in Eq. 1); the function of each gate can be interpreted as write, reset, and read operations, with respect to the cell internal state (c). The gate units in a memory cell facilitate the preservation and access of the cell internal state over long periods of time. The implemented LSTM cells have been implemented without peephole connections that are optional and according to some research even redundant [6]. There are recurrent connections from the cell output (y) to the cell input (a) and the three gates. Equation 1 summarizes formulas for LSTM network forward pass. Rectangular input weight matrices are shown by W , and square weight

matrices by R . x is the input vector, b refers to bias vectors, and t denotes the time (and so $t-1$ refers to the previous timestep). Activation functions are point-wise non-linear functions, that is *logistic sigmoid* ($\frac{1}{1+e^{-x}}$) for the gates (σ) and *hyperbolic tangent* for input to and output from the node. Point-wise vector multiplication is shown by \odot .

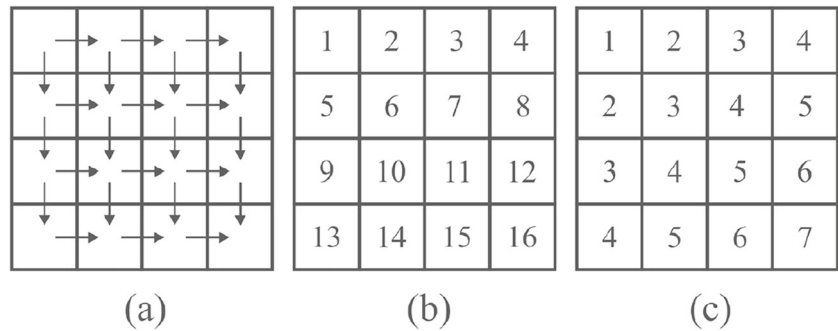
2.2 MD-LSTM

A MD-LSTM neural network was firstly proposed in [19] to extend the applicability of unidimensional counterpart for data with more than one spatio-temporal dimension like in image and video processing or volumetric medical imaging. Having recurrent connections along multiple axes, typically the x-axis and y-axis in image processing, allows MD-LSTM to capture long-term dependencies along all dimensions of the data that makes MD-LSTM very robust to local distortions along any combination of input dimensions. MD-LSTM provides state-of-the-art performance on most of benchmarks for offline Handwritten Text Recognition [38, 54]. They show advanced results in semantic segmentation [7] and medical imaging [14, 51]. Probably, MD-LSTM is one of the most controversial neural networks. While having high theoretical potential for learning, [19, 38, 45, 55] acknowledged that MD-LSTMs are in principle more powerful and more robust to distortions than CNNs, they suffer from lack of investigation due to slow training. Recently, there were a lot of discussions if MD-LSTM layers are needed at all. It was shown that in some scenarios, MD-LSTM layers can be replaced with CNN layers that can provide similar or higher accuracy at lower computational cost [45]. However, most recent research on handwritten text recognition [38] proves that MD-LSTM interleaved with CNN layers and 1D-LSTM on the top provides a superior performance than pure CNN based topology or CNNs combined with 1D-LSTM in the case of more complex and challenging real-world datasets. That justifies research related to MD-LSTM.

As already said, MD-LSTM is a generalization of a recurrent neural network, which can deal with higher-dimensional data. In the following we restrict ourselves to the two-dimensional case, which is commonly used for handwriting and similar tasks. A 2D Long Short-Term Memory (2D-LSTM) scans the input image along horizontal and vertical axes and produces an output image of the same size following equations (Eq. 2) for each coordinate (i, j) in the input image x .

It is common to use parallel 2D-LSTM layers processing an input image in four directions (top-left origin, top-right origin, bottom-left origin and bottom-right origin) as it is depicted in Fig. 1 for top-left origin. The outputs from four directions are later combined (concatenated) so that a

Figure 1 In the case of top-left origin: **a** the dependencies between each position (pixel), **b** the pixel-by-pixel processing order, **c** the diagonal-wise processing order.



spatio-temporal context from all four directions is available at each position (pixel).

For the sake of clarity, we review the basic 2D-LSTM algorithm. Equation 2 summarizes formulas for the network forward pass. Similar to unidimensional LSTM, 2D-LSTM architecture is composed of a number of recurrently connected memory cells. However, each memory cell is composed of four, rather than tree multiplicative gating connections, namely input, forget for the y-coordinate and x-coordinate, and output gates (k , f , g , and o). There are two square weight matrices, one for the y-coordinate (U) and one for x-coordinate (V). The rest is similar to the unidimensional case.

3 MD-LSTM

$$\begin{aligned}
 a_{i,j} &= \tanh(W_a x_{i,j} + U_a y_{i-1,j} + V_a y_{i,j-1} + b_a) \\
 k_{i,j} &= \sigma(W_k x_{i,j} + U_k y_{i-1,j} + V_k y_{i,j-1} + b_k) \\
 f_{i,j} &= \sigma(W_f x_{i,j} + U_f y_{i-1,j} + V_f y_{i,j-1} + b_f) \\
 g_{i,j} &= \sigma(W_g x_{i,j} + U_g y_{i-1,j} + V_g y_{i,j-1} + b_g) \\
 o_{i,j} &= \sigma(W_o x_{i,j} + U_o y_{i-1,j} + V_o y_{i,j-1} + b_o) \\
 c_{i,j} &= f_{i,j} \odot c_{i-1,j} + g_{i,j} \odot c_{i,j-1} + a_{i,j} \odot k_{i,j} \\
 y_{i,j} &= o_{i,j} \odot \tanh(c_{i,j})
 \end{aligned}
 \tag{2}$$

This section presents a complete design methodology that starts from selecting a neural topology. Later training, refinement and selection of a full-precision model is done. Further, we train a model with quantized learning parameters to identify a search space of accuracy as a function of hardware cost for various precision points. In the following, a selection of a configuration with an optimal trade-off between accuracy and hardware cost using Pareto frontier is performed. As the main goal of the paper, we present a novel hardware architecture and provide implementation details. We perform a throughput scalability analysis to identify a configuration with the highest throughput that is used for comparison to the state-of-the-art with respect to accuracy, achieved throughput and energy efficiency.

3.1 Related work

In this section, we review related work on MD-LSTM with respect to its application, and earlier presented optimisation techniques for accelerating both inference and training. As there are no prior publications on hardware architectures for MD-LSTM, we review state-of-the-art architectures for recurrent neural networks, namely unidirectional LSTM.

3.1.1 Application

Afzal et al. [2] has proposed treating the image binarization as a sequence learning problem and modeled the image as a two-dimensional sequence. MD-LSTM was used to learn spatial dependencies in the two-dimensional image, the same as 1D-LSTM is used in temporal domain to learn temporal dependencies. In general the task was formulated as a semantic segmentation problem, where each pixel was classified either as text or non-text. The approach was evaluated on DIBCO 2013 dataset achieving comparable results with the state-of-the-art. Similar idea was applied by one of the participants at DIBCO 2017 [44] competition using Grid LSTM cells [27]. We use similar topology to Afzal et al. but with more memory cells to cope with increased complexity of DIBCO 2017 dataset. Graves et al. [19] could achieve accuracy comparable to state-of-the-art applying MD-LSTM on MNIST dataset without using any pre-processing or augmentation. They formulated the task as a semantic segmentation problem, where each pixel was classified according to the digit it belonged to. Our approach is different as we do not make prediction for each pixel but for the complete image based on all outputs from hidden layers taken together. We selected an alternative topology in order to explore more on architectural level, otherwise the architecture would be identical to the semantic segmentation one.

3.1.2 Optimization

There is a number of papers proposing to accelerate inference and training of MD-LSTM by new parallelization approaches or improving stability of the cells for faster convergence. Leifert

et al. [31] has shown that MD-LSTM has a severe conceptual problem related to computational instability of a value of the memory state that grows drastically over time and can even become infinite. They proposed an improved version of a cell called *Leaky-LP* that overcomes the instability, while preserves the state over long period of time, however at the expense of more complex computational graph. In our research we use vanilla MD-LSTM cells, so to be able to evaluate all optimizations with respect to the original model.

MD-LSTM has constraints on computing order of the steps (pixels) due to recurrent dependencies between the pixels, see Fig. 1a. Each pixel is considered as one time step/position here. Each position requires an input from two neighboring ancestor positions, one located on the previous column, another on the previous row. The straightforward implementation follows a pixel-by-pixel processing order, one pixel is processed after another as it is shown in Fig. 1b. Voigtlaender et al. [54] and Oord et al. [41] proposed conceptually similar parallelization approaches. They have noticed that all pixels on the same diagonal can be computed in parallel because they are computationally independent from each other, the processing order is depicted in Fig. 1c. The diagonal-wise approach reduces the time complexity to process an image from $\mathcal{O}(W \cdot H)$ to $\mathcal{O}(W + H)$ because there are $(W + H - 1)$ diagonals in an image with H rows and W columns. Our architecture is based on the pixel-by-pixel order of processing. In the case of GPU, we have implemented both approaches. Additionally, Wenniger et al. [55] proposed an efficient approach of packing samples in batches with low computational and memory overhead. The approach could achieve 6.6 times speed-up on word-based handwriting recognition by avoiding wasteful computation on padding. In the case of MNIST dataset, the padding is not necessary because all images are of the same size. For DIBCO dataset, the padding is negligible because we process big size images (up to 4M pixels) in small chunks (64 by 64 pixels). Our approach is orthogonal to all previously proposed, as we present a hardware acceleration on alternative computing platform, namely FPGA. Our approach potentially can be combined with previously presented ones.

3.1.3 Architecture

To the best of our knowledge, we are proposing the first hardware architecture of MD-LSTM. However, it is not the first architecture of RNNs. In [48] Rybalkin et al. gives a good overview of FPGA-based implementations. The most remarkable architectures providing the highest throughput are [48] and [21]. Where the later one operates on partially stored on-chip sparse model. However, in this paper we investigate influence of low precision on accuracy rather

than sparsity, and keep our model dense. Furthermore, as it has been mentioned in Section 1, the MD-LSTM models are much more compact in comparison to other types of neural networks [38, 45] that makes them very appealing for on-chip implementation. In the following, we primary compare new architecture with [48] as it operates on a dense model stored completely on-chip. The new architecture has additional datapaths for recurrent dependencies along extra dimensions. We exploit a concept of interleaving data corresponding to different processing directions proposed in [49], and extend it to four processing directions (see Section 2). Interleaving in the case of RNNs enables to keep pipelined datapath always busy (see Section 3.4). We show that in the case of MNIST dataset, it is not necessary to align corresponding outputs from four processing directions that significantly reduces the required on-chip memory for intermediate computations. However, in the case of semantic segmentation problem, like in binarization, the alignment is necessary. We propose an architecture of an output layer operating on multidimensional buffer.

3.2 Training

In the following chapter, we give an overview of the selected applications and benchmarking datasets, namely MNIST and DIBCO. A complete training flow is presented in this section.

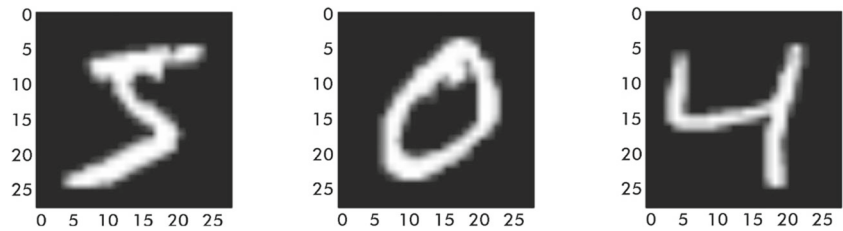
During training we experimented with a different number of MD-LSTM cells, precision and different hyper parameters. The quantized versions of the networks were trained de novo without full-precision pre-training. We trained the networks with quantized weights, biases, input, output, recurrent activations, and full-precision in-cell activations. A quantized LUT-based version of the in-cell activations was used during the test. It was found that gradient clipping can be necessary to achieve stable learning that is reasonable in the case of vanilla cells in particular [55]. We applied dropout between the 2D-LSTM layers and output layer for better generalization.

3.2.1 MNIST

We used a well-known MNIST dataset [30] of isolated handwritten digits to have a common benchmark to compare our hardware implementation to existing FPGA-based designs with respect to throughput and energy efficiency. The database consists of gray-scale images, each of which is 28 pixels high and 28 pixels wide depicting a single handwritten digit like in Fig. 2.

