Unlocking Efficiency in BNNs: Global by Local Thresholding for Analog-Based HW Accelerators

Mikail Yayla, Fabio Frustaci, *Senior Member, IEEE*, Fanny Spagnolo, *Member, IEEE*,
Jian-Jia Chen, *Senior Member, IEEE*, and Hussam Amrouch, *Member, IEEE*

***Abstract*— For accelerating Binarized Neural Networks (BNNs), analog computing-based crossbar accelerators, utilizing XNOR gates and additional interface circuits, have been proposed. Such accelerators demand a large amount of analog-to-digital converters (ADCs) and registers, resulting in expensive designs. To increase the inference efficiency, the state of the art divides the interface circuit into an Analog Path (AP), utilizing (cheap) analog comparators, and a Digital Path (DP), utilizing (expensive) ADCs and registers. During BNN execution, a certain path is selectively triggered. Ideally, as inference via AP is more efficient, it should be triggered as often as possible. However, we reveal that, unless the number of weights is very small, the AP is rarely triggered. To overcome this, we propose a novel BNN inference scheme, called Local Thresholding Approximation (LTA). It approximates the global thresholdings in BNNs by local thresholdings. This enables the use of the AP through most of the execution, which significantly increases the interface circuit efficiency. In our evaluations with two BNN architectures, using LTA reduces the area by 42x and 54x, the energy by 2.7x and 4.2x, and the latency by 3.8x and 1.15x, compared to the state-of-the-art crossbar-based BNN accelerators.**

***Index Terms*— Neural networks, error tolerance, approximate computing, hardware design.**

I. Introduction

NUMEROUS fields have benefited from the application of neural networks (NNs). However, NNs are highly resource demanding. To achieve high accuracy, they require a massive number of parameters and with them need to perform a large number multiply-accumulate (MAC) operations. While NN models become increasingly larger, low-power operation for sustainability is rapidly gaining importance, especially for resource-constrained systems, posing immense challenges in the system design.

To overcome this challenge, binarized neural networks (BNNs) have been proposed, which have binarized weights and activations. Due to this, multiplications are computed by bitwise XNOR and accumulations by popcount. The result of the popcounts are then thresholded, i.e. compared with thresholds to produce binary outputs. This reduces the memory usage, energy, and latency by a large factor, while providing high accuracy [1]. Furthermore, BNNs are also highly error tolerant, which has been studied and exploited for efficiency in several works, i.e. concerning memory usage, latency, area, and energy [2], [3], [4].

For accelerating the BNN workload, crossbar accelerators have been proposed [5], [6]. They are organized in columns, where each column consist of XNOR gates, circuits to perform the popcount (e.g. through Kirchhoff's circuit law in analog computing), and *interface circuits*. The interface circuits employ analog-to-digital conversion and accumulate intermediate results for further processing. This necessitates the use of analog-to-digital converters (ADCs), registers connected to adders for accumulation, and digital comparators. Specifically, ADCs are one of the most critical building blocks in crossbar accelerators. It has been reported that in crossbar accelerators, the ADCs use the most on-chip area and energy. For example, in the ISAAC accelerator, ADCs use a large portion of the tile power and tile area [7].

To avoid using the ADCs and digital components, the state-of-the-art study in [5] employs an "analog path" (AP) in the interface circuit, which only uses an analog comparator. In contrast, the "digital path" (DP) uses ADCs and other digital components. The AP and DP are shown in Fig. 1. However, the AP of the interface circuit is and can only be used for a very small number of weights due to technological limitations and due to inherent variations of the analog signals, e.g. nonidealities or process variation, IR-drop, sneak paths, finite wire resistance, and other sources of noise [8], [9], [10]. Specifically, in [5], the AP is only used up to case in which 64 analog signal states need to be distinguished. This means that the AP is *not used* in common BNN architectures, which generally require a large number of weights β per neuron (see e.g. Tab. V). Therefore, the DP, with the ADCs and digital components, is used in every crossbar invocation in

Manuscript received 9 May 2023; revised 10 August 2023 and 5 September 2023; accepted 8 September 2023. Date of publication 14 September 2023; date of current version 29 December 2023. This work was supported in part by the Deutsche Forschungsgemeinschaft (DFG) through OneMemory Project under Grant 405422836 and through ACCROSS Project under Grant 428566201; in part by the Collaborative Research Center SFB 876 "Providing Information by Resource-Constrained Analysis" under Project 124020371 (subproject A1); in part by the Federal Ministry of Education and Research of Germany and the state of North Rhine-Westphalia as part of the Lamarr-Institute for ML and AI under Grant LAMARR22B; and in part by PON Ricerca Innovazione-MUR through Ministero dell–Universitã e della Ricerca, Italian Government, under Grant 062_R24_INNOVAZIONE. This article was recommended by Guest Editor C. Lammie. *(Corresponding authors: Mikail Yayla; Hussam Amrouch.)*

Mikail Yayla and Jian-Jia Chen are with the Design Automation for Embedded Systems Group, TU Dortmund University, 44227 Dortmund, Germany, and also with the Lamarr Institute for Machine Learning and Artificial Intelligence, 44227 Dortmund, Germany (e-mail: mikail.yayla@udo.edu; jian-jia.chen@udo.edu).

Fabio Frustaci and Fanny Spagnolo are with the DIMES Department, University of Calabria, 87036 Arcavacata, Italy (e-mail: f.frustaci@dimes.unical.it; f.spagnolo@dimes.unical.it).

Hussam Amrouch is with the Chair of AI Processor Design, Technical University of Munich (TUM), 80333 Munich, Germany, and also with the Munich Institute of Robotics and Machine Intelligence (MIRMI), 80992 Munich, Germany (e-mail: amrouch@tum.de).

Color versions of one or more figures in this article are available at https://doi.org/10.1109/JETCAS.2023.3315561.

Digital Object Identifier 10.1109/JETCAS.2023.3315561

BNN inference, causing high energy usage and high latency. Furthermore, each column in the crossbar needs one interface circuit, causing high area usage. For example, with 64 crossbar columns, 64 interface circuits with both APs and DPs are needed.

Without the DP of the interface circuits, only analog comparators would be needed for the BNN operations. The elimination of the DP would lead to breakthroughs in small area usage, low energy consumption, and low latency inference in BNN accelerators.

However, without the DP, the thresholdings in BNNs cannot be performed with the full result of the popcount, referred to as "global thresholding". Instead, when using only the AP, the global thresholdings have to be approximated by "local thresholdings" using the local popcounts in each crossbar column. To enable this, thresholds for the local thresholdings have to be derived, while the results of the local thresholdings have to be combined to reach an approximate global solution. Despite the high error tolerance of BNNs, this way of approximation may cause high accuracy drop, which necessitates the evaluation of the trade-offs between interface circuit efficiency and BNN inference accuracy. To the best of our knowledge, using local thresholdings for approximating the global thresholdings has not been explored yet, and no method or evaluations exist.

Concisely, our key focus is to propose methods for approximating the global thresholdings by local thresholdings, with the goal of designing highly efficient interface circuits in crossbar accelerators for BNNs.

Our contributions are as follows:

- We present a novel computing scheme for BNNs in Sec. IV, which approximates the global thresholdings by combining the results of local thresholdings, called local thresholding approximation (LTA). To tolerate the approximations of LTA, we propose to train the BNNs with approximations.
- We propose an efficient interface circuit design for the LTA and show that it needs less resources than the state of the art. Since LTA is a novel computing scheme, we propose tailored BNN workload mapping strategies. Both contributions are in Sec. V.
- In the experiments in Sec. VI, we reveal the impact of using the LTA in BNNs and show that using the LTA may lead to significant accuracy degradation if no countermeasures are employed. We then demonstrate that when training with the approximations from the LTA, we consistently achieve high accuracy even under noise. We further show that for two BNN architectures using the LTA with the efficient interface circuit reduces the area by 42× and 54×, the energy by 2.7× and 4.2×, and the latency by 3.8× and 1.15×, compared to state-of-the-art analog computing based BNN accelerators. The tools for simulating our LTA approach are available in https://github.com/myay/LTA-BNN.

II. Related Work

For computing the BNN workload in the digital domain, typically CMOS-based XNOR gates, popcount units, and other digital components are used. Especially the popcount unit uses a large amount of resources (e.g., [11], the performance of BNN acceleration HW is measured by XNOR–popcount operations per second). This popcount unit is completely removed in analog computing based accelerators.

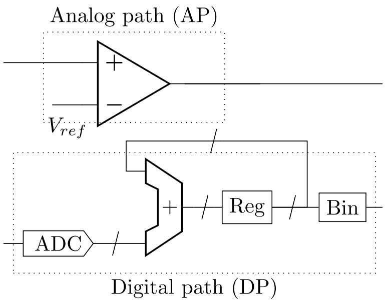


Fig. 1. Analog path (AP) and digital path (DP) of one interface circuit in [5]. Reg: Registers, Bin: Digital comparator.

In the analog domain, currents that come out of the XNOR gates are summed by employing Kirchhoff's current law. For accelerating BNN workloads in the analog domain, crossbars from different memory technologies have been evaluated. In [5], Ferroelectric Field Effect Transistor (FeFET) technology is used to build a BNN accelerator from crossbar arrays. They use standard circuit designs and show that drastic efficiency benefits due to the use of FeFET can be achieved. The study in [6] also works on FeFET-based BNN accelerators and focus on different architecture configurations to maximize efficiency.

However, to enable computing in the analog domain, analog signals need to be converted into the digital domain, for which ADCs are typically used in the interface circuit. Depending on the used technology and implementation of the ADC and popcount unit, an ADC may even require more resources compared to a popcount unit. To alleviate the resource demand by ADCs, in [12], the ADCs are replaced by 1-bit sense amplifiers and perform retraining for the loss of representation ability for general NNs. By doing this, they reduce the partial results of each crossbar column to 1 bit. The study in [13] proposes ADC-free execution at the cost of approximations. However, their study does not account for the fact that ADCs may still be required when dealing with large workload dimensions.

In this work we focus on BNNs and exploit their error tolerance to avoid ADC usage. For BNNs, the tolerance to approximations or errors has been studied in several works. In [2], the capability of BNNs to tolerate errors is demonstrated. The work in [14] proposes a margin-based method to achieve high tolerance of BNNs for errors in approximate memories, while other studies exploit the tolerance for efficient BNN inference (e.g. in [3], [4]).

In summary, the tolerance of BNNs to approximations or errors has been widely studied. However, efficient interface circuit designs tailored for BNNs, avoiding ADC usage have not received much attention yet. In order to design highly efficient interface circuits, we propose the LTA, a novel computation scheme for BNNs based on local and global thresholdings, which enables the use of highly efficient

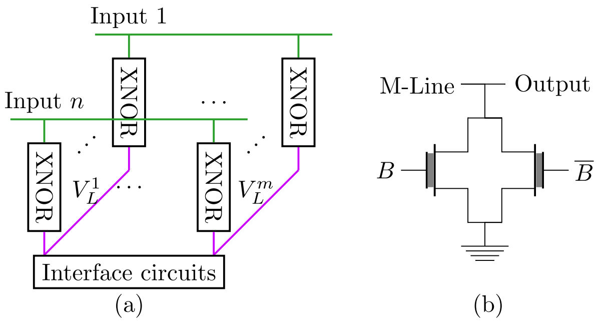


Fig. 2. (a): Crossbar with interface circuit. A possible realization of an interface circuit is shown in Fig. 1. The voltages $V_L^1, \dots, V_L^m$ or the currents are passed to the interface circuits. (b): Realization of an XNOR gate from FeFET transistors.

interface circuit with the goal of ADC-less computations at the cost of small accuracy degradation.

III. Background

We present the basics of BNNs in Sec. III-A, the description of crossbar accelerators in Sec. III-B, and the problem definition in Sec. III-C.

A. Binarized Neural Networks

We assume for a certain layer a weight matrix $\mathbf{W}$ with dimensions ($\alpha \times \beta$), where α is the number neurons and β the number of weights of a neuron. The input matrix has dimensions ($\gamma \times \delta$), where $\beta = \gamma$ and δ is the number of convolution windows in the input. We leave out any layer indices for brevity. Every convolution of a conventional NN can be mapped to this matrix notation.

In BNNs, the weights and activations are binarized. The output of a BNN layer can be computed with

$$2 * popcount(XNOR(\mathbf{W}, \mathbf{X})) - \#bits > \mathbf{T}, \qquad (1)$$

where $XNOR(\mathbf{W}, \mathbf{X})$ computes the XNOR of the rows in $\mathbf{W}$ with the columns in $\mathbf{X}$ (analogue to matrix multiplication), *popcount* counts the number of set bits in the XNOR result, $\#bits$ is the number of bits in the XNOR operands, and $\mathbf{T}$ is a vector of learnable threshold parameters, with one entry for each neuron. The thresholds are computed with the batch normalization parameters, i.e. $T = \mu - \frac{\sigma}{\psi}\eta$, where each neuron has a mean μ and a standard deviation σ over the result of the left side of Eq. (1), and ψ and η are learnable paramaters (details about the batch normalization paramters can be found in [1] and [15]). Finally, the comparisons against the thresholds produce binary values. In this work, we focus on convolution and fully connected layers of BNNs with binary inputs and binary activation in the outputs, which use the majority of the energy and execution time.

B. BNN Crossbar Accelerators

Crossbar accelerators for BNNs use XNOR gates, which store binary weights and process binary inputs. The crossbars are organized with m columns and n XNOR gates per column, i.e. they have size ($m \times n$), which determines the workload they can process. For mapping the workloads in matrix notation of the weights $\alpha \times \beta$ and inputs $\beta \times \delta$ (as described in Sec. III-A) to one crossbar, multiple strategies exist with different trade-offs regarding energy, area, and latency [5], [6].

A high-level overview of an analog computing based crossbar for BNNs is shown in Fig. 2(a). The input bits (1 to n) are applied to the input lines (in green) to the XNOR gates. The XNOR gates are programmed to store the binary weights (the circuits for programming are omitted).

The XNOR gates can be built from standard CMOS or other emerging technologies, such as FeFET, which promise highly efficient inference [5], [6]. FeFET-based XNOR gates are built by coupling two FeFET transistors together, as shown Fig. 2(b). The binary weight is stored in a complementary manner in both transistors. Depending on the input signal B and its complement $\overline{B}$, the output will either be logic "1" or "0". In practise, the match line (M-Line) is charged to high, and when there is a match between the input and the stored value, both transistors will be off and no conducting path is formed, leading to a logic "1". Only when there is a mismatch, a conducting path is formed, which causes the voltage at the output line to drop, returning logic "0".

With FeFET-based XNOR gates, the popcount can be computed in the analog domain, i.e. the output currents of all XNOR gates are summed by employing Kirchhoff's circuit law. The resulting m currents are passed to the interface circuits for further processing.

The interface circuits consist of analog comparators, ADCs, and also digital components, such as accumulators, registers, and digital comparators, see Fig. 1.

C. Problem Definition

In the state of the art (SOTA) by Chen et al. [5], one crossbar column has one interface circuit, each consisting of an analog comparator, an ADC, an adder, registers, and a digital comparator, illustrated in Fig. 1. In their design, m interface circuits are needed for a crossbar of size ($m \times n$). To avoid using the ADCs and other digital components, their approach employs an "analog path" (AP) in the interface circuit, which only uses analog comparators. In contrast, the "digital path" (DP) uses ADCs and other digital components.

However, the AP is only used when the number of weights β per neuron is very small, i.e. $\beta \leq n$, which is rarely the case (in [5], n is only 64). High-performing BNN architectures actually use a large β, and the size of β of a layer depends on the number of neurons in the previous layer. For example, in our VGG-based BNNs, β is large in every case, as shown in Tab. V. Therefore, the DP is used in every crossbar invocation, causing high energy usage and high latency. Further, the interface circuit is the most area demanding part of the crossbar accelerator, mainly due to the ADCs and registers, while the other components also cannot be neglected.

1) Problem Definition: Given a trained BNN with high accuracy and a crossbar accelerator, as described in Secs. III-A and III-B respectively, in this work, *we focus on the problem of reducing the complexity of interface circuits in BNN crossbar accelerators*, to achieve inference using small area, low energy, and low latency.