MNIST has a training set with 60k images and a test set with 10k. We used 5% of the training set for validation. We report the best test accuracy of a model with the highest

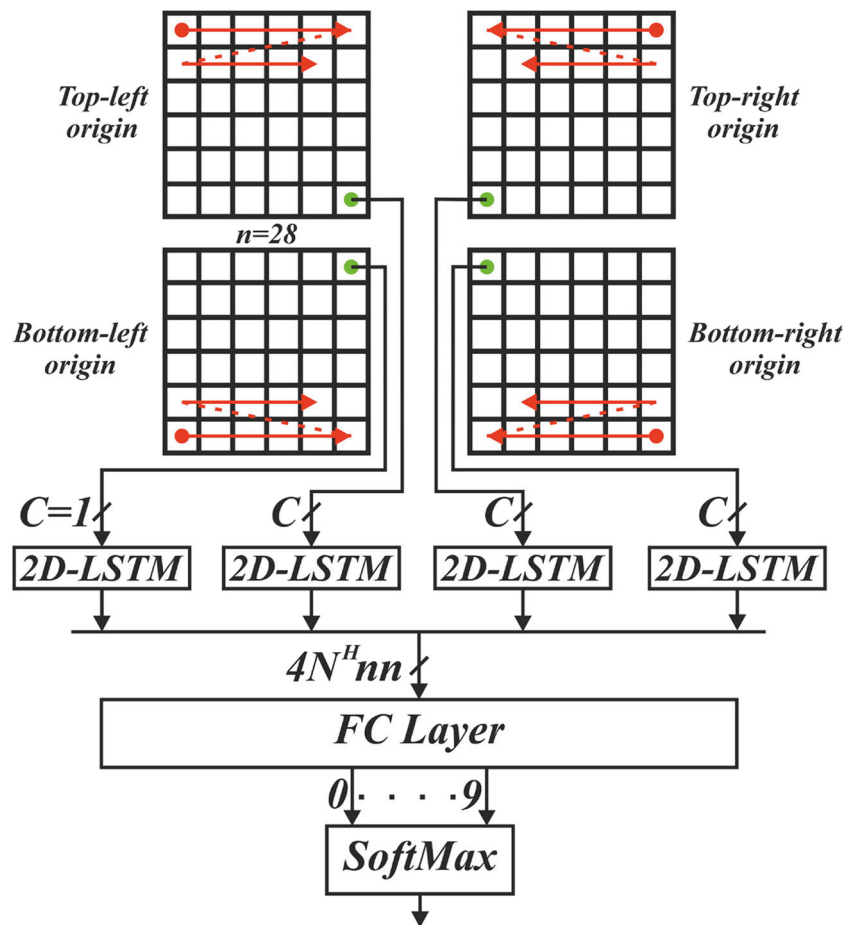
Figure 2 Sample images from the MNIST dataset.



validation accuracy after at most 1000 epochs. We did not perform any pre-processing or augmentation on MNIST dataset. We used a threshold-based binarization on the input pixels in the case of quantized network and real-value pixels in the case of full-precision configuration. The architecture of 2D-LSTM network depicted in Fig. 3 has a single hidden layer that is separated into four independent LSTMs processing the input in four directions with N^H memory cells each. The red arrows in Fig. 3 indicate the direction of propagation during the forward pass. The

input layer is of size $(C) 1$. The outputs from all hidden layers and from all pixels are concatenated and fed into the common output layer of size $(N^O) 10$ that is a number of classes (digits from 0 to 9). The LogSoftmax activation function was used at the output layer. We used a negative log likelihood loss criterion with Adam optimizer and a mini-batch size of 512 or 256 depending on a size of a model. We used larger batch size for training smaller models to fully utilize the GPU resources. The learning rate starts from 0.01 and reduced by a factor of 0.8 every

Figure 3 Image classification topology for MNIST dataset. The forward pass.



20 epochs. We used gradient clipping with a value of 100.

3.2.2 DIBCO 2017

Document image binarization is a process of separating text from images, where text consists of pixels of interest, thus we train a model to label every pixel as belonging to text or background. Binarization of document images is a common pre-processing step for the document image processing that enhances quality of the images. Document images, especially of historical documents, usually contain different types of degradation and noise, i.e. different artifacts, uneven background, and bleedthrough text like it is shown in Fig. 4.

We considered a dataset that is used in DIBCO 2017 document image binarization contest to proof representability and correctness of our implementation. For training we used a combination of datasets from DIBCO 2009 - 2016 competitions that gave us 86 images, where 10 of them were randomly chosen for validation. We report the highest test accuracy associated with the highest validation accuracy after at most 300 epochs. For testing we used a dataset from DIBCO 2017 [44] as the most challenging among all of them. The test datasets contains 10 printed and 10 handwritten documents. All datasets contain both color and gray-scale images with up to 4M pixels per image. We treated all images as 3-channel color images ($C = 3$). We did not perform any pre-processing or augmentation. The images are divided into patches of size n by n and are fed into the 2D-LSTM. We could achieve the best results with $n = 64$. The architecture of 2D-LSTM network depicted in Fig. 5 has a single hidden layer that is separated into four independent LSTMs processing the patches from all the four possible directions. Each LSTM consists of N^H memory cells. The values from each LSTM are concatenated with respect to corresponding pixels, and fed into a common output layer of size $(N^O) 2$. Now, the output has a context from four different directions of the image at each pixel. Each

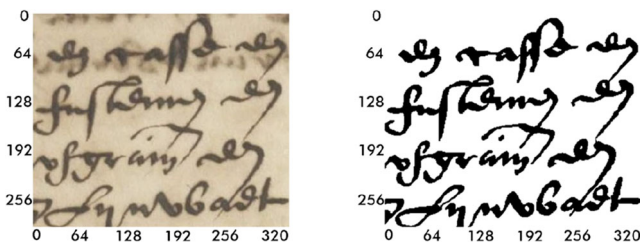


Figure 4 Sample image with bleedthrough text between the lines and corresponding ground truth from the DIBCO 2017 dataset.

output value indicates the probability of each pixel to be either text or background. We used a negative log likelihood loss criterion with Adam optimizer with a mini-batch size of 128 or 96 patches depending on a size of a model. We used larger batch size for training smaller models to fully utilize the GPU resources. The learning rate starts from 0.01 and reduced by a factor of 0.7 every 10 epochs. We used gradient clip with a value of 10.

3.3 Quantization

In this section, we present the approaches that have been used for quantizing learning parameters and activations of the networks during training.

Our quantization methods are based on those presented in [24, 46, 58]. For multi-bit weights and activations, we adopt Eq. 3a as a quantization function, where \mathbf{a} is a full-precision activation or weight, \mathbf{a}_q is the quantized version, k is a chosen bit-width, and f is a number of fraction bits. For multi-bit weights, the choice of parameters is shown in Eq. 3b. For multi-bit activations, we assume that they have passed through a bounded activation function before the quantization step, which ensures that the input to the quantization function is either $x \in [0, 1)$ after passing through a *logistic sigmoid* function, or $x \in [-1, 1)$ after passing through a *tanh* function. As for the quantization parameters, we use Eq. 3b after the *tanh* and Eq. 3c after the *logistic sigmoid*.

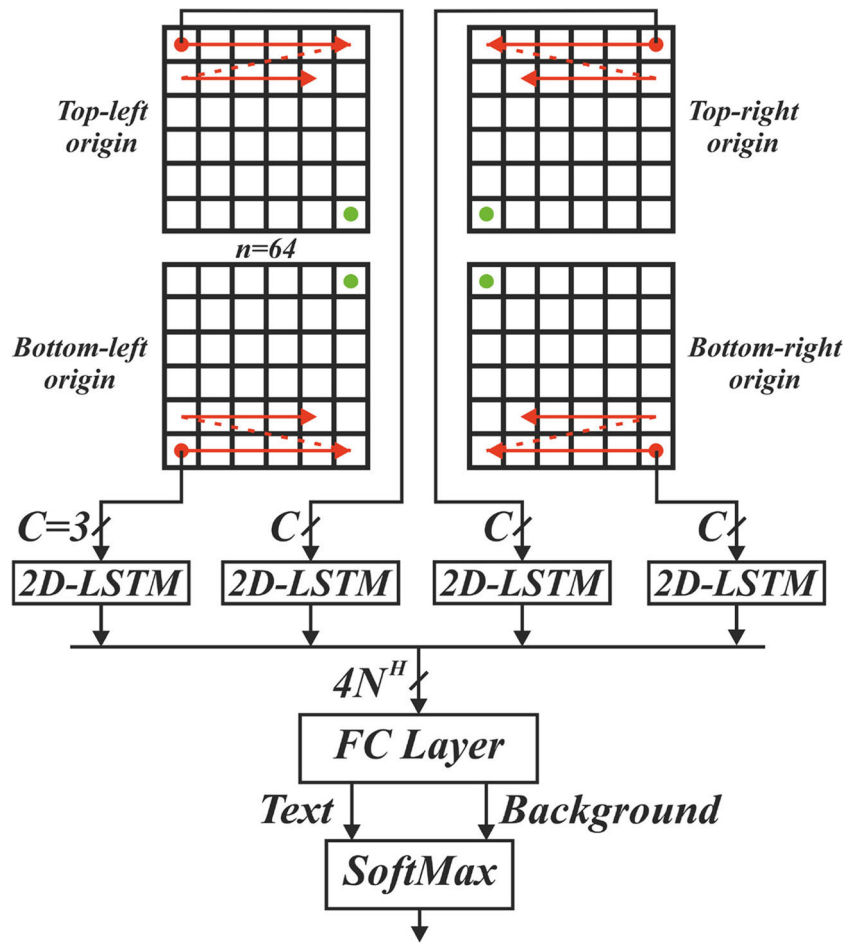
$$\mathbf{a}_q = clip(round(\mathbf{a} \cdot 2^f) \cdot 2^{-f}, \min, \max), \text{ where} \quad (3a)$$

$$\begin{aligned} f &= k - 1 \\ \min &= -(2^{(k-f-1)}) = -1 \\ \max &= -\min - 2^{-f} = 1 - 2^{-f} \end{aligned} \quad (3b)$$

$$\begin{aligned} f &= k \\ \min &= 0 \\ \max &= 2^{k-f} - 2^{-f} = 1 - 2^{-f} \end{aligned} \quad (3c)$$

In the case of binarized activations \mathbf{a}_b , we apply a *sign* function as the quantization function [46], as shown in Eq. 4. In the case of binarized weights \mathbf{w}_b , on top of the sign function we scale the result by a constant factor that is a standard deviation calculated according to Glorot et al. [17]. Our motivation to use this scaling coefficient based on that we use Glorot initialization for the weights in the case of full-precision model. In the case of binarization, the weights

Figure 5 Document image binarization topology for DIBCO 2017 dataset. The forward pass.



are constrained to -1 and 1, thus it makes sense to scale them to the boundary limits of the full-precision weights.

$$a_b = \text{sign}(a) \tag{4}$$

$$w_b = \text{sign}(a) \cdot \text{scaling_factor} \tag{5}$$

3.4 Architecture

This section presents a hardware architecture of 2D-LSTM and fully-connected layers. The main challenge to design an efficient architecture of an algorithm with feed-back loop comes from recurrency that becomes a throughput bottleneck. A next time step (pixel) can be calculated only if the previous step has been finished due to precedence constraint that makes pipeline part-time idle. An efficient solution is to make pipeline busy with calculations that do not have recurrent dependencies between each other. In the case of 2D-LSTM, the processing of the different directions (top-left origin, top-right origin, bottom-left origin and

bottom-right origin) are independent. Being inspired with the idea presented in [49], we propose to interleave calculations corresponding to different directions. First, the architecture processes first pixel from each direction in a sequence, then second pixel, and so on. This approach keeps pipeline always busy without throughput penalties, if the number of sequentially processed LSTM cells is enough.

The architecture is parametrizable with respect to performance scaling. The parametrization allows to apply coarse-grained parallelization on a level of 2D-LSTM cells, and fine-grained parallelization on a level of dot-products. The former, indicated as PE_LSTM unrolling, while the latter, indicated as SIMD_INPUT and SIMD_RECURRENT unrolling factors for input data and recurrent paths, respectively. This flexibility allows for tailoring of parallelism according to the available resources on the target device. When PE unrolling is performed, the LSTM cell will be duplicated PE_LSTM times, generating for each clock cycle PE_LSTM output activations. When SIMD unrolling is applied, each gate will receive a portion of the input data, namely SIMD_INPUT and a portion of the recurrent

state `SIMD_RECURRENT` in each clock cycle. The same is applied to the fully-connected layer with `PE_FC` and `SIMD_FC` unrolling factors.

In the following, we explain the functionality of each component of the 2D-LSTM accelerator depicted in Fig. 6.

- *Mem2Stream* core is responsible for DMA reads of input data from DRAM using AXI memory-mapped non-coherent interface, and converting each transaction into AXI stream. The input data (pixels) are concatenated into 64- or 128-bit words that correspond to a single bus transaction depending on a bus width. For simplicity, the pixels are quantized so that the bus width is a multiple of pixel width, so the pixels do not spread over several transactions, and no padding is required.
- The *Data Width Converter* splits each transaction into separate pixels and concatenates them according to `SIMD_INPUT`.
- The *2D-LSTM Hidden Layer* core depicted in Fig. 7 implements all functionality according to Eq. 2 except the recurrent connections. All memory cells and corresponding dot-products are instantiated with respect to `PE_LSTM`, and `SIMD_INPUT`, `SIMD_RECURRENT` respectively. Computations corresponding to different memory cells and dot-products can be calculated with various degree of parallelization, but always in a sequence with respect to different directions. Sequential execution of computations corresponding to different directions instead of parallel execution allows to save hardware resources as we do not quadruple datapath and memory ports. Although it increases latency per image, it keeps pipeline always busy that results in a higher computational efficiency.
- The *Recurrent Path X/Y-axis Data With Converter* converts the output from the hidden layer that has a width that is a multiple of `PE_LSTM` into input with a width that is a multiple of `SIMD_RECURRENT`. The buffers

are required to match the inputs and outputs from the corresponding directions. The *X-axis buffer* contains $4 \times N^H$ recurrent values $(y_{i-1,j})$ along x-axis, while the *Y-axis buffer* contains $W \times 4 \times N^H$ recurrent values $(y_{i,j-1})$ along y-axis.

- The *Output Layer* implements the output fully-connected layer with `PE_FC = FULL` and `SIMD_FC` equal to `PE_LSTM` of the previous hidden layer, so to match the throughput. The architecture of the core is depicted in Fig. 8. In the case of MNIST, the outputs from different directions do not need to be matched for the same pixels, thus there is no need for a large matching buffer. The results are accumulated in a single-value buffer instantiated for each output unit. Opposite for DIBCO, outputs from hidden layer corresponding to the same pixel processed from different directions have to be matched, thus there is a matching buffer of size $(H \times W)$ for each output unit shown in Fig. 8. The buffers are implemented with dual-port read-write memories that allow to read and write partial sums in a pipelined manner at each clock cycle. Due to pixel-by-pixel processing order and sequential computations corresponding to different directions, the read and write patterns never risk to address the same address (partial sum) in the buffer in the same clock cycle. The processing directions follow distinct access sequence that is encoded in *Row offset* and *Column index* buffers.
- The *SoftMax* core implements a simplified version of Softmax function. It finds a label with the maximum value among all outputs from the output layer in the case of MNIST, or among all outputs corresponding to the same pixel in the case of DIBCO.
- The *Data Width Converter* concatenates the labels into 64- or 128-bit words depending on bus width.
- The *Stream2Mem* writes the labels to DRAM using AXI memory-mapped non-coherent interface.