Next, we present our novel method, which enables us to use the AP through most of the execution (i.e. for $\beta \leq mn$) and needs only one interface circuit for all crossbar columns (instead of m interface circuits in the SOTA), at the cost of approximations.

IV. Local Thresholding Approximation

We describe our method, the local thresholding approximation (LTA) in Section IV-A. Due to the approximations in LTA, errors occur in the computations. We discuss countermeasures in Section IV-B.

A. LTA Execution

We consider that the weights and activations are binary values in $\{-1, 1\}$ to denote XNOR as multiplication and popcount as summation, without loss of generality. The weights of one neuron (a certain row in $\mathbf{W}$) are described as $W = (w_1, w_2, \ldots, w_\beta)$, with $w_j \in \{-1, 1\}$ and with β as the number of weights. The input (a column in $\mathbf{X}$) is denoted as $X = (x_0, x_1, \ldots, x_\beta)$ with $x_j \in \{-1, 1\}$. We assume that the layers have the following structure, without any operations inbetween: Convolution, batch norm, binary activation. With this assumptions in BNNs, the activations are computed by

$$a = \mathbf{1}[\sum_{i=1}^{\beta} w_i x_i \geq T]. \tag{2}$$

T is the threshold for the neuron in W and $\mathbf{1}[predicate]$ is a modified Iverson bracket, which returns 1 if the condition $predicate$ holds and -1 otherwise. The computation in Eq.(2) is precise, i.e. without any errors. To perform precise computations, m interface circuits with APs and DPs are needed, as described in Sec. III-C.

To use the AP and interface circuits with less complexity, *we propose to employ local thresholdings using subsequent samples of size n, and combine the results of local thresholdings with a majority function to obtain an approximate global result*. By this, the computation result of a crossbar column with n XNOR gates is represented by one binary value. The majority vote of m binary values from all crossbar columns is then performed to reach a final approximate decision. This means that local thresholdings are performed to reach an approximate global thresholding result.

We call this way of computing the local thresholding approximation (LTA). In the LTA, the AP is triggered throughout most of the execution, i.e. for $\beta \leq mn$. Furthermore, only one interface circuit is needed for the crossbar, whereas state of the art by Chen et al. [5] requires m interface circuits. We focus on the interface circuit and the corresponding workload mapping in Sec. V, with a comparison to the state of the art.

For the LTA, local thresholdings in the form $\mathbf{1}[\sum_{i=1}^{n} w_i x_i \geq T^*]$ are performed, i.e. the sums are computed up to a value n, which is the number of XNOR gates in a column of a crossbar. The local thresholdings are performed with a local threshold T^*, followed by the majority vote of all local thresholdings.

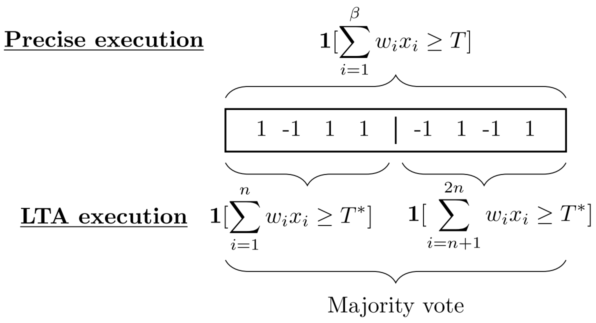

Precise execution
$\mathbf{1}[\sum_{i=1}^{\beta} w_i x_i \geq T]$
1 -1 1 1 | -1 1 -1 1
LTA execution
$\mathbf{1}[\sum_{i=1}^{n} w_i x_i \geq T^*]$
$\mathbf{1}[\sum_{i=n+1}^{2n} w_i x_i \geq T^*]$
Majority vote

Fig. 3. Precise and LTA execution in BNNs for $\beta = 8$, $n = 4$.

The LTA is defined as:

$$\begin{aligned}\mathbf{1}[\sum_{i=1}^{\beta} w_i x_i \geq T]& \\ &\approx \langle a_1, a_2, \ldots, a_N \rangle \\ &= \langle \mathbf{1}[\sum_{i=1}^{n} w_i x_i \geq T^*], \mathbf{1}[\sum_{i=n+1}^{2n} w_i x_i \geq T^*], \ldots, \\ &\quad \mathbf{1}[\sum_{i=(N-1)n+1}^{\beta} w_i x_i \geq T^*_{last}] \rangle.\end{aligned} \tag{3}$$

The majority is denoted as $\langle a_1, \ldots, a_N \rangle = Majority(a_1, \ldots, a_N)$. The threshold T^* for the local thresholdings except the last is acquired by dividing the global T by the number of local comparisons $N = \left\lceil \frac{\beta}{n} \right\rceil$. The threshold T^*_{last} for the last window (which may have smaller window size than other windows) is derived by scaling T^* according to the size of the rest $\beta - (N-1) \cdot n$. The formulas for deriving the thresholds in Eq. (3) are

$$T^* = round\left(\frac{T}{N}\right), \quad T^*_{last} = round\left(T^*\left(\frac{\beta}{n} - (N-1)\right)\right), \tag{4}$$

where the $round$ function rounds to the nearest integer and ties are broken by rounding up. An example for the LTA with $n = 4$, $\beta = 8$, and $N = 2$ is shown in Fig. 3.

The number of local comparisons N depends on n, which cannot be chosen to be arbitrarily high due to technological limitations of crossbars. When n is large, N is small, since $N = \left\lceil \frac{\beta}{n} \right\rceil$. When n becomes smaller, the global thresholdings are increasingly based on approximations from local thresholdings, which makes the approximations less precise.

The LTA execution is approximate, and the trade-offs concerning level of approximation and inference accuracy have to be evaluated (in Sec. VI-B, we will evaluate this trade-off). To introduce a metric which compares the results of the computations, and not only accuracy, we define the ratio R, for a layer, as the number of equal activations divided by the number of all activations ($|\mathbf{A}|$), i.e.

$$R = \sum_{a \in \mathbf{A}} \frac{1}{|\mathbf{A}|} \mathbf{1}\left[a = a_{LTA}\right], \tag{5}$$

Algorithm 1 Forward Pass for LTA Training

```
  Input: model, X
  Output: X
1 for each binarized layer do
2     $X_{copy} \leftarrow X.clone().detach()$
      // Regular execution
3     $execution(X, \text{regular})$, according to Eq. (2)
      // LTA execution
4     for every neuron_j for $j = 1, 2, \ldots, \alpha$ do
5         derive $\mu_j$,$\sigma_j$, $\psi_j$, and $\eta_j$, see Subsec. III-A,
           derived from Line 3;
6         $T_j \leftarrow \mu_j - \frac{\sigma_j}{\psi_j}\eta_j$;
7     $execution(X, \text{LTA})$, according to Eq. (3) (applied
       for each neuron);
8     $X.data \leftarrow X_{copy}.data$
9 return X
```

where a is the activation from the correct global thresholding and a_{LTA} the activation from the LTA.

B. Training With LTA

One method is to use the inherent error tolerance of BNNs to tolerate the approximations stemming from the LTA. However, the approximations may be heavily dependent upon the configurations. To make BNNs error tolerant, applying errors during the training has been suggested in the literature [2], [14], and [16].

We use the idea of error application during training to make the BNNs tolerant to the approximations of the LTA. To this end, we replace the correct activations by the approximate activations in the forward pass of the training procedure.

Our method for training with the LTA is illustrated in Alg. 1. We refer to the functions of BNNs that perform the operations of a layer (both 1D and 2D) as *execution*. During inference, the BNN layers are first executed in the regular way (i.e. convolution, batch norm, htanh, activation, Line 3), according to Eq. (2). After the regular execution is finished, the thresholds for each neuron in the layer are extracted (Line 6), from which the thresholds for the windows in the LTA can be computed, see Eq. (4). Then the BNN operations are performed with the LTA according to Eq. (3) (Line 7). After the LTA, the tensor of the correct activations is replaced with the tensor of the (approximate) LTA activations (Line 8).