Figure 6 System-level view of the 2D-LSTM Accelerator.

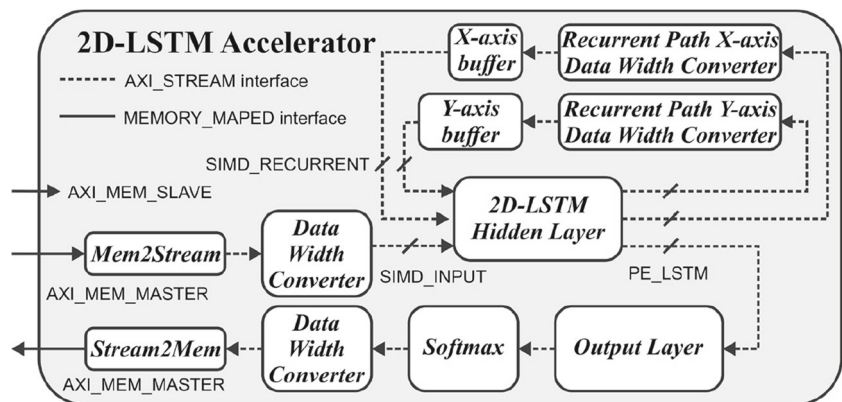


Figure 7 Hardware Architecture of the Hidden Layer.

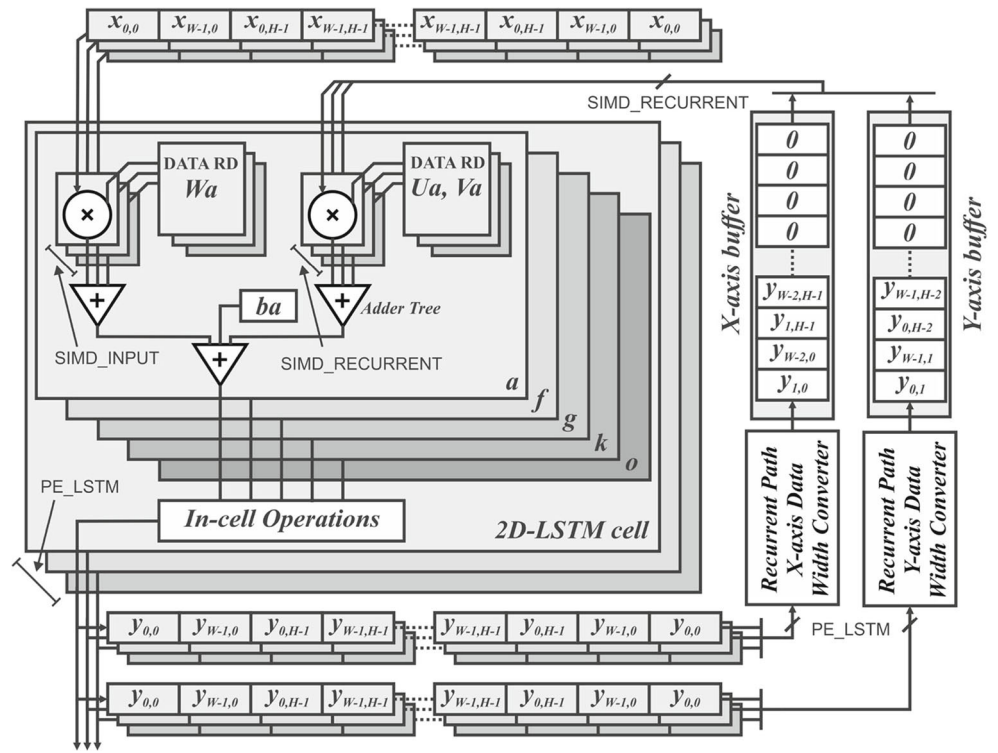
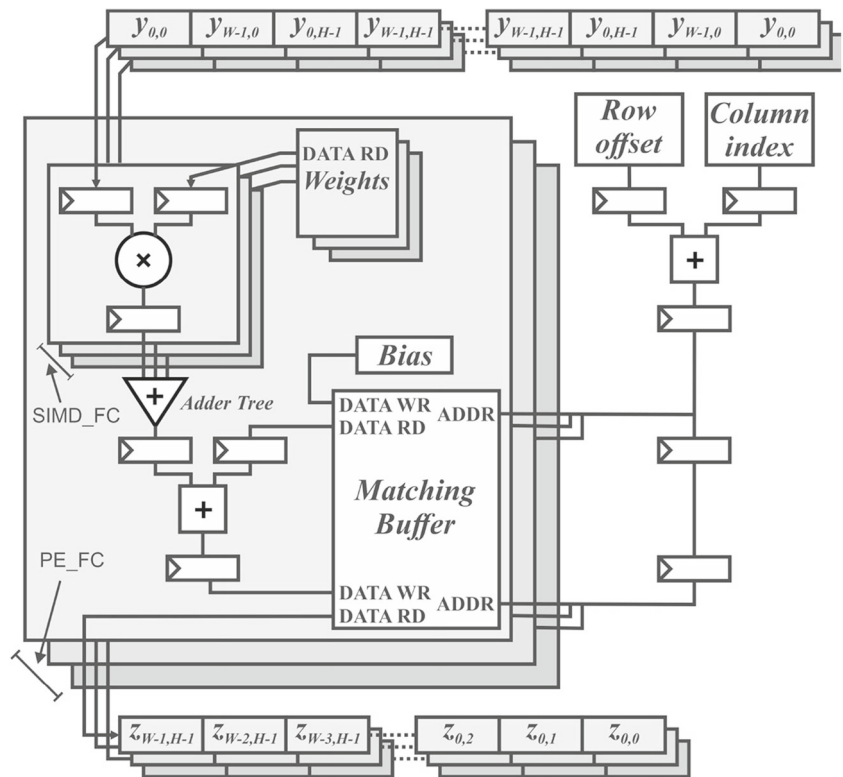


Figure 8 Hardware Architecture of the Output Layer.



3.5 Results

3.5.1 Full-Precision Accuracy

In the following, we are presenting accuracy achieved with a full-precision 2D-LSTM network on MNIST and DIBCO 2017 datasets.

3.5.2 MNIST

Table 1 presents classification accuracy achieved with the full-precision 2D-LSTM on MNIST dataset. Our full-precision version is approaching accuracy of the state-of-the-art network [29] without using any pre-processing or augmentation, and confidently outperforms the result of a similar approach presented in the original paper on MD-LSTM [19] (see Section 3.1). The network presented in [29] is based on three deep learning architectures working in parallel, namely fully-connected deep network, deep CNN and deep RNN. The top result was achieved using a combination of 30 deep learning models that by far exceeds the size and complexity of our network. The 2D-LSTM architecture with 20 memory cells shows accuracy comparable with the state-of-the-art. This configuration has been selected for a trade-off analysis of accuracy and hardware cost for various precisions.

3.5.3 DIBCO

Table 1 presents accuracy (F-measure) achieved with the full-precision 2D-LSTM on DIBCO 2017 dataset in comparison with previous methods. Our implementation with the highest accuracy outperforms the similar approach (see Section 3.1) and loses only 2.8% in accuracy to the winner of the competition presented in [44] without using any pre-processing or augmentation of the dataset.

The winner approach presented in [44] is based on U-Net [47] type of fully convolutional neural network and data augmentation during training. The 2D-LSTM architecture with 40 memory cells shows accuracy comparable with the state-of-the-art. In the following experiments, this configuration has been used for a trade-off analysis of accuracy and hardware cost for various precisions.

3.5.4 Accuracy vs. Hardware Cost vs. Precision

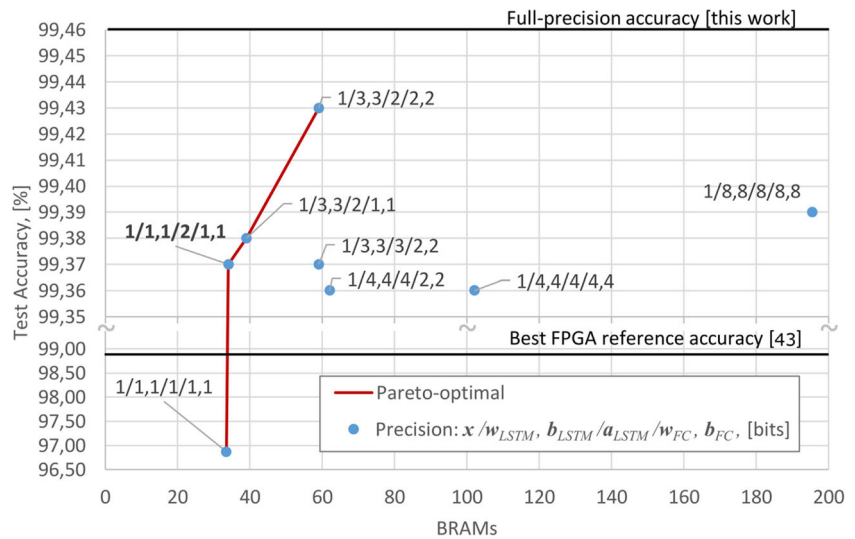
In the following experiments we explore accuracy as a function of hardware cost for various precision. All hardware prototypes have been implemented on Xilinx Zynq UltraScale+ MPSoC ZCU102 board featuring a XCZU9EG device that contains (1) Quad-core ARM Cortex-A53 and (2) FPGA fabric. The HLS synthesis and place and route have been performed with Vivado Design Suite 2017.4.

Figures 9, 10, 11, 12 plot the achieved test accuracy over BRAM36k and LUT utilization for different precisions for MNIST and DIBCO 2017 datasets. All resources are given after post-placement and routing. Each point on the plots represents a configuration represented with a combination of number of bits used for inputs (x), weights (w_{LSTM}), biases (b_{LSTM}), output activation (y_{LSTM}) of the 2D-LSTM layer and weights (w_{FC}) and biases (b_{FC}) of the FC layer. In the experiments, the parallelism of the hardware accelerator has been kept constant, with a number of instances of the complete accelerator $M=1$, $PE_{LSTM}=1$, $SIMD_INPUT=FULL$, $SIMD_RECURRENT=FULL$, $PE_{FC}=FULL$ and $SIMD_FC=1$ with a target frequency that has been set to 300 MHz and 240 MHz for accelerators running MNIST and DIBCO 2017 datasets, respectively. In order to identify a configuration with the lowest hardware cost and accuracy that satisfies chosen criteria, we find Pareto frontier for various precisions in a space of accuracy

Table 1 Accuracy of a full-precision 2D-LSTM on MNIST and DIBCO 2017 datasets depending on a number of memory cells (N^H) in comparison to earlier approaches.

	Approach	N^H	Train Accuracy, [%]	Test Accuracy, [%]
MNIST	State-of-the-art [29]	-	-	99.8200
	Similar approach [19]	-	-	98.9000
	2D-LSTM [this work]	40	100	99.5700
	2D-LSTM [this work]	20	99.9983	99.4600
DIBCO	A winner of DIBCO 2017 [44]	-	-	91.0400
	Similar approach [44]	-	-	82.4400
	2D-LSTM [this work]	60	94.1131	88.2360
	2D-LSTM [this work]	40	93.7852	88.0040

Figure 9 Pareto frontier for accuracy against BRAMs for 2D-LSTM with $N^H = 20$ memory cells running on MNIST dataset depending on precision of input activation (x), weights (w_{LSTM}), biases (b_{LSTM}), output activation (y_{LSTM}) of the 2D-LSTM layer, and precision of weights (w_{FC}) and biases (b_{FC}) of the FC layer.



as a function of hardware cost. The Pareto frontier identifies the configurations with the best trade-off between accuracy and hardware cost.

3.5.5 MNIST

The full-precision configurations presented in Table 1 have achieved higher test accuracy with a higher value of dropout than quantized versions (0.8 vs. 0.7, respectively) that confirms observations that quantization has a regularization effect on neural network similar to dropout that prevents model overfitting [23, 56]. Furthermore, high dropout values mean that MD-LSTM has a strong tendency to overfitting that is also in accordance with earlier observations presented in [43] and [38]. Considering Figs. 9 and 10 in the following, a configuration with 2-bit weights is not depicted on the plots because it has shown bad convergence that is probably a result of imbalanced representation of negative and positive

values in the case of 2-bit quantization, namely (-1.0, -0.5, 0, 0.5). We expect that ternarisation of the model will result in higher accuracy.

Configuration 1/1,1/1/1,1 demonstrates that binarization of activation causes stronger degradation of accuracy than binarization of the weights that can be explained by that we did not use scaling factor for binarized activation but weights and biases only. Some of the configurations with lower precision have higher test classification accuracy than some with higher precision that can mean that regularization that is brought by the quantization outperforms the negative effects of the lower precision. Moderate degradation in accuracy even for the configuration with binary weights means that the 2D-LSTM is an overkill for the MNIST problem. We constrained a selection of an optimal configuration with accuracy that is higher than the best test accuracy on MNIST dataset for existing FPGA-based implementations presented in Table 3. A hardware

Figure 10 Pareto frontier for accuracy against LUTs for 2D-LSTM with $N^H = 20$ memory cells running on MNIST dataset depending on precision of input activation (x), weights (w_{LSTM}), biases (b_{LSTM}), output activation (y_{LSTM}) of the 2D-LSTM layer, and precision of weights (w_{FC}) and biases (b_{FC}) of the FC layer.

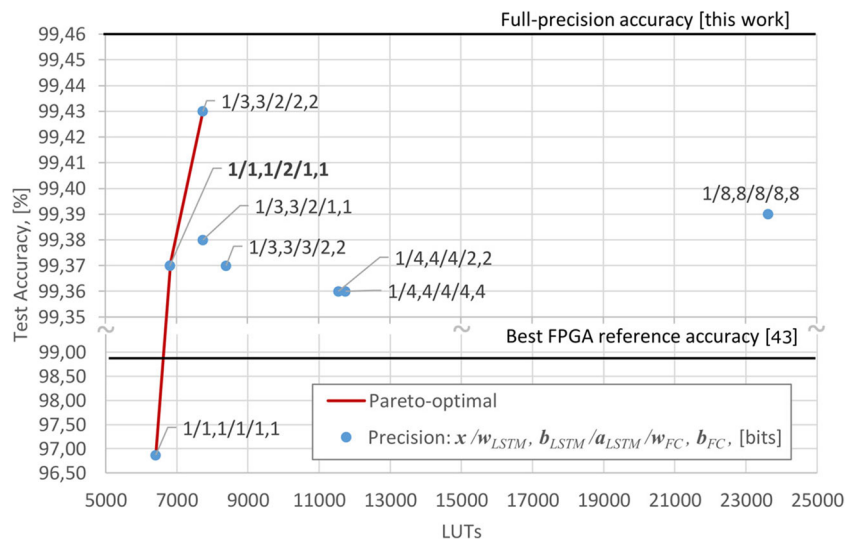
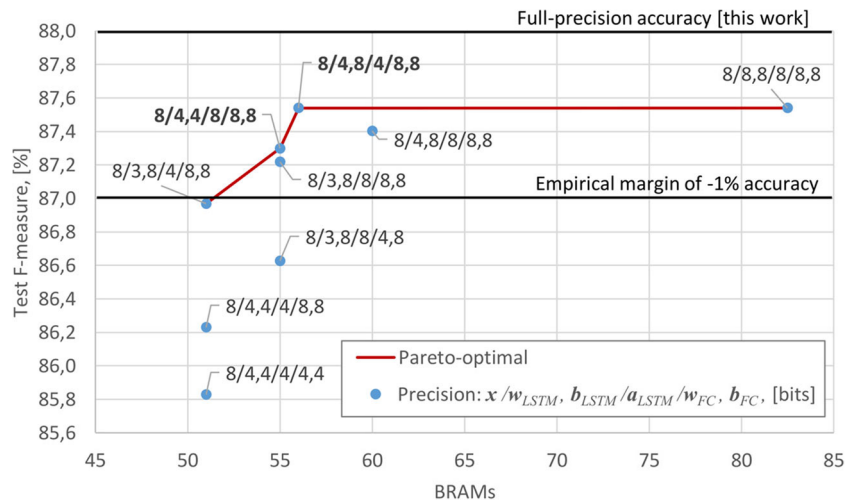


Figure 11 Pareto frontier for accuracy against BRAMs for 2D-LSTM with $N^H = 40$ memory cells running on DIBCO 2017 dataset depending on precision of input activation (x), weights (w_{LSTM}), biases (b_{LSTM}), output activation (y_{LSTM}) of the 2D-LSTM layer, and precision of weights (w_{FC}) and biases (b_{FC}) of the FC layer.



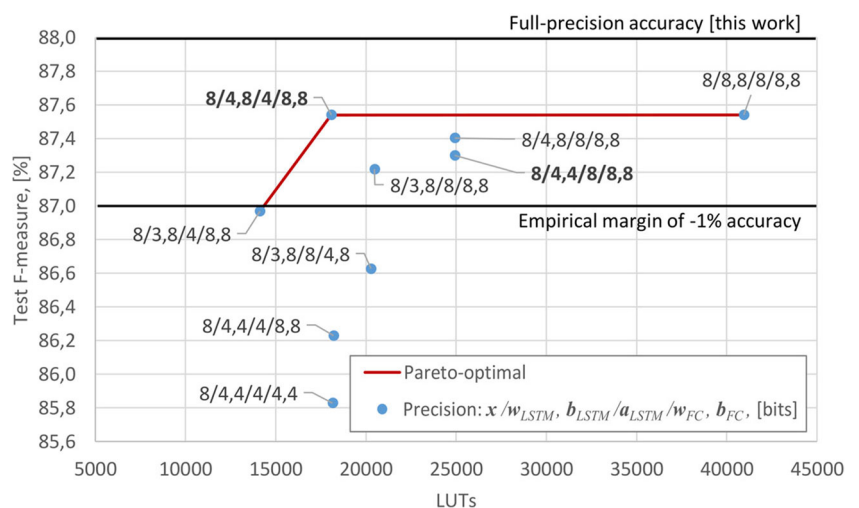
implementation of a multilayer perceptron presented in [42] with 98.92% of accuracy identifies the low bound. We selected a configuration 1/1,1/2/1,1 that is Pareto optimal with the lowest hardware cost and accuracy higher than the low bound for further throughput scalability analysis and comparison with existing FPGA implementations processing MNIST dataset.

3.5.6 DIBCO

Considering Sections 11 and 12 in the following, the results show a strong correlation between the precision of the operands and accuracy. Comparing configurations 8/4,4/4/8,8 and 8/4,8/4/8,8, we see that precision of a bias has a noticeable influence on accuracy with 1.31% gain in accuracy for the later one. Keeping higher precision for bias, while reducing precision for weights or activation can be an efficient approach for increasing accuracy at low hardware cost. The noticeable influence of biases on accuracy can be explained by higher contribution of biases to the final result

in the case of smaller models. The full-precision versions presented in Table 1 could achieve higher accuracy with higher value of dropout than quantized versions (0.6 vs. 0.5, respectively). There is still a noticeable gap between train and test accuracy that can mean an overfitting of the model, however higher dropout values did not improve the accuracy. The gap in accuracy between training and test can be also explained by a different complexity between training and test datasets as DIBCO 2017 dataset is more challenging for binarization than previous additions. We constrained a selection of an optimal configuration with degradation in accuracy of less than 1% with respect to full-precision version. 1% degradation is an empirical value for fair trade-off between complexity and accuracy. According to Fig. 11 configurations 8/4,4/8/8,8 and 8/4,8/4/8,8 are Pareto optimal with accuracy higher than the low bound. However, in the case of LUT utilization, 8/4,4/8/8,8 is not a Pareto optimal configuration that makes 8/4,8/4/8,8 a selection for further investigation. The preference is given to Pareto optimal configurations with respect to LUT resources

Figure 12 Pareto frontier for accuracy against LUTs for 2D-LSTM with $N^H = 40$ memory cells running on DIBCO 2017 dataset depending on precision of input activation (x), weights (w_{LSTM}), biases (b_{LSTM}), output activation (y_{LSTM}) of the 2D-LSTM layer, and precision of weights (w_{FC}) and biases (b_{FC}) of the FC layer.



as the LUT utilization is the implementation bottleneck, see Table 2.

3.5.7 Throughput Scalability

According to Eq. 2, a number of learnable parameters per single 2D-LSTM cell is $5 \times (C + 2 \times N^H + 1)$, where five comes from a number of gates including input to a memory cell. In the case of semantic segmentation problem, a single neuron of the output layer has $(4 \times N^H + 1)$ parameters, where $4 \times N^H$ gives a number of concatenated outputs from the hidden layers. In the case of MNIST, a single unit of the output layer has $(4 \times N^H \times H \times W + 1)$ parameters. The number of operations of the 2D-LSTM layer per image is $Op_{LSTM} = (2 \times 5 \times (C + 2 \times N^H) + 11) \times N^H \times H \times W \times 4$, where two comes from multiplication and addition counted as separate operations, eleven is a number of point-wise operations, and four is a number of possible processing directions. The same applies to the output layer: $Op_{FC} = (2 \times (4 \times N^H) + 1) \times N^O \times H \times W$. We neglect a complexity of the rest of the network.

The resource utilization for selected configurations of the accelerator for MNIST and DIBCO 2017 datasets are presented in Table 2. In the case of design implemented for DIBCO dataset, a LUT utilization is a resource bottleneck that condition a selection of a Pareto optimal configuration with higher BRAM number but lower LUT utilization. The resource utilization of a baseline configuration ($PE_LSTM = 1$, $SIMD_LSTM = FULL$, $PE_FC = FULL$, $SIMD_FC = PE_LSTM$) allows to place more instances. The architecture allows to scale throughput by increasing a number of PE_LSTM or instantiating multiple (M) instances of the complete accelerator. The following Figs. 13 and 14 show a throughput scalability depending on a number of PE_LSTM with $M = 1$. The measured throughput scales near perfectly with respect to theoretical throughput with discrepancy that increases with a higher PE_LSTM number. The discrepancy can be explained by increased throughput penalty due to reduced number of sequentially executed cells that makes the pipeline partially idle. The theoretical throughput is calculated according to the following formula: $(Op_{LSTM} + Op_{FC}) \times F \times (PE_LSTM \times M) / L$, where F is target frequency and $L = N^H \times 4 \times H \times W$ is a theoretical latency in clock cycles to process a single image without considering any architecture

related features like depth of a pipeline. In the following experiments we aim to find a configuration with the highest throughput instantiating multiple ($M > 1$) accelerators with $PE_LSTM > 1$.

Sections 13 and 14 clearly show that instantiating multiple cells rather than multiple instances of the complete accelerator is more efficient because we avoid unnecessary replication of conversion cores and weights' memory. In the case of instantiating multiple cells, although the design operates on the same model, there is still an increase in memory utilization that comes from more parallel ports required to supply higher throughput. The measured throughput shows near theoretical values with increased discrepancy over a number of PE_LSTM .

The configurations $PE_LSTM=5$, $M=14$ and $PE_LSTM=10$, $M=2$ for MNIST and DIBCO 2017 datasets, respectively, with the highest throughput are used for comparison with GPU and other FPGA implementations in Table 3. Configurations with higher PE_LSTM are not presented as they failed routing.

3.5.8 Comparison

The host code runs on Cortex-A53 running Ubuntu 16.04.6 LTS. It reads test images from the SD card, stores them in the shared DRAM, starts the accelerator execution, measures runtime, and computes accuracy against the ground truth classifications. During processing, we perform two types of power measurements: P_{board} that is a power consumption of the complete board, in the case of FPGA measured using digital multimeter Voltcraft VC-870, and P_{chip} that has been acquired using Zynq UltraScale+ MPSoC Power Advantage Tool [1] for FPGA, and $nvidia-nsi$ for GPU.

As it is shown in Table 3, 2D-LSTM model provides the highest accuracy with respect to other FPGA-based implementations, while having one of the smallest models that confirms finding in the earlier works [38, 45] that MD-LSTM layers require less trainable parameters to achieve similar accuracy to other types of neural networks. Although, it has one of the smallest models, the number of operations is the highest that is also in alignment with the previous works, and this is the primary reason why other types of networks are preferred over MD-LSTM in the recent years.

Table 2 Resource utilization for a baseline configuration ($M=1$, $PE_LSTM=1$, $SIMD_INPUT=FULL$, $SIMD_RECURRENT=FULL$, $PE_FC=FULL$ and $SIMD_FC=PE_LSTM$) of the accelerator implemented on XCZU9EG for MNIST and DIBCO 2017 datasets.

Dataset	Precision, [bits]	Frequency, [MHz]	LUT	FF	BRAM	DSP
MNIST	1/1,1/2/1,1	300	6817(2,49%)	9162(1,67%)	34(3,73%)	11(0,44%)
DIBCO	8/4,8/4/8,8	240	18080 (6,60%)	15657(2,86%)	56(6,14%)	6(0,24%)

Table 3 Comparison to FPGA and GPU implementations processing MNIST, DIBCO 2017 datasets.

Name	Platform	Model, [bits]	Params, [M]	Ops. ^a , [M]	Acc., [%]	Freq., [MHz]	θ , [GOp/s]	kFPS ^b	P_{chip} , [W]	P_{board} , [W]	$\frac{kFPS}{P_{chip}}$	$\frac{kFPS}{P_{board}}$
MNIST												
FINN [53]	ZC706	1	3.34E-1	6.69E-1	95.83	200	8265.45	12361	7.30	21.2	1.69E+3	5.83E+2
FINN [53]	ZC706	1	2.91E+0	5.82E+0	98.40	200	9085.67	1561	8.80	22.6	1.77E+2	6.91E+1
BNN [33]	Stratix-V 5SGSD8	1	1.00E+1	2.00E+1	98.32	150	12219.40	610.36	-	26.2	-	2.33E+1
TNN [4]	Kintex-7 160T	2	1.99E-1	3.97E-1	97.76	200	101.28	255.102	0.32	-	7.97E+2	-
TNN [4]	Kintex-7 160T	2	1.72E+0	3.44E+0	98.33	200	877.81	255.102	2.86	-	8.92E+1	-
[42]	ZC706	3	2.90E+0	5.80E+0	98.92	172	384.16	66.255	4.98	11.4	1.41E+1	4.98E+0
This work	ZCU102	1	6.39E-1	2.77E+1	99.37	300	8710.28	314.82	13.20	39.3	2.39E+1	8.01E+0
This work ^D	Tesla K80	32F	6.39E-1	2.77E+1	99.46	-	239.83	8.66	273.85	-	3.16E-2	-
This work ^P	Tesla K80	35F	6.39E-1	2.77E+1	99.46	-	103.33	3.73	193.21	-	1.93E-2	-
DIBCO												
This work	ZCU102	4/8	6.75E-2	1.35E-1	87.54	240	3618.23	6.53	15.47	43.6	4.22E-1	1.50E-1
This work ^D	Tesla K80	32F	6.75E-2	1.35E-1	88.00	-	319.27	0.58	247.47	-	2.34E-3	-
This work ^P	Tesla K80	35F	6.75E-2	1.35E-1	88.00	-	101.91	0.18	183.11	-	9.83E-4	-

^aindicates number of operations per 28×28 image for MNIST, and number of operations per pixel for DIBCO

^btaking into account 28×28 images in the case of MNIST, and 64×64 patches in the case of DIBCO

^Ddiagonal-wise order of execution.