In this method, the LTA operations are removed from the computation graph. They are not considered during the backpropagation, due to the copy and detach in Line 2. Furthermore, only the values of **X** are replaced by the approximated values $\mathbf{X}_{copy}$, see Line 8. This means that the training is performed with the approximations and that the weights and batch norm parameters of the layer in Line 3 are adapted in the backpropagation based on the LTA.

V. Data Flow, Interface Circuit and Workload Mapping for LTA

In this section, first describe the input data flow needed for the LTA in Sec. V-A. We then describe our interface circuit design for the LTA in Sec. V-B. After that, we explain general workload mapping strategies in Sec. V-C and present our mapping strategy for the LTA with a comparison to the state of the art (SOTA) in Sec. V-D.

A. Input Data Flow in LTA

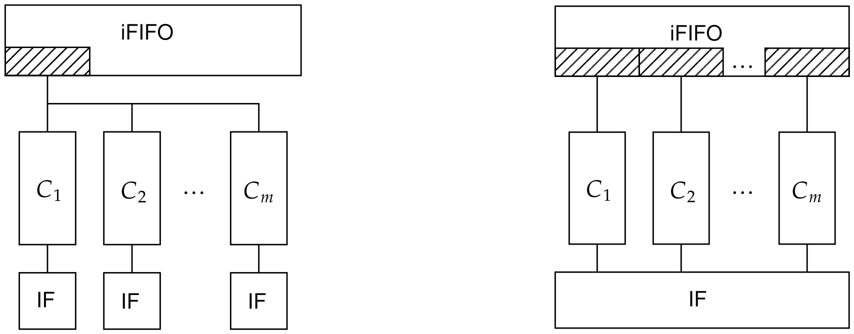
Our LTA approach requires a different way to apply inputs (i.e. types of input data flows) compared to the baseline in [5]. In the following, we consider two ways of input data flow to an accelerator with sub-components, such as crossbar columns in BNN accelerators in our case. The two types of input data flows have been explained by Kung in 1982 [17], in the context of systolic arrays: (1) One input is broadcasted to all columns (e.g. see design “B1” in [17]), which is a well-known method in modern NN accelerators and is also applied in [5]. (2) Multiple (different) inputs are applied to different columns, e.g. the first input is applied to the first column, the second input to the second column, etc. (see the designs “F” in [17]). Our LTA method requires the second type of input data flow.

Fig. 4 illustrates the input data flow type required by the LTA execution and the input data flow required by the baseline for comparison. To move the input data to the accelerator, FIFOs are used, which are called iFIFO here. The system architecture regarding the FIFOs is inspired by [18]. In the baseline case in Fig. 4(a), one input (striped area) is accessed from the iFIFO. Then, the input is broadcasted to all columns. In the LTA case in Fig. 4(b), m different inputs (striped areas in the iFIFO) are accessed from the iFIFO. Then, the inputs are moved into the crossbar columns.

In both cases in Fig. 4(a) and in Fig. 4(b), techniques and corresponding circuits need to be used to move the data into the crossbar columns, while considering the design trade-offs. For example, for both cases, the inputs can be moved into the crossbar array in an iterating manner over all columns and inputs can be applied to multiple columns in parallel. The difference between (a) and (b) with respect to the iFIFOs is that in (a) a single input needs to be accessed and supplied to the columns, while in (b), m inputs need to be accessed. If inputs are applied to multiple columns in parallel, then in the case of (a), the iFIFO needs one output port (since one input is accessed), while in (b) multiple output ports are needed (since multiple different inputs are accessed).

B. Interface Circuit for LTA

Our interface circuit design for an entire crossbar with m columns and n XNOR gates computing with the LTA is shown in Fig. 5. See also Fig. 4 for the overall architecture, i.e. where the interface circuits are and that m interface circuits are needed in the baseline, while only one interface circuit is needed for the entire crossbar for the LTA. The interface circuit in Fig. 5 has m incoming summed currents from m crossbar columns. These currents are converted into voltages (e.g. V_L^1) by the resistors R_1 and then amplified with the operational amplifiers (opamps) A_1. The resulting voltages are compared to thresholds in the analog comparators A_2. The reference voltages V_{ref} are derived by dividing the original threshold, see Eq. (4). The outputs of A_2 are voltage levels, which are



(a) Input data flow for baseline.

(b) Input data flow for LTA.

Fig. 4. Comparison of input data flow between the (a) baseline and (b) the LTA execution. iFIFO: FIFO for input data. $C_1, \dots C_m$: Crossbar columns. IF: Interface circuit.

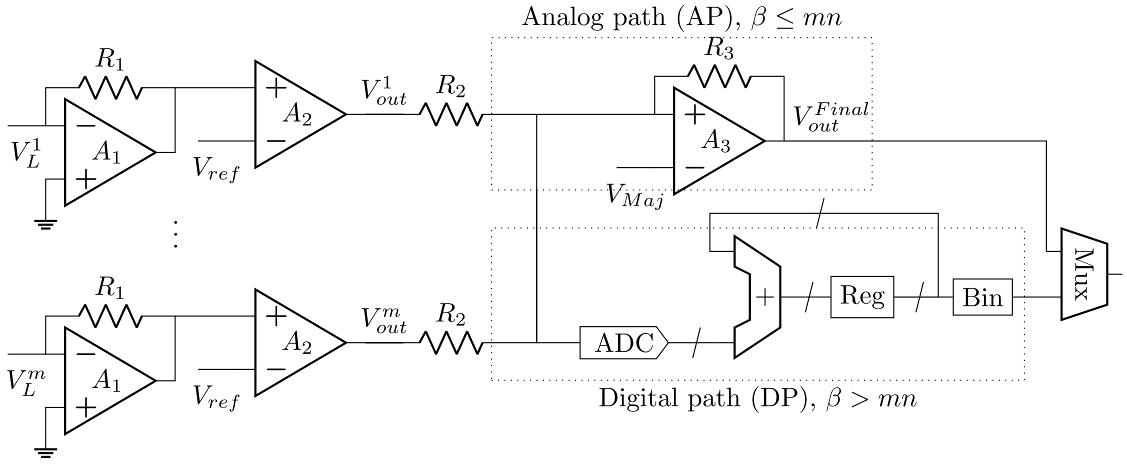


Fig. 5. Our interface circuit design for our proposed LTA method. The voltages V_L^1 and V_L^m in are input voltages from Fig. 2(a).

converted to currents by the resistors R_2. In the analog path (AP) of the circuit, these currents are summed by Kirchhoff's circuit law, converted back to a voltage by R_3, and then a majority vote is performed with the analog comparator A_3, by which the final output V_{out}^{final} is obtained.

The AP is employed in our LTA interface circuit design when the workload has size $\beta \leq mn$. If the workload is larger than that, i.e. $\beta > mn$, then the digital path (DP) is used. In the DP, the sum of currents from the opamps A_2 are converted into the digital domain using an ADC. Then, accumulations are performed, whose results are stored in registers. When the accumulation is finished, a binary comparison is performed and the final result is obtained (details are described in Sec. V-D).

Note that, for the LTA, the AP is used when $\beta \leq mn$ and only *one* interface circuit for the entire crossbar is needed. In comparison, the AP is used only when $\beta \leq n$ in the SOTA [5], and m interface circuits are needed for the entire crossbar, which all need an ADC, an accumulator, registers, and binarization logic.

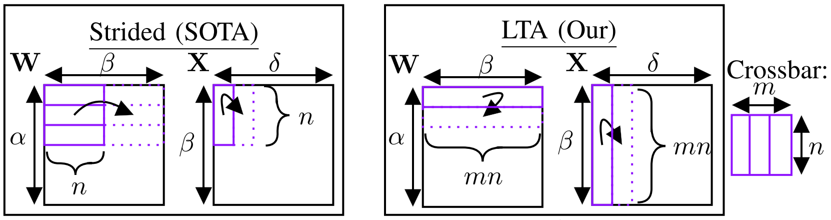
The specifications of the interface circuit for the LTA and a comparison to the state of the art are summarized in Tab. I, and the table for explaining the notations is in Tab. II. The ADC resolution for our LTA is dependent on the number of columns m, and the number of bits is calculated by $\lfloor \log_2(m) \rfloor + 1$, since at maximum, there are $m+1$ analog states (currents) that need to be distinguished by the ADC for m columns. For the SOTA, the ADC resolution is dependent on the number of XNOR gates n, and the number of bits is derived by $\lfloor \log_2(n) \rfloor + 1$, since $n+1$ analog states need to be distinguished in the same way as above.