^Ppixel-by-pixel order of execution.

Figure 13 Throughput scalability depending on a number of PE_LSTM and a number of instances (M) of the complete accelerator for MNIST dataset.

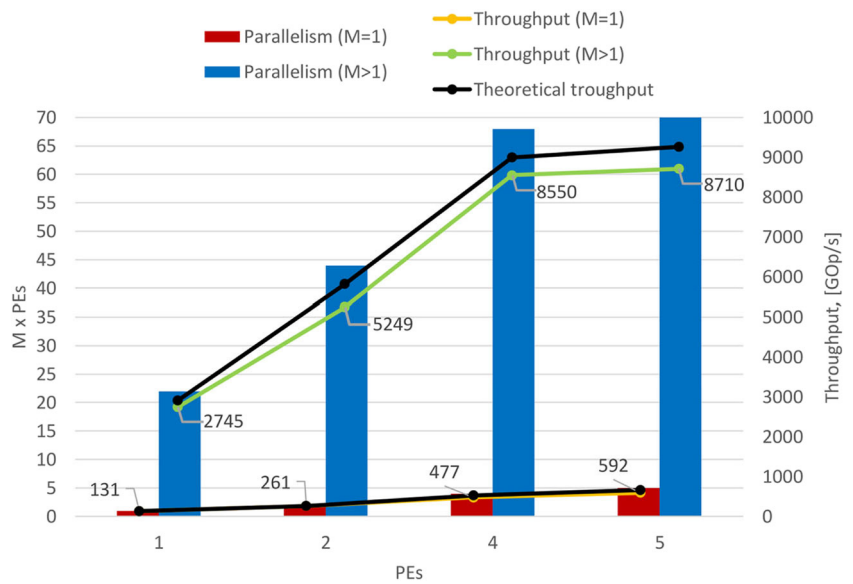
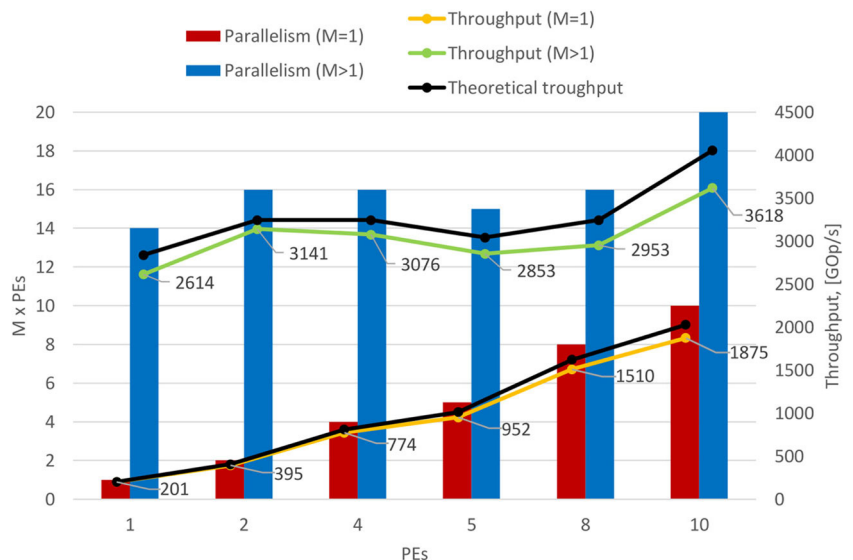


Table 3 presents comparison with earlier FPGA implementations of multilayer perceptron processing MNIST dataset, and our 2D-LSTM implementations for pixel-by-pixel (*P*) and diagonal-wise (*D*) approaches running on GPU for MNIST and DIBCO 2017 datasets. Our FPGA-based implementation processing MNIST dataset outperforms all reference FPGA implementations in accuracy, however it loses to implementations with similar precision in terms of throughput and energy efficiency due to several reasons. First of all, all other implementations use multilayer perceptron models that have much lower implementation complexity due to only feed-forward dataflow and low-complexity activation functions like ReLu or just a step function in the case of binarized models [53]. In contrast, 2D-LSTM has two recurrent paths and multiple non-linear activation functions, and has to process image in four directions. Additionally, the

implemented architecture follows the pixel-by-pixel order of execution that has the highest time complexity. Nevertheless, it shows very promising results taking into account that we did not apply any of the orthogonal approaches for accelerating 2D-LSTM (Section 3.1).

Our FPGA-based implementation outperforms pixel-by-pixel and diagonal-wise approaches running on GPU in terms of throughput by 36-84× and 11-36×, and in terms of energy efficiency by 429-1238× and 180-756×, respectively. However, a difference in throughput between pixel-by-pixel and diagonal-wise GPU implementations is less than expected $(H \times W)/(H + W)$, due to overhead of picking up pixels from diagonals for diagonal-wise processing. Additionally, in the case of MNIST dataset, the batch size for pixel-by-pixel approach was larger than for diagonal-wise approach (2560 vs. 2048, respectively) due

Figure 14 Throughput scalability depending on a number of PE_LSTM and a number of instances (M) of the complete accelerator for DIBCO 2017 dataset.



to lower memory overhead that also reduces the speedup of the later approach. In the case of DIBCO dataset, the batch size was the same for both implementations. Clearly there is a big room for optimization. However, even if we consider perfect speedup with respect to pixel-by-pixel approach, our FPGA-based implementation still outperforms the perfect diagonal-wise GPU implementation in terms of throughput and energy efficiency.

It has been shown that FPGA is a promising computing platform for deep-learning that achieves higher performance and energy efficiency in comparison to conventional platforms. However, even higher levels of efficiency can be achieved following emerging computing paradigms. In recent years, new computing approaches have been proposed targeting much higher energy efficiency and parallelism than ever before. One such technology is Processing-in-Memory (PIM).

4 1D-LSTM PIM

The peak performance and energy efficiency of many advanced neural network accelerators (e.g. LSTM) are constrained due to limited external memory bandwidth and high data access energy. PIM is an emerging computing paradigm that bridges the memory-computation gap by integrating the computation units inside the memory device. The tight coupling of memory and computation logic minimizes the data movement cost, enables the massive internal data parallelism and results in high energy efficiency. In this section, we present a novel DRAM based PIM hardware architecture for binary 1D-LSTM inference. As an initial step, this work only focuses on implementing of a 1D-LSTM inference model in a DRAM-based PIM that will be the base architecture for future variants of LSTM models, such as MD-LSTM.

4.1 Related Work

Recently, there are several research publications on CNN PIM accelerators that are based on RRAM [8, 10, 50, 57], SRAM [3, 5, 16, 28, 34] and DRAM [15, 25, 32, 52] devices. However, a limited number of publications have focused on RNN PIM accelerators. The authors of [37] and [35] propose a PIM accelerator for basic RNNs that are based on SRAM and RRAM, respectively. However, these works did not consider advanced networks, such as LSTMs. The authors of [36] extended the work of [35] for other advanced RNN networks such as LSTMs. The basic computation unit of this architecture is a vector multiplication unit that is realized using RRAM crossbar array. This architecture needs an area, power, and latency intensive 8-bit analog to digital converter (ADC) in the

Sub-Array (SA). The ADCs increase the area of PIM SAs by $10\times$ compared to a standard RRAM SA. The ADCs consumed 43% of the total system power and 49% of the total chip area, respectively. Another issue of RRAM based PIM accelerators is the high process variation that affects the final computation results between the SAs/devices. Additionally, the RRAM process is not technologically as mature as SRAMs or DRAMs. On the other side, SRAM based accelerators have low memory density and are not suitable for networks with a large memory footprint.

Hence, this paper presents a DRAM based LSTM PIM accelerator.

4.2 DRAM Basics

In this section, we describe the DRAM architecture and its functionality in order to provide the required background to understand our PIM architecture. A DRAM device is organized as a set of banks that include memory arrays. Each bank consists of row and column decoders, Master Wordline Driver (MWLD), and Secondary Sense Amplifiers (SSAs) (see Fig. 15). The memory array is designed as an hierarchical structure of SAs (e.g. 1024×1024 cells). To access the data from DRAM, an activate command (*ACT*) is issued to a specific row, which activates the associated MWLD and Local Wordline Drivers (LWDs). This initiates the charge sharing between the memory cells and the respective Local Bitlines (LBLs), the voltage difference on each of these LBLs is sensed by the Primary Sense Amplifiers (PSAs) (e.g. 1024 PSAs) integrated in each SA. The PSAs are integrated on either sides of a SA (half the number of PSAs on each side) and are shared between the two neighbouring SAs. The concurrently sensed data by the PSAs of all the SA in a block (i.e. row of SA) creates an illusion of a large row buffer or a page (e.g. Page size = 1 KB). After row activation, read (*RD*) or write (*WR*) commands are issued to specific columns of this logical row buffer to access the data via *Master Datalines* and SSAs. Each column of SAs (CSA) has a separate set of Master Datalines and SSAs (e.g. 8 per CSAs) that are shared. Reading or writing the data from/to the PSAs of a block (or page) is performed in a fixed granularity of *I/O data width* \times *bursts length* (e.g. $8 \times 8 = 64$ bit in DDR3/4) and equal amount of data is fetched (e.g. 8 bits for x8 device) from all the associated SA (e.g. 8 SAs) of a block. In order to activate a new row in the same bank, a precharge (*PRE*) command has to be issued to equalize the bitlines and to prepare the bank for a new activation.

4.3 Architecture

This section presents the DRAM based 1D-LSTM PIM architecture in detail. One of the major goals of this

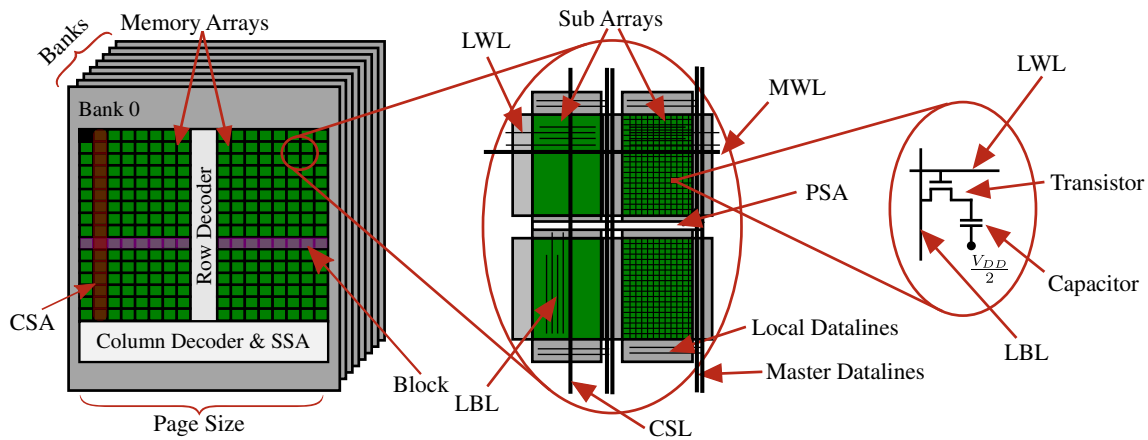


Figure 15 DRAM architecture.

architecture is to conserve the regular and repetitive structure of a DRAM bank to benefit from the high memory density. The computation units should not break this modularity. The aim is to maximize the data parallelism and minimize the data transfer cost by integrating the computation units close to the storage units without modifying the highly optimized commodity SA. Hence, the nearest possible location to integrate the computation units is at the output of the PSA where the data parallelism is maximum. However, the DRAM architecture and its process restrict the integration of logic units in this region i.e. at the output of the PSA due to the following reasons:

- PSA region has no clocking signals i.e. only combinational logic can be integrated.
- Among the three available metal layers in the DRAM process one metal layer in this region is utilized for the LBL of the neighboring SA due to PSA sharing. Hence, the computation logic will only have two metal layers, requiring more engineering efforts for floor planning and routing.
- The width (in μm) of the computational logic is limited to the SA width in order to conserve the repetitive structure of a DRAM bank.

Contrarily, integrating the computation logic near the SSAs do not impose hard design constraints but compromises the parallelism. Hence, our new architecture partitions the computation logic into basic computation units and secondary processing units (SPU) in order to attain a balance between parallelism and design constraints of the DRAM architecture and its process. Only, the basic computation units are integrated at the output of the PSA in each SA. The rest of the logic required to process the output of these basic units are in SPU that is integrated at the output of

the SSAs. The computational hierarchy matches the memory hierarchy without affecting the structure of the DRAM bank. The basic computation units are explicitly built using combinational logic due to lack of clocking signals near the PSA.

In the following, we first elaborate the micro-architecture of a basic unit followed by the details of the overall architecture. The proposed architecture is for a state-of-the-art commodity 8 Gb x8 DDR4 DRAM device with 16 banks, 1024×1024 SA dimension, 16 SAs per block (8 per Half-Bank see Fig. 15.), 2 KB page size, 1024 PSAs per SA, 128 master bitlines (eight per SA), and 128 SSAs (eight per CSA). The architecture is scalable to other dynamic random access memory (DRAM)s and their technologies. This architecture targets a binary 1D-LSTM with 1 bit weight (i.e. *logic 0* $\rightarrow +1$, *logic 1* $\rightarrow -1$), 2 bit data and 2 bit activation. The authors of [24] and [48] have shown that the binary networks have negligible impacts on the accuracy as compared to full-precision networks for various applications such as OCR.