Furthermore, note that with larger crossbar sizes (e.g. doubling n or m), the LTA interface circuit gets more efficient, as there is a higher chance that the analog path will always be triggered, since in the LTA case, the condition for using the analog path is $\beta \leq mn$, whereas in the SOTA it is $\beta \leq n$. In the LTA, both n and m decide whether the analog path is used, whereas in the SOTA, only n is the deciding factor. This holds true until the case that $n \geq \beta$, in which case the LTA and the SOTA would use the same area and resources. In the case of $m = 1$, the LTA also operates in the same way as the SOTA.

C. Crossbar for Inference

For the BNN workload, we employ the matrix notation using **W** and **X** as described in Sec. III-A. The crossbar computes the matrix multiplication $\mathbf{O} = \mathbf{W} \times \mathbf{X}$. The computation can be separated into two stages, *programming* and *application*. In the programming stage, the weights of neurons are programmed into the crossbar. In the application stage, the inputs are applied to the crossbar and the results of the matrix multiplication are returned.

The work in [5] applies a tile-based approach using a strided workload mapping scheme, which minimizes the number of



(a) (b)

Fig. 6. Mapping schemes. (a): SOTA [5]. (b): LTA.

reprogammings – an important issue in non-volatile-memory-based (NVM-based) crossbars. In their approach, the crossbar is programmed with a weight tile of n weights from m neurons, where each neuron occupies one column of the crossbar, as shown in Fig. 6(a) in **W**. Note that the tile may not consist of all weights of the neurons. Then, parts of the columns of **X** of size n, which are the corresponding inputs for the programmed weights, are pushed to the crossbar, such that all inputs that are possible to be processed with the programmed weights are processed. This is performed so that the programmed weights are reused without reprogramming. After all the input columns (arrow in **X** in Fig. 6(a)) are finished for the loaded weights of the current tile, the weights of the next tile are programmed into the crossbar (arrow in the **W** matrix for SOTA). This process is repeated over the columns of **X**.

However, this mapping scheme rarely uses the AP, since usually $\beta \gg n$, resulting in heavy use of the DP with a high cost to digitalize and buffer many intermediate results. For these operations, m DPs (for m crossbar columns) are employed, using m ADCs, $m\delta$ registers, and m other digital components with a large number of bits.

D. Our Workload Mapping for LTA

Consider now that a crossbar is given with the interface circuit in Sec. V-B. In the LTA, the weights of one neuron in **W** (one row in **W**) are programmed into the crossbar, as shown with mn for the loaded weights in Fig. 6(b). The weights are partitioned to the columns of the crossbar as described in Eq. (3). Each column of the crossbar is mapped to one window of Eq. (3). After programming, a column of **X** is pushed to the crossbar as input, as shown in Fig. 6(b). The crossbar is invoked and the majority vote of all m local thresholdings is performed. This is repeated with the same weights until all the columns in **X** are finished (arrow in **X** in Fig. 6(b)). When all columns in **X** are processed, the same procedure is repeated for the next neuron (arrow in **W** in Fig. 6(b)). Only the AP is applied when the weights of one neuron fit into one crossbar, i.e. when $\beta \leq mn$.

When $\beta > mn$, the DP needs to be used to digitalize and buffer the intermediate results, since the weights of one neuron do not fit into the entire crossbar. When using the DP, the crossbar is first programmed with mn weights of one neuron in **W**. Then the inputs that need to be processed with the programmed nm weights are supplied to the crossbar. The sum of m currents from m analog comparators is converted to the digital domain using the ADC. The results are accumulated in a designated register for the column of **X**. This is repeated until all columns of **X** are processed. Then, the next nm weights of one neuron are processed. After all operations for a neuron are finished, the values in the registers are binarized. The same procedure is repeated for the subsequent neurons.

In Tab. I, we summarize the equations for the interface circuit properties and equations regarding area, energy, and latency. The notation is explained in Tab. II. As mentioned above, the intermediate storage of values is only necessary, if the crossbar is too small to hold all the values ($\beta > mn$). In this case, the number of registers needed for the interface circuit of the crossbar in LTA is δ. For the width of the digital path in our LTA, the number of crossbar invocations for one neuron ($\left\lceil \frac{\beta}{mn} \right\rceil$) is multiplied by the maximum popcount value in each invocation (m), and to acquire the number of bits to represent this, the log is applied, while a 1 is added to cover the case in which the popcount result is 64 for each invocation. For the width of the digital path in the SOTA, there are $\left\lceil \frac{\beta}{n} \right\rceil$ column invocations for one neuron, where the maximum value in each invocation is n. In our LTA, only δ registers are needed in the entire interface circuit, while for the SOTA it is $m\delta$. Finally, in the LTA, the entire crossbar can be used for one neuron, therefore the number of crossbar invocations is $\delta\alpha \left\lceil \frac{\beta}{mn} \right\rceil$, whereas in the case of SOTA, since each neuron is assigned to one crossbar column, the number of invocations is $\delta \left\lceil \frac{\alpha}{m} \right\rceil \left\lceil \frac{\beta}{n} \right\rceil$.

To estimate area, energy, and latency, we add the individual parts of each subcomponent in Tab. I. For example, for area, we have the crossbar A_{cb} (which include the XNOR gates), the analog comparator with A_{acomp}, the ADC with A_{adc} and the area of the digital components A_{add} (adder), A_{reg} (register), and A_{bin} (binarizer), where the superscript is either lta or $sota$ for distinguishing. For the energy calculations, the A is replaced by E, and we use the number of crossbar invocations I^{cb} to multiply it with the energy used by each subcomponent in with sample. For latency calculations, we replace E by L and we also use I^{cb}.

Note that, when β is small compared to the crossbar size, i.e. $\beta \ll mn$, the LTA mapping above may not fully utilize the crossbar in one invocation, since one neuron always occupies the entire crossbar. For example, with 576 weights per neuron, and 4096 XNOR gates in total, the crossbar utilization is merely around 14.1% (and seven neurons could be computed in parallel). This may lead to a long latency, since the number of crossbar invocations is larger than the SOTA. To alleviate this, when $\beta \leq \frac{mn}{2}$, multiple neurons can be mapped to one crossbar, enabling parallel computation, which increases the crossbar utilization. We call this extension

TABLE I

Crossbar Interface Circuit Comparison Between LTA (This Work) and the State of the Art (SOTA) in [5], for a Crossbar Size of m Columns and n XNOR Gates Per Column. The Notations Are Described in Tab. II. The Formulas for the SOTA in [5] Can Acquired by the Following Substitutions: $\alpha = C_{out}$, $\beta = W_F H_F C_{in}$, $\delta = W_O H_O$, $S = \left\lceil \frac{\beta}{n} \right\rceil$, $m = N$, and $n = M$