4.3.1 Basic Computation Unit

The basic computation unit (i.e. dot-product unit) consists of N multiplication units and an adder-tree as shown in Fig. 16. A single multiplication unit for a binary NN is realized using two XOR gates and a 2s complement converter (as shown in Fig. 17.). The 2s complement converter is a simple adder that adds logic 1 to the output of the XOR gates if the sign of the weight is negative.

The 3-bit output of the multiplication units is fed to an adder tree that outputs the dot-product result of two 1D-arrays of dimension N . The value of the N is crucial as it directly influences the size/depth of the adder-tree and can potentially blow up the design near the SA.. Hence, based

on the architectural exploration the value of N is confined to 32 (i.e. 8-bit dot-product result), which also satisfies all the aforementioned physical implementation constraints.

The architecture specifically stores the weight array (i.e. W or R) and its corresponding data array (i.e. x^t or y^{t-1}) in different rows of the same SA. Elements of the weight array are aligned column-wise with the corresponding elements of the data array. For example, the element zero of weight array has been stored in the same columns of a SA as element zero of the data array. Hence, 1-bit weights are duplicated to align with 2-bit data in the column direction. Before the start of the dot-product computations, the weight array row is first activated and are accessed by the PSA. These weights are also buffered in shadow latch (SHL) by enabling the isolation (ISO) switch. Once the weights are latched in shadow-latches it is reused for several computations by changing only the corresponding data in PSAs (i.e. activation of a new row), refer Fig. 17.

A single SA of a DRAM bank is integrated with 16 basic computation units (i.e. dot-product units). Each of this unit can only compute the dot-product of the arrays whose dimension are less than or equal to 32. If the dimensions of the dot-product operation are larger than 32 then the dot-product operation is broken down to multiple smaller dot-products such that it could be mapped to the basic units. The 8-bit partial dot-product results computed by multiple basic units are transferred to the SSA region (i.e. same as read operation) where they are added in the SPU to obtain the final result of the large dot-product operation. The 8-bit width of the basic unit output also matches with the DRAM architecture since the master bitlines connecting a single SA to the SSAs are also 8-bit wide.

4.3.2 Overall Architecture

Figure 18 shows the architecture of a single bank that is enhanced with LSTM computation units. The architecture exploits different levels of parallelism (i.e SA parallelism, bank parallelism) available in DRAM to achieve high

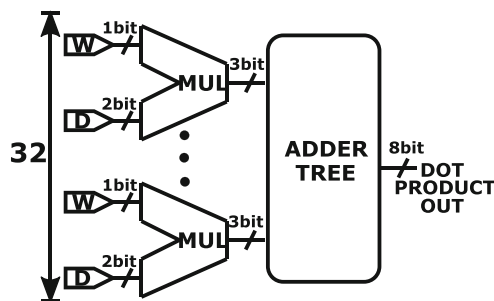


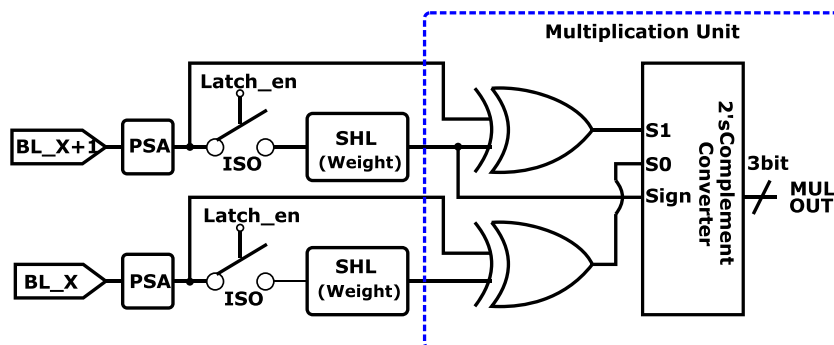
Figure 16 Dot-Product Unit.

throughput. The computations of each LSTM cell are mapped to an individual CSA (refer Fig. 15) in a DRAM bank. Hence, the computations of each LSTM cell are executed in parallel. For example, a DRAM with 16 banks and 16 CSAs per bank executes a maximum of 256 LSTM cells in parallel. In our architecture, one bank, called as FC-bank, is reserved for the fully-connected layer computations. The remaining banks are used for LSTM cell computations. Hence, the maximum LSTM cells executed in parallel is 240. This architecture offers flexibility to configure any bank as FC-bank.

In an individual CSA, the dot-product operations of each gate of an LSTM cell are mapped to an individual SA or to a group of SAs. These SAs operate in a pipeline fashion i.e. while a particular SA is transferring the results of 16 basic units to the SPU unit, another SA belonging to the same CSA starts its computation by activating (i.e. ACT) the required rows of that sub-array. The transfer of basic unit output from the SA to the SSA region is similar to a read operation. The CSL lines that are used to select the required burst of data from the currently active row (in the PSAs) for a read/write operation are also used to select one of the 16 basic units outputs when a SA is operated in computation mode. The selected 8-bit data from a SA in each CSA is then transferred to the respective SSAs/SPU via the master bitlines. Figure 19 shows the block diagram of a SPU. The partial dot-product results computed in the basic computation unit are added using a sequential adder in the SPU to complete the summation operations of each LSTM gate (e.g. $W_a x^t + R_a y^{t-1} + b_a$, see Eq. 1). The sequential adder and the transfer operations are executed in pipeline manner. Hence, adding only one clock cycle delay to the overall latency. The intermediate results of each LSTM gate are stored in a buffer. As the summation result corresponding to a gate for an instance t is finished, the *dot-product done* signal is set high and the result is transferred to an appropriate activation function (i.e. tanh or sigmoid). The non-linear activation functions are implemented as a piece-wise linear function similar to [40]. Figure 20. shows the comparison between the actual value and the hardware implemented piece-wise linear sigmoid function. Once the activation function outputs a^t , i^t , f^t and o^t are computed for a given instance, c^t and y^t computations are triggered.

Each of the SPU produces a 2-bit output corresponding to an LSTM cell. The 2-bit output of all concurrently executed LSTM cells are concatenated to form the y^t output array. If the LSTM cells are distributed across the banks, each bank broadcasts its 16×2 -bit (i.e. 32-bit) result to all other banks including FC-bank. This architecture does not use an explicit bus for such broadcasting. It reuses the shared

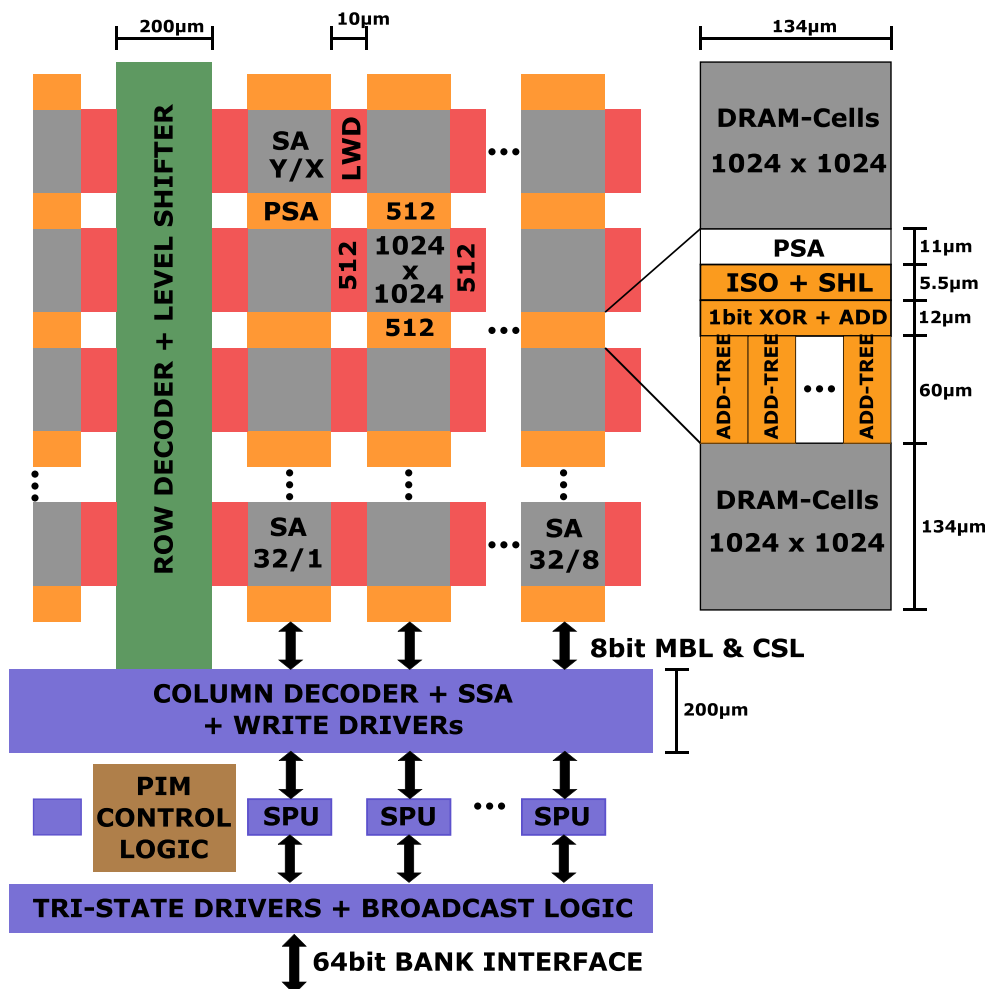
Figure 17 Basic unit at the PSAs.



bus (i.e. 64 bit wide for DDR4) that is used to transfer the data from the banks to the interface and vice-versa. Once the y_t output array is available in all the banks, it is written back to the appropriate SAs that are responsible for the computations of the next instance in each CSA. Note that the writebacks are concurrent in all the CSAs. Since the y^t is applicable to all the gates of an LSTM cell, this y^t has to be written multiple time to all the memory locations that are responsible for the respective LSTM gate

computations within the CSA. This writing of the same data multiple times within the CSA imposes a huge latency. Hence, the architecture enables multiple CSL lines instead of only 8 CSLs per CSA. This enables concurrent writes of the same data to multiple 8-bit word memory locations in each CSA but at the cost of increased load capacity of the write drivers. Since the LSTM cells have a maximum of four gates (including input to a memory cell), the concurrent write is limited to four memory locations in each CSA,

Figure 18 PIM bank architecture.



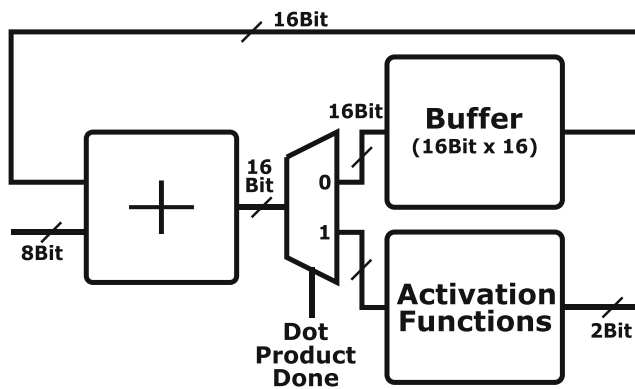


Figure 19 SPU unit.

which requires an increase in driver strength by 10-15% only.

Although the architecture considers each CSA as a parallel entity, the commodity DRAM bank architecture imposes a major restriction. All the SAs belonging to the same block (i.e. row of SAs) has to perform the same set of operations at any given time due to coupling of these SAs by the MWL and other control lines.

The basic unit computations require activation of rows and these commands in a DRAM bank corresponds to an entire block that activates the same row in each SA of that block. Similarly, transfer of data between the SA and SSAs involve read and write commands that are applicable for the entire block i.e. all the SAs in a block are concurrently in data transfer mode. However, the restriction that imposes dependency between the CSAs is addressed by mapping and executing the same processing steps of the respective LSTM cell in each CSA in a bank.

FC-bank stores the y^t results and performs the fully-connected operations of the previous instance (i.e. $t - 1$) in parallel with the LSTM cell operations of the current instance (i.e. t). The same basic units and the sequential adder in the SPU are sufficient for the fully-connected layer. The final results of the fully-connected layer are also stored

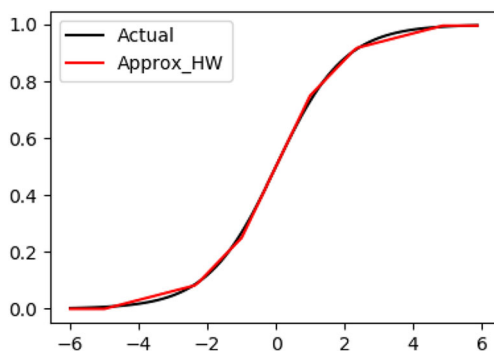


Figure 20 Actual vs the hardware implemented piece-wise linear sigmoid function.

in the same bank. The number of dot-product operations per unit/neuron of the fully-connected layer is less than the LSTM cell i.e. LSTM cells are more computationally intensive. Hence, the FC-bank finishes its operations before the availability of y^t results of the next instance. This architecture only performs the operations corresponding to the input layer, hidden layer and fully-connected layer of the LSTM inference engine. The remaining processing steps like softmax and connectionist temporal classification (CTC) are assumed to be executed in an external processing unit.