Specification	This work (LTA)	State of the art (SOTA) [5]
ADC resolution	$\lfloor \log_2(m) \rfloor + 1$	$\lfloor \log_2(n) \rfloor + 1$
Width digital path	$\left\lfloor \log_2(m \left\lceil \frac{\beta}{mn} \right\rceil) \right\rfloor + 1$	$\left\lfloor \log_2(n \left\lceil \frac{\beta}{n} \right\rceil) \right\rfloor + 1$
Registers	δ	$m\delta$
Crossbar invocations	$I_{cb}^{lta} = \delta\alpha \left\lceil \frac{\beta}{mn} \right\rceil$	$I_{cb}^{sota} = \delta \left\lceil \frac{\alpha}{m} \right\rceil \left\lceil \frac{\beta}{n} \right\rceil$
Area	$A_{cb} + (m+1)A_{acomp} + A_{adc} + A_{add}^{lta} + \delta A_{reg}^{lta} + A_{bin}^{lta}$ For $\beta \leq mn$: $A_{cb} + (m+1)A_{acomp}$	$A_{cb} + m(A_{acomp} + A_{adc} + A_{add}^{sota} + \delta A_{reg}^{sota} + A_{bin}^{sota})$ For $\beta \leq n$: $A_{cb} + mA_{acomp}$
Energy	$I_{cb}^{lta}(E_{cb}^{lta} + mE_{acomp} + E_{adc} + E_{add}^{lta} + E_{reg}^{lta}) + \alpha\delta E_{bin}^{lta}$ For $\beta \leq mn$: $I_{cb}^{lta} E_{cb} + (m+1) I_{cb}^{lta} E_{acomp}$	$I_{cb}^{sota} E_{cb}^{sota} + I_{cb}^{sota} m(E_{adc} + E_{add}^{sota} + E_{reg}^{sota}) + \left\lceil \frac{\alpha}{m} \right\rceil m\delta E_{bin}^{sota}$ For $\beta \leq n$: $I_{cb}^{sota} E_{cb} + mI_{cb}^{sota} E_{acomp}$
Latency	$I_{cb}^{lta}(L_{cb} + L_{acomp} + L_{adc} + L_{add}^{lta} + L_{reg}^{lta}) + \alpha\delta L_{bin}^{lta}$ For $\beta \leq mn$: $I_{cb}^{lta} L_{cb} + 2I_{cb}^{lta} L_{acomp}$	$I_{cb}^{sota}(L_{cb} + L_{adc} + L_{add}^{sota} + L_{reg}^{sota}) + \left\lceil \frac{\alpha}{m} \right\rceil \delta L_{bin}^{sota}$ For $\beta \leq n$: $I_{cb}^{sota} L_{cb} + I_{cb}^{sota} L_{acomp}$

TABLE II

Notation for Tab. I

Variable	Definition
m	Number of columns in a crossbar
n	Number of XNOR gates per column
α	Number of neurons in a layer
β	Number of weights (of neurons) in a layer
δ	Second dimension of the input matrix
I_{cb}	Number of crossbar invocations
A, E, L	Area, energy, and latency of a component, respectively
lta	Local thresholding approximation
$sota$	State of the art
cb	Crossbar
$acomp$	Analog comparator
adc	Analog-to-digital converter
reg	Register
bin	Digital binarizer
add	Digital accumulator

LTA maximum utilization (LTA-MU). Compared to LTA, LTA-MU additionally divides the number of crossbar invocations I_{cb}^{lta} in Tab. I by the factor $f = \lfloor \frac{mn}{\beta} \rfloor$, which means f neurons are processed in parallel in one crossbar. However, when LTA-MU is used, the number of interface circuits with analog comparators in Tab. I needs to be multiplied by f. These trade-offs are evaluated and compared to the SOTA in Sec. VI. Note that when $\beta \geq \frac{mn}{2}$, LTA-MU cannot be used, since multiple neurons cannot be computed in parallel in this case. We note that using LTA-MU instead of LTA never influences the inference accuracy, because the computation are the same. The only difference is that LTA-MU exploits parallelism.

VI. Experiments

In Sec. VI-A, we present the experiment setup, in Sec. VI-B the trade-off of the LTA regarding accuracy, and in Sec. VI-C we evaluate the area energy, and latency of BNN inference with the LTA and our efficient interface circuit. Since there can be different types of noise in the circuit, especially when analog computing is employed, we perform a general noise analysis and how it can be overcome by training with the noise together with the LTA in Sec. VI-D. In Sec. VI-E, we discuss the feasibility of the LTA input data flow. In Sec. VI-F we discuss other methods to improve the LTA that we explored and in Sec. VI-G, we discuss the limitations of BNNs models that can be used with the LTA in its current form. Finally, in Sec. VI-H, we explore the consequences of different ADC bit widths in our design.

A. Experiment Setup

We evaluate following BNNs: A VGG3-based convolutional BNN, with FashionMNIST and KuzujishiMNIST, and a VGG7-based BNN with the SVHN, CIFAR10, and Imagenette (image size is scaled to 64 × 64 and an additional maxpool layer is added to the VGG7 BNN). VGG3 and VGG7 are modified versions of the VGG-architectures [19], adapted for the image sizes in the above datasets. We use moderately difficult prediction tasks, with a small convolutional BNN (VGG3) and a relatively large BNN (VGG7), which are suitably sized examples for resource constrained inference. Please note that we use the weakest variant of BNNs, with binarized weights and binarized activations, which are the hardest to train. The details of the datasets and BNN architectures are presented in Tab. III and Tab. IV respectively. The BNNs use convolutional (C) layers with size 3 × 3 , fully connected (FC) layers, maxpool (MP) with size 2 × 2, and batch normalization (BN) layers.

We use Adam for optimizing BNNs in a PyTorch-based framework. We use the modified hinge loss (MHL) with the hyperparameter $b = 128$ (see [14]) to achieve high accuracy and error tolerance by margin-maximization. The batch size is 256 for all models, except for Imagenette, where the batch size is 128. We use an initial learning rate of 10^{-3} for in all cases. We halve the learning rate every 10th epoch for Kuzujishi, Fashion, SVHN, and halve it every 50th epoch for CIFAR10 and Imagenette. For each model we train 100 epochs for Kuzujishi, Fashion, SVHN, and 200 epochs for CIFAR10 and Imagenette.

The LTA execution is not supported in PyTorch, and cannot be performed using the standard MAC engine of PyTorch. To execute the BNNs with the LTA in PyTorch, we developed our framework (https://github.com/myay/LTA-BNN) using a custom MAC library, with our own custom CUDA extensions.

TABLE III
DATASETS USED FOR EXPERIMENTS

Name	# Train	# Test	# Dim	# classes
FashionMNIST	60000	10000	(1,28,28)	10
KuzushijiMNIST	60000	10000	(1,28,28)	10
SVHN	73257	26032	(3,32,32)	10
CIFAR10	50000	10000	(3,32,32)	10
IMAGENETTE	9470	3925	(3,64,64)	10

TABLE IV
BNN ARCHITECTURES. LAYER TYPES ARE FULLY CONNECTED (FC), CONVOLUTIONAL (C), AND MAXPOOL (MP). EACH CONVOLUTIONAL LAYER IS FOLLOWED BY A BATCH NORMALIZATION LAYER, EXCEPT THE OUTPUT LAYER

Name	Architecture
VGG3	In → C64 → MP2 → C64 → MP2 → FC2048 → FC10
VGG7	In → C128 → C128 → MP2 → C256 → C256 → MP2 → C512 → C512 → MP2 → FC1024 → FC10

TABLE V
MATRIX DIMENSIONS OF THE WEIGHT MATRIX $\mathbf{W}$ AND INPUT $\mathbf{X}$

NN Architecture	Layer index	$\mathbf{W}(\alpha, \beta)$	$\mathbf{X}(\gamma, \delta)$
VGG3	1	(64, 576)	(576, 196)
	2	(2048, 3136)	(3136, 1)
VGG7	1	(128, 1152)	(1152, 1024)
	2	(256, 1152)	(1152, 256)
	3	(256, 2304)	(2304, 256)
	4	(512, 2304)	(2304, 64)
	5	(512, 4608)	(4608, 64)
	6	(1024, 8192)	(8192, 1)

The calls to nn.linear and nn.conv2D layers in PyTorch are redirected to our custom CUDA kernels, which implement the local thresholding and majority voting for the BNN layers.

When we train with the LTA, i.e. apply Alg. 1, we always train from scratch with the LTA execution. This means we always start a completely new training process when we train with the LTA. This includes the cases in which we train with a certain number of XNOR gates (Sec. VI-B) and the cases in which we perform the noise analysis (Sec. VI-D).

B. Accuracy Trade-off in LTA

In Fig. 7, we show the accuracy trade-off for the LTA. We use Eq. (4) for acquiring the local thresholds from the global thresholds and in the inference we apply the LTA computation scheme as shown in Eq. (3). In the black line, the accuracy under the LTA without any countermeasures is shown. For all datasets, we observe that with an increase of the number of XNOR gates (n), the accuracy increases. However, the accuracies fluctuate and the accuracy drops can be large, compared to the original accuracy. An explanation for this is that the majority vote is disturbed by the rest in Eq. (4) because of the layer dimensions.

To alleviate these issues, we apply the LTA-train method in Sec. IV-B. In these cases, the accuracy is high consistently, and the difference to the original accuracy becomes small without any fluctuations. For example, for $n = 64$ XNOR gates, which is a reasonable number for a crossbar, the accuracy of Kuzushiji is 89.25% (93.74% baseline), for Fashion 88.34% (90.68% baseline), for SVHN is 91.88% (92.84% baseline), and for Imagenette 72.00% (72.15% baseline). For CIFAR10, the accuracy tradeoff is larger, i.e. it is 77.85 (85.50% baseline), which reaches the maximum of 82.05% for 192 XNOR gates. The reason for the low accuracy in case of CIFAR10 is that the CNN cannot tolerate the approximations for this dataset. SVHN uses the same BNN architecture and is a less challenging dataset than CIFAR10, while having high accuracy. For CIFAR10, a more approximation tolerant CNN model needs to be chosen, to apply the LTA with higher accuracy.

In Fig. 9, we show the ratio R (see Eq. (5)) of different activations, between the traditional and the LTA execution for the Fashion dataset as an example. We observe for both layers L1 and L2, that on average, the ratio gets smaller for a higher number of XNOR gates n. This means, the larger n, the more activations in the LTA execution equal the traditional execution. This is also in line with the results in Fig. 7. Especially for L1, the higher the accuracy, the smaller the ratios. When the accuracy has a large drop, the ratio also peaks. For the other datasets and models the figure looks similar.

In the Table VII, we have provided the time for training BNNs and also for training the LTA (for training from scratch with LTA). For VGG3, the standard BNN training time (without LTA) per epoch is 5 seconds and with LTA it is 23.50 seconds on average. For VGG7 the numbers are 53.03 seconds and 599.82 seconds per epoch on average respectively. The reason for the long experiment run-times of the LTA training in our framework is the fact that we replaced the standard MAC engine in PyTorch with our own custom MAC engine, as explained in Sec. VI-A.

C. Area, Energy, and Latency

The area, energy, and latency are calculated based on the formulas in Tab. I. The formulas are composed of variables for A (area), E (energy), and L (latency). The variables are defined in Tab. II. The values for these variables (A, E, and L) are acquired for digital components by own simulations (in Cadence Genus using 28nm FDSOI technology). For the ADC and the analog comparator, the values are taken from other studies. For the latency and energy usage of the FeFET-based crossbar, we perform circuit simulations based on HSPICE. We explain the details in the following.

We assume the same crossbar configurations as in [5], i.e. it is based on FeFET technology and it has the dimensions $m = 64 \times n = 64$. In the literature, FeFET-based XNOR crossbars have been built for 64 × 64 in [5] and for 48 × 64 in [6]. Importantly, in [22], a full FeFET-based crossbar array for a commercial 28nm technology node from GlobalFoundries is demonstrated, and their design only supports activating 64 FeFET gates at once to combat effects from variations and avoid the cost of large ADCs. In order to compare the LTA to the SOTA in our manuscript, we use the size 64 × 64 for a specific example of a realistic analog-based BNN accelerator.

In our evaluations, the crossbar energy and latency for application are relevant, since the number of crossbar invocation differs among the LTA/LTA-MU and SOTA execution schemes. To estimate the energy and latency of the crossbar, we employ accurate circuit simulations using HSPICE

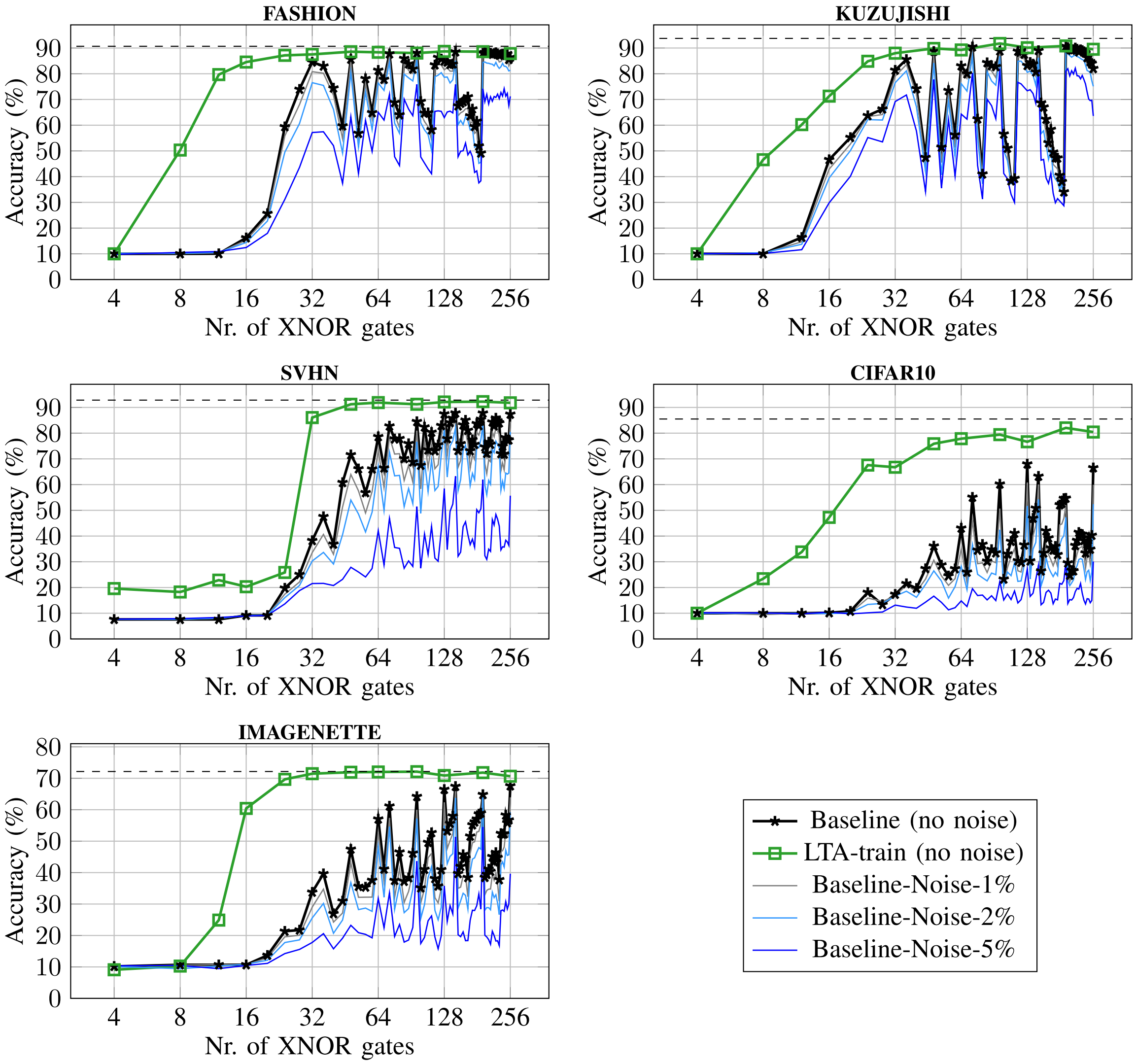
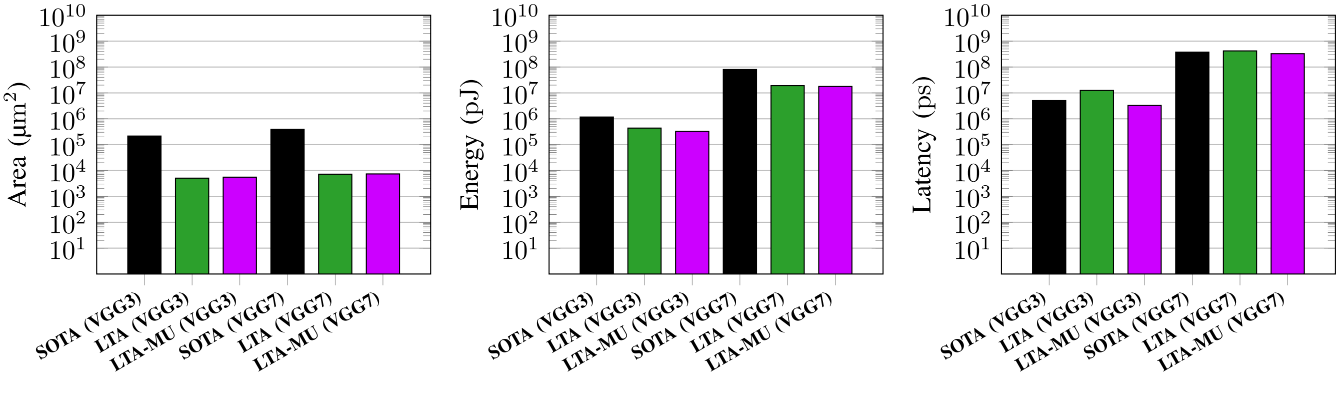