4.4 Experimental Setup

This work investigated all the relevant DRAM circuits (e.g. PSAs) and the aforementioned PIM circuits using the DRAM architecture exploration tool DRAMSpec [39] and an adapted UMC 65 nm Low-Leakage CMOS bulk technology, respectively. The UMC 65 nm process technology was modified to emulate a 65 nm DRAM process by changing the transistor parameters, restricting the number of metal layers (three metal-layers only), and layout rules, such as the minimum metal pitch. The area, power, latency results of all the PIM circuitry are based on the exhaustive circuit level simulations and the full custom layout implementations that are done using Cadence Spectre and Virtuoso, respectively. To retain the modular and regular architecture of commodity DRAMs, the layout confines the total width of the PIM circuitry integrated near the SA to the width of the SA.

The rest of the DRAM architecture results, including SAs, SSAs, peripherals circuitry, row and column decoders, are evaluated with DRAMSpec. All the estimated results are for the DRAM core clock period of 3ns. The results obtained from DRAMSpec and the circuit-level simulations are substituted in the bit-true model of the proposed PIM architecture to calculate the total area, power consumption, and performance. Finally, the calculated total area in 65 nm technology is conservatively scaled to a 20 nm node to make a fair comparison with recent commodity DRAMs.

4.5 Results

This section presents the experimental results of our DRAM based PIM implementation. The system performance is evaluated using the inference of a Bi-LSTM model for OCR application adapted from [48]. The Bi-LSTM model is trained for 1-bit weight and 2-bit data/activation that achieves a classification accuracy of 94% for the same dataset as used in [48]. The inference of the Bi-LSTM model is also implemented on Xilinx Zynq UltraScale+MPSoC ZCU102 board featuring a XCZU9EG device using the same architecture as in [48]. The FPGA implementation is used as a baseline for comparison in this paper.

Table 4 Area comparison of DRAM PIM vs Standard DRAMs

Type	Area, [mm ²]
Standard DRAM 8Gb 65 nm (Calculated)	207.15
Standard DRAM 8Gb 20 nm (Scaled)	63.73
Standard DRAM 8Gb 20 nm (Samsung-DDR4) [11]	59.00
Standard DRAM 8Gb 21 nm Gen-1 (Hynix-DDR4) [12]	76.0
Standard DRAM 8Gb 21 nm Gen-2 (Hynix-DDR4) [12]	53.6
Standard DRAM 8Gb 18 nm (Micron-DDR4) [13]	58.48
PIM DRAM 8Gb 65 nm (all MAC rows)	385.55
PIM DRAM 8Gb 20 nm (all MAC rows)	118.63
PIM DRAM 8Gb 65 nm (4 MAC rows)	242.69
PIM DRAM 8Gb 20 nm (4 MAC rows)	74.67

4.5.1 Area

We implemented the PIM architecture and designed all necessary circuits in a 65 nm technology. The obtained die area results are conservatively scaled to 20 nm in order to compare it to a state-of-the-art 8 Gb commodity DRAM. Table 4 shows the calculated area results of a standard DRAM die and the PIM enhanced DRAM die for a 65 nm technology and for a scaled 20 nm technology. The scaled area of a standard DRAM die for 20 nm technology is comparable to 8 Gb commodity DRAM dice from various vendors. This shows the correctness of our estimation methodology.

The presented area results for the PIM enhanced DRAM are evaluated for two architecture variants, (1) all DRAM SAs are enhanced with basic computation units (referred as *all MAC rows*) (2) limited number of blocks (four in this case) in each bank are enhanced with basic computation units (referred as *4 MAC rows*). 4 MAC rows are sufficient to store all the parameters of the specified Bi-LSTM inference model. However, for the networks with larger parameter size, the parameters corresponding to initial timestamps are stored in the available space of 4 MAC rows. The parameters of the subsequent timestamps are overwritten on a timely basis that is stored in non-PIM SAs.

Alternatively, a suitable architecture variants like *8 MAC rows* or *16 MAC rows* can be chosen at a cost of larger area.

The *4 MAC rows* architecture variant is preferable as it achieves a high memory density, similar to a commodity DRAM chip. This architecture variant (i.e *4 MAC rows*) enables the DRAM based PIM chips to be used as both a conventional memory device and an accelerator.

Such a PIM chip has a total area overhead of 18% compared to an 8 Gb commodity DRAM. All the evaluation results presented henceforth are only for the *4 MAC rows* variant.

4.5.2 Power Consumption

The power overhead of NN circuits compared to total power consumption is only 10%. The remaining power consumption is due to the conventional DRAM operations required for the computation. The prime benefit of this architecture is the low power consumption. The total power of our PIM implementation is 0.384 W, while the FPGA implementation consumes a power of 19 W measured using Zynq UltraScale+ MPSoC Power Advantage Tool [1]. Only the FPGA core power is presented not the board power for a fair comparison.

Table 5 Results of our work against RRAM based LSTM PIM

	RRAM-PIM [36]	This work
Technology [nm]	28	20
Quantization [bits]	16	2
Dataset	HAR	Plaintext
Input size	28	32
Number of LSTM cells	128	128
Area overhead [%]	≈50	18
Power [W]	0.6	0.384
Computation efficiency [GOp/s/W]	116.3	3409.28

4.5.3 Throughput and Energy Efficiency

The throughput of the proposed architecture for the aforementioned Bi-LSTM inference is 6329113 input layer columns per second that accounts to 1309.16 GOp/s. The same inference executed on an aforementioned FPGA results a throughput of 4000 GOp/s. Although, the FPGA based implementation achieves higher throughput, the goal of our new architecture is mainly to optimize the energy efficiency, i.e. performance per Watt. The PIM based implementation is $16.19\times$ more energy efficient compared to an FPGA based implementation. As there are no prior publications on DRAM based LSTM PIM, we show for the sake of completeness the results of this work against the results of the RRAM based implementation from [36] (Refer Table 5). Note that, both the systems are designed for different quantizations and the obtained results are for different applications. Hence, a fair comparison of these two works is not possible.

5 Conclusion

We presented two novel hardware architectures for inference of LSTM neural networks.

First, a hardware architecture for MD-LSTM neural network was presented. We conducted a systematic exploration of accuracy as a function of hardware cost for various precisions using Pareto frontier for DIBCO 2017 and MNIST datasets. It was demonstrated that MD-LSTM achieves an accuracy that is comparable with state-of-the-art for MNIST dataset with binary weights and 2-bit activation. It also approaches best results for DIBCO 2017 dataset with low bit-width weights and activation. Based on this new architecture we implemented an FPGA-based accelerator that outperforms NVIDIA K80 GPU implementation in terms of runtime and energy efficiency. At the same time, our accelerator demonstrates higher accuracy and comparable throughput in comparison to state-of-the-art FPGA-based implementations of multilayer perceptron for MNIST dataset.

Second, we proposed a DRAM based PIM architecture for 1D-LSTM with binary weights and 2-bit data/activation. This architecture is explicitly proposed for energy efficient computing and addressing the memory bound issues of LSTM networks. One of the major highlights of our architecture is retaining the regular and repetitive DRAM structure without compromising the memory density. The small area overhead (18%) compared to an 8 Gb DDR4 DRAM qualifies this architecture to be embedded in upcoming DRAM devices. The results show that the PIM implementation is $16.19\times$ more energy efficient as compared to the FPGA based design. In the future, this work

will be extended to different input data quantizations and weights, and to MD-LSTM.

We have shown that the new compute paradigm PIM, and FPGAs are very promising platforms for deep learning that can offer a solution in cases, where conventional platforms, such as GPU, fail to provide the necessary performance and energy efficiency required by the applications.

Acknowledgments This work was initiated in the context of a cooperation with Huawei. The project OPRECOMP acknowledges the financial support of the Future and Emerging Technologies (FET) programme within the European Unions Horizon 2020 research and innovation programme, under grant agreement No.732631 (<http://www.oprecomp.eu>). Partially this work was supported by the InnoProm program of the state Rhineland-Palatinate, Germany. The training was partly performed on High Performance Computer 'Elwetritsch' at the TU Kaiserslautern. We also acknowledge Cadence and Synopsys for the design and evaluation of ASIC circuits.

Funding Open Access funding enabled and organized by Projekt DEAL.

Open Access This article is licensed under a Creative Commons Attribution 4.0 International License, which permits use, sharing, adaptation, distribution and reproduction in any medium or format, as long as you give appropriate credit to the original author(s) and the source, provide a link to the Creative Commons licence, and indicate if changes were made. The images or other third party material in this article are included in the article's Creative Commons licence, unless indicated otherwise in a credit line to the material. If material is not included in the article's Creative Commons licence and your intended use is not permitted by statutory regulation or exceeds the permitted use, you will need to obtain permission directly from the copyright holder. To view a copy of this licence, visit <http://creativecommons.org/licenses/by/4.0/>.

References

1. Zynq UltraScale MPSoC Power Advantage Tool. <https://xilinx-wiki.atlassian.net/wiki/spaces/A/pages/18841813/Zynq+UltraScale+MPSoC+Power+Management>.
2. Afzal, M. Z., Pastor-Pellicer, J., Shafait, F., Breuel, T. M., Dengel, A., & Liwicki, M. (2015). Document image binarization using lstm: a sequence learning approach. In *Proceedings of the 3rd international workshop on historical document imaging and processing* (pp. 79–84). ACM.
3. Agrawal, A., Jaiswal, A., Roy, D., Han, B., Srinivasan, G., Ankit, A., & Roy, K. (2019). Xcel-RAM: accelerating binary neural networks in high-throughput SRAM compute arrays. *IEEE Transactions on Circuits and Systems I: Regular Papers*, 66(8), 3064–3076. <https://doi.org/10.1109/TCSI.2019.2907488>.
4. Alemdar, H., Leroy, V., Prost-Boucle, A., & Pétrot, F. (2017). Ternary neural networks for resource-efficient ai applications. In *2017 international joint conference on neural networks (IJCNN)* (pp. 2547–2554). IEEE.
5. Ando, K., Ueyoshi, K., Orimo, K., Yonekawa, H., Sato, S., Nakahara, H., Takamaeda-Yamazaki, S., Ikebe, M., Asai, T., Kuroda, T., & Motomura, M. (2018). BRein memory: a single-chip binary/ternary Reconfigurable in-memory deep neural network accelerator achieving 1.4 TOPS at 0.6 W. *IEEE Journal of Solid-State Circuits*, 53(4), 983–994. <https://doi.org/10.1109/JSSC.2017.2778702>.

6. Breuel, T. M. (2015). Benchmarking of lstm networks. arXiv:1508.02774.
7. Byeon, W., Breuel, T. M., Raue, F., & Liwicki, M. (2015). Scene labeling with lstm recurrent neural networks. In *Proceedings of the IEEE conference on computer vision and pattern recognition* (pp. 3547–3555).
8. Chen, X., Zhu, J., Jiang, J., & Tsui, C.Y. (2019). CompRRAE: RRAM-based convolutional neural network accelerator with reduced computations through a runtime activation estimation. In *Proceedings of the 24th Asia and South Pacific design automation conference* (pp. 133–139). New York: ASPDAC '19, ACM. <https://doi.org/10.1145/3287624.3287640>.
9. Chen, Y., Krishna, T., Emer, J. S., & Sze, V. (2017). Eyeriss: an energy-efficient reconfigurable accelerator for deep convolutional neural networks. *IEEE Journal of Solid-State Circuits*, 52(1), 127–138. <https://doi.org/10.1109/JSSC.2016.2616357>.
10. Chi, P., Li, S., Xu, C., Zhang, T., Zhao, J., Liu, Y., Wang, Y., & Xie, Y. (2016). PRIME: a novel processing-in-memory architecture for neural network computation in reRAM-based main memory. In *2016 ACM/IEEE 43rd annual international symposium on computer architecture (ISCA)* (pp. 27–39). <https://doi.org/10.1109/ISCA.2016.13>.
11. Choe, J. (2017). Samsung 18 nm DRAM cell integration: QPT and higher uniformed capacitor high-k dielectrics. <https://www.techinsights.com/blog/samsung-18-nm-dram-cell-integration-qpt-and-higher-uniformed-capacitor-high-k-dielectrics>.
12. Choe, J. (2017). SK hynix' 21 nm DRAM cell technology: comparison of 1st and 2nd generation. <https://www.techinsights.com/blog/sk-hynix-21-nm-dram-cell-technology-comparison-1st-and-2nd-generation>.
13. Choe, J. (2018). Micron's 1x DRAMs examined. https://www.eetimes.com/author.asp?section_id=36&doc_id=1333289.
14. Davidson, B., Kalitzeos, A., Carroll, J., Dubra, A., Ourselin, S., Michaelides, M., & Bergeles, C. (2018). Automatic cone photoreceptor localisation in healthy and stargardt afflicted retinas using deep learning. *Scientific Reports*, 8(1), 7911.
15. Deng, Q., Jiang, L., Zhang, Y., Zhang, M., & Yang, J. (2018). DrAcc: a DRAM based accelerator for accurate CNN inference. In *Proceedings of the 55th annual design automation conference* (pp. 168:1–168:6). New York: DAC '18, ACM. <https://doi.org/10.1145/3195970.3196029>.
16. Eckert, C., Wang, X., Wang, J., Subramaniyan, A., Iyer, R., Sylvester, D., Blaauw, D., & Das, R. (2018). Neural cache: bit-serial in-cache acceleration of deep neural networks. In *Proceedings of the 45th annual international symposium on computer architecture* (pp. 383–396). Piscataway: ISCA '18, IEEE Press. <https://doi.org/10.1109/ISCA.2018.00040>.
17. Glorot, X., & Bengio, Y. (2010). Understanding the difficulty of training deep feedforward neural networks. In *Proceedings of the thirteenth international conference on artificial intelligence and statistics* (pp. 249–256).
18. Graves, A. (2012). Supervised sequence labelling. In *Supervised sequence labelling with recurrent neural networks* (pp. 5–13). Springer.
19. Graves, A., Fernández, S., & Schmidhuber, J. (2007). Multidimensional recurrent neural networks. In *International conference on artificial neural networks* (pp. 549–558). Springer.
20. Graves, A., & Schmidhuber, J. (2005). Framewise phoneme classification with bidirectional LSTM and other neural network architectures. *Neural Networks*, 18(5–6), 602–610.
21. Han, S., Kang, J., Mao, H., Hu, Y., Li, X., Li, Y., Xie, D., Luo, H., Yao, S., Wang, Y., & et al. (2017). Ese: efficient speech recognition engine with sparse lstm on fpga. In *Proceedings of the 2017 ACM/SIGDA international symposium on field-programmable gate arrays* (pp. 75–84). ACM.
22. Hochreiter, S., & Schmidhuber, J. (1997). Long short-term memory. *Neural Computation*, 9(8), 1735–1780.
23. Hou, L., Yao, Q., & Kwok, J.T. (2016). Loss-aware binarization of deep networks. arXiv:1611.01600.
24. Hubara, I., Courbariaux, M., Soudry, D., El-Yaniv, R., & Bengio, Y. (2016). Quantized neural networks: Training neural networks with low precision weights and activations. arXiv:1609.07061.
25. Jiang, L., Kim, M., Wen, W., & Wang, D. (2017). XNOR-POP: a processing-in-memory architecture for binary convolutional neural networks in wide-IO2 DRAMs. In *2017 IEEE/ACM international symposium on low power electronics and design (ISLPED)* (pp. 1–6(7)). <https://doi.org/10.1109/ISLPED.2017.8009163>.
26. Jouppi, N. P., Young, C., Patil, N., Patterson, D., Agrawal, G., Bajwa, R., Bates, S., Bhatia, S., Boden, N., Borchers, A., & et al. (2017). In-datacenter performance analysis of a tensor processing unit. In *2017 ACM/IEEE 44th annual international symposium on computer architecture (ISCA)* (pp. 1–12). IEEE.
27. Kalchbrenner, N., Danihelka, I., & Graves, A. (2015). Grid long short-term memory. arXiv:1507.01526.
28. Kang, M., Gonugondla, S. K., Patil, A., & Shanbhag, N.R. (2018). A multi-functional in-memory inference processor using a standard 6T SRAM array. *IEEE Journal of Solid-State Circuits*, 53(2), 642–655. <https://doi.org/10.1109/JSSC.2017.2782087>.
29. Kowsari, K., Heidarysafa, M., Brown, D. E., Meimandi, K. J., & Barnes, L.E. (2018). Rmdl: random multimodel deep learning for classification. In *Proceedings of the 2nd international conference on information system and data mining* (pp. 19–28). ACM.
30. LeCun, Y., Bottou, L., Bengio, Y., Haffner, P., & et al. (1998). Gradient-based learning applied to document recognition. *Proceedings of the IEEE*, 86(11), 2278–2324.
31. Leifert, G., Strauß, T., Grüning, T., Wustlich, W., & Labahn, R. (2016). Cells in multidimensional recurrent neural networks. *The Journal of Machine Learning Research*, 17(1), 3313–3349.
32. Li, S., Niu, D., Malladi, K. T., Zheng, H., Brennan, B., & Xie, Y. (2017). DRISA: a DRAM-based Reconfigurable in-situ accelerator. In *Proceedings of the 50th annual IEEE/ACM international symposium on microarchitecture* (pp. 288–301). New York: MICRO-50 '17, ACM. <https://doi.org/10.1145/3123939.3123977>.
33. Liang, S., Yin, S., Liu, L., Luk, W., & Wei, S. (2018). Fpbn: binarized neural network on fpga. *Neurocomputing*, 275, 1072–1086.
34. Liu, R., Peng, X., Sun, X., Khwa, W., Si, X., Chen, J., Li, J., Chang, M., & Yu, S. (2018). Parallelizing SRAM arrays with customized bit-cell for binary neural networks. In *2018 55th ACM/ESDA/IEEE design automation conference (DAC)* (pp. 1–6). <https://doi.org/10.1109/DAC.2018.8465935>.
35. Long, Y., Jung, E. M., Kung, J., & Mukhopadhyay, S. (2016). reRAM crossbar based recurrent neural network for human activity detection. In *2016 international joint conference on neural networks (IJCNN)* (pp. 939–946). <https://doi.org/10.1109/IJCNN.2016.7727299>.
36. Long, Y., Na, T., & Mukhopadhyay, S. (2018). reRAM-based processing-in-memory architecture for recurrent neural network acceleration. *IEEE Transactions on Very Large Scale Integration (VLSI) Systems*, 26(12), 2781–2794. <https://doi.org/10.1109/TVLSI.2018.2819190>.
37. Mathuriya, A., Manipatruni, S., Lee, V., Sumbul, H., Chen, G., Kumar, R., Knag, P., Krishnamurthy, R., Young, I., & Sharma, A. (2019). In-memory analog neural cache. <http://www.freepatentsonline.com/y2019/0057304.html>.
38. Moysset, B., & Messina, R. (2018). Are 2d-lstm really dead for offline text recognition? arXiv:1811.10899.
39. Naji, O., Weis, C., Jung, M., & Wehn, N. (2015). Hansson, a.: a high-level DRAM timing, power and area exploration tool. In *2015 International conference on embedded computer systems:*

- architectures, modeling, and Simulation (SAMOS) (pp. 149–156(7)). <https://doi.org/10.1109/SAMOS.2015.7363670>.
40. Ngah, S., Bakar, R. A., Embong, A., & Razali, S. (2016). Two-steps implementation of sigmoid function for artificial neural network in field programmable gate array. In *ARNP journal of engineering and applied sciences*.
 41. Oord, A. V. D., Kalchbrenner, N., & Kavukcuoglu, K. (2016). Pixel recurrent neural networks. arXiv:1601.06759.
 42. Park, J., & Sung, W. (2016). Fpga based implementation of deep neural networks using on-chip memory only. In *2016 IEEE International conference on acoustics, speech and signal processing (ICASSP)* (pp. 1011–1015). IEEE.
 43. Pham, V., Bluche, T., Kermorvant, C., & Louradour, J. (2014). Dropout improves recurrent neural networks for handwriting recognition. In *2014 14th international conference on frontiers in handwriting recognition*. pp. 285–290. IEEE.
 44. Pratikakis, I., Zagoris, K., Barlas, G., & Gatos, B. (2017). Icdar2017 competition on document image binarization (dibco 2017). In *2017 14th IAPR international conference on document analysis and recognition (ICDAR)*. vol. 1, pp. 1395–1403. IEEE.
 45. Puigcerver, J. (2017). Are multidimensional recurrent layers really necessary for handwritten text recognition? In *2017 14th IAPR international conference on document analysis and recognition (ICDAR)*. vol. 1, pp. 67–72. IEEE.
 46. Rastegari, M., Ordonez, V., Redmon, J., & Farhadi, A. (2016). Xnor-net: Imagenet classification using binary convolutional neural networks. In *European conference on computer vision*. pp. 525–542. Springer.
 47. Ronneberger, O., Fischer, P., & Brox, T. (2015). U-net: Convolutional networks for biomedical image segmentation. In *International conference on medical image computing and computer-assisted intervention*. pp. 234–241. Springer.
 48. Rybalkin, V., Pappalardo, A., Ghaffar, M. M., Gambardella, G., Wehn, N., & Blott, M. (2018). Finn-l: library extensions and design trade-off analysis for variable precision lstm networks on fpgas. In *2018 28th international conference on field programmable logic and applications (FPL)*. pp. 89–897. IEEE.
 49. Rybalkin, V., Wehn, N., Yousefi, M. R., & Stricker, D. (2017). Hardware architecture of bidirectional long short-term memory neural network for optical character recognition. In *Proceedings of the conference on design, automation & test in Europe*. pp. 1394–1399. European Design and Automation Association.
 50. Song, L., Qian, X., Li, H., & Chen, Y. (2017). PipeLayer: a pipelined reRAM-based accelerator for deep learning. In *2017 IEEE international symposium on high performance computer architecture (HPCA)*. pp. 541–552. <https://doi.org/10.1109/HPCA.2017.55>.
 51. Stollenga, M. F., Byeon, W., Liwicki, M., & Schmidhuber, J. (2015). Parallel multi-dimensional lstm, with application to fast biomedical volumetric image segmentation. In *Advances in neural information processing systems*. pp. 2998–3006.
 52. Sudarshan, C., Lappas, J., Ghaffar, M. M., Rybalkin, V., Weis, C., Jung, M., & Wehn, N. (2019). An in-DRAM neural network processing engine. In *2019 IEEE international symposium on circuits and systems (ISCAS)*. pp. 1–5. <https://doi.org/10.1109/ISCAS.2019.8702458>.
 53. Umuroglu, Y., Fraser, N. J., Gambardella, G., Blott, M., Leong, P., Jahre, M., & Vissers, K. (2017). Finn: a framework for fast, scalable binarized neural network inference. In *Proceedings of the 2017 ACM/SIGDA international symposium on field-programmable gate arrays*. pp. 65–74. ACM.
 54. Voigtlaender, P., Doetsch, P., & Ney, H. (2016). Handwriting recognition with large multidimensional long short-term memory recurrent neural networks. In *2016 15th international conference on frontiers in handwriting recognition (ICFHR)*. pp. 228–233. IEEE.
 55. Wenniger, G. M. D. B., Schomaker, L., & Way, A. (2019). No padding please: efficient neural handwriting recognition. arXiv:1902.11208.
 56. Xu, C., Yao, J., Lin, Z., Ou, W., Cao, Y., Wang, Z., & Zha, H. (2018). Alternating multi-bit quantization for recurrent neural networks. arXiv:1802.00150.
 57. Yu, S., Li, Z., Chen, P., Wu, H., Gao, B., Wang, D., Wu, W., & Qian, H. (2016). Binary neural network with 16 Mb RRAM macro chip for classification and online training. In *2016 IEEE international electron devices meeting (IEDM)*. pp. 16.2.1–16.2.4. <https://doi.org/10.1109/IEDM.2016.7838429>.
 58. Zhou, S., Wu, Y., Ni, Z., Zhou, X., Wen, H., & Zou, Y. (2016). Dorefa-net: training low bitwidth convolutional neural networks with low bitwidth gradients. arXiv:1606.06160.

Publisher's Note Springer Nature remains neutral with regard to jurisdictional claims in published maps and institutional affiliations.



Vladimir Rybalkin received his Master's degree in Electrical and Computer Engineering from University of Kaiserslautern, Germany, in 2014. Currently, he is a research associate doing a PhD in Microelectronic Systems Design Research Group at the University of Kaiserslautern. His research interests lie in hardware acceleration for applications ranging from financial computations and big data algorithms to machine learning and deep learning algorithms for hand gesture recognition, historical document analysis, and medical applications. The main focus of his research is given to designing efficient hardware architectures for recurrent neural networks.



Chirag Sudarshan received his Master's (2017) degree in Electrical and Computer Engineering from the University of Kaiserslautern, Germany. He is working towards the PhD degree with the Microelectronic Systems Design Research Group, in the same university. His research interests are DRAM architecture, Process-in-Memory, DRAM controllers, and emerging memory technologies.



Christian Weis received the Ph.D. degree in electrical engineering from the TU Kaiserslautern, Germany, in 2014. From 1996 to 1998, he was with Mitsubishi Semiconductor Europe, Germany, where he was engaged in the design and development of microcontrollers. From 1998 to 2009, he was with Siemens Semiconductor, Infineon Technologies AG and Qimonda AG, Munich, Germany, in DRAM design. During this time frame, he

was involved in DRAM design for graphics and commodity DRAM products. In 2006, he was a Design Team Leader for the 1Gb DDR3 DRAM, the first DDR3 volume product at Infineon/Qimonda. Since 2009, he has been with the Microelectronic System Design Research Group, TU Kaiserslautern, Germany. He holds several patents related to DRAMs and DRAM design, and published more than 60 papers. His current research interests include DRAM controller design, Near- & In-Memory processing, 3D-integrated DRAMs, heterogeneous memory architectures, and MPSoCs.



Jan Lappas received the Master's degree in electrical engineering from the University of Kaiserslautern, Germany, in 2017. Since 2017 he is a researcher in the Microelectronic Systems Design Research Group in the Electrical and Computer Engineering department of the University of Kaiserslautern. His research interest are design automation of analog-mixed signal designs, modelling of memory systems as well as in-memory processing.



Norbert Wehn holds the chair for Microelectronic System Design in the department of Electrical Engineering and Information Technology at the University of Kaiserslautern. He received his Diploma and PhD from the TU Darmstadt in Germany. He is associate editor of various journals and member of several scientific industrial advisory boards. He has more than 350 publications in various fields of microelectronic system design and holds 20 patents. His

special research interests are VLSI-architectures for mobile communication, forward error correction techniques, low-power techniques, advanced SoC and memory architectures, 3D integration, reliability issues in SoC, IoT and hardware accelerators for machine learning.



Li Cheng received the B.S. degree in physics and the Ph.D. degree in electronics from University of Science and Technology of China, in 2013 and 2018, respectively. He is currently working as an Electronic Design Engineer at Huawei Technologies Co.,Ltd. His current interests include the design of processing in memory architecture and neural network computing acceleration.

Affiliations

Vladimir Rybalkin¹  · Chirag Sudarshan¹ · Christian Weis¹ · Jan Lappas¹ · Norbert Wehn¹ · Li Cheng²

Chirag Sudarshan
sudarshan@eit.uni-kl.de

Christian Weis
weis@eit.uni-kl.de

Jan Lappas
lappas@eit.uni-kl.de

Norbert Wehn
wehn@eit.uni-kl.de

Li Cheng
licheng56@huawei.com

¹ Technische Universität Kaiserslautern, Kaiserslautern, Germany

² Huawei Technologies Co., Ltd., Chengdu, China