Fig. 7. Accuracy over number of XNOR gates for different datasets. In the baseline case, the BNN is executed with the LTA for varying numbers of XNOR gates, called "baseline" (the training employs only standard methods). In the "LTA-train" case, the BNN is trained with a specified number of XNOR gates. The original test accuracy is shown with the dashed line. The accuracy trade-off is explained in Sec. VI-B. Since there can be different types of noise in the analog circuit, we show the baseline BNN accuracy with a general type of noise injected alongside applying the LTA (the evaluations regarding noise are explained in Sec. VI-D).



(a) Area (b) Energy (c) Latency

Fig. 8. Area, energy, and latency comparison for the crossbar and interface circuits between the state of the art (SOTA), and our proposed method (LTA). Note that the y-axes are in log scale. For the BNN models, VGG3 and VGG7 are used, see Tab. IV.

after careful calibrations against measurements. First the industry-compact model for the FinFET technology (BSIM-CMG) is calibrated to reproduce measurement data from Intel 14nm FinFET. We calibrate the transistor, which forms the underlying FET device, against experimental measurements. The electrostatics, charge carrier transport, mobility models,

TABLE VI

Energy, Area, and Latency Configurations of the Interface Circuit'ssubcomponents, Based on the Literature (Analog Components) and Own Evaluations (i.e. in Cadence Genus Using Commercial 28nm FDSOI Technology for the Digital components). For the Digital Components, $\beta = 3136$ for VGG3, and $\beta = 8192$ for VGG7. Note That for VGG3 Under LTA, Digital Components Are Not Used. The Total Energy, Area, and Latency of the BNN Crossbar and Interface Circuit Are Calculated Based on the Values in This Table, Which Are Substituted in the Area, Energy, and Latency Formulas in Tab. I

Interface circuit parameters (28 nm technology node)				
Component	Specification	Energy (pJ/op)	Area (μm^2)	Latency (ps)
Analog Comparator	See [20]	0.163	78	74
ADC	See [21]	2.55	2000	1000
Digital path SOTA	VGG3 | VGG7	1.61 | 4.51	1282.10 | 4011.00	270
Digital path LTA	VGG7	0.223	150.9	240

TABLE VII

Avg. and (Max.-Avg., Avg.-Min.) Training Run-Times in Seconds for Ten Epochs (FashionMNIST for VGG3, CIFAR10 for VGG7). For LTA Training, the Number of XNOR Gates Is Set to $n = 64$. Hardware: Intel Core i7-8700K 3.70 GHz, 32 GB RAM, GeForce GTX 1080 8 GB

Case	Standard BNN Training	LTA training (Alg. 1)
VGG3	5.00, (0.06, 0.02)	23.50, (0.20, 0.33)
VGG7	53.03 (0.34, 0.96)	599.82, (0.99, 3.79)

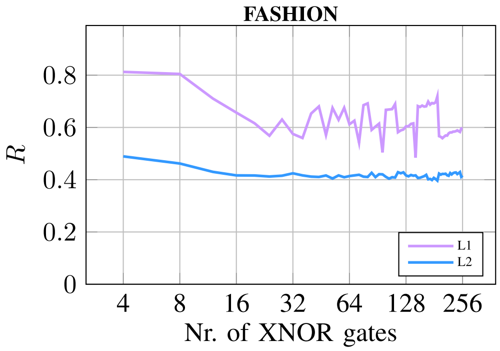


Fig. 9. The plots show the ratio of different activations (first and second layer in Fashion VGG3) between the traditional and the LTA execution, over the number of XNOR gates.

and other device parameters are carefully tuned until TCAD simulations come with an excellent agreement with the measurement data in which transfer characteristics obtained by our TCAD simulations for I_{ds}-V_{gs} under various V_{ds} biases and I_{ds}-V_{ds} under various V_{gs} biases reproduce very well the experimental measurements. Next, the high-κ dielectric in the gate-stack of the transistor is replaced with a thick (10nm) HfO2-based FE layer. The key parameters of the HfO2 layer (i.e., remnant polarization, saturation polarization, and coercive field) are also calibrated against measured Q_{FE} - V_{FE} data from a metal-ferroelectric-metal capacitor [23]. Further details on the FeFET calibration and modeling are available in our previous works [24], [25]. Then, we incorporate a physics-based model that captures the ferroelectric behaviour in the high-κ layer in the transistor gate stack. The ferroelectric parameters such as remnant polarization and coercive field are obtained from a fabricated FeCap (ferroelectric capacitor) after measuring the P-V loop. To form an XNOR logic, two Fe-FinFET devices are connected in parallel, as shown in Fig. 2(b). They always store the data in a complementary manner. Therefore, whenever an input data is applied, the device will conduct a current only in the case of mismatch (i.e., when the coming data and the stored data are different). Afterwards, we form from such an FeFinFET-based XNOR device an array and we then measure the delay, power, and energy using HSPICE. Depending how many mismatches between the input vector data and stored data, the array will conduct more or less current (i.e., the higher the mismatches level, the larger the current). We then calculate the energy and latency for the worst case in which the highest level of mismatches occurs.

Based on the above, the energy usage of the crossbar for application (without reprogramming) is 1.32 pJ per crossbar column. To calculate the energy usage of an entire crossbar, this number is multiplied by the number of used crossbar columns. The latency of one FeFET-based XNOR gate (and therefore the entire crossbar due to parallelism) is 706 ps. We do not need to incorporate the reprogramming performance of the XNOR gates, since the number of reprogrammings is the same in the LTA and the SOTA execution. The area usage of the XNOR gates is also not incorporated, since the crossbar is assumed to be the same in each technique.

To evaluate the LTA, the area, energy, and latency of the analog and digital components of the interface circuit is reported in Table VI. For the digital path we synthesized the hardware for the two highest values of $\beta = 3136$ and 8192 (for VGG3 and VGG7 respectively, such that all layers of one BNN model can be executed with one device) using Cadence Genus and is mapped to a commercial 28nm FDSOI technology. For the analog components, we have selected from the literature the ADC and the analog comparator designed in the same technological node size as we used to synthesize the digital paths (i.e. 28nm). In particular, we selected the ADC in [21] for our study because it can be inferred from the survey in [26] that it has the best area and energy tradeoff among the reported 28nm ADCs. In a similar way, we selected the analog comparator in [20] since it has the best performance among the 28nm comparators we could find in the literature.

By considering that the interface circuit of our analog BNN accelerator is composed of these analog and digital subcomponents, it enables us to compare our proposed LTA to the SOTA with up-to-date subcomponents. For calculating area/energy/latency and compare our LTA to the SOTA, we rely on the formulas in Tab. I, for which the notation is explained in Tab. II.

With the crossbar sizes $n = m = 64$ and the corresponding interface circuit data, we evaluate the area, energy, and latency of our LTA/LTA-MU computation schemes and compare them to the SOTA in Fig. 8. For the BNN models in our work, in VGG3 only the AP in Fig. 5 is used. For VGG7 the DP is only used for layers 5 and 6 (see Tab. V). Our results show that the proposed LTA technique is able to reduce the total area by a factor of 42× and 54× for VGG3 and VGG7, respectively, compared to the SOTA. Furthermore, the energy consumption is reduced by a factor of 2.7× and 4.2× for VGG3 and VGG7 compared to the SOTA, respectively. LTA does not show a reduction of latency. It is 2.5× and 1.12× higher than the SOTA, for VGG3 and VGG7 respectively. However, when the LTA-MU scheme is employed, the latency is reduced by 3.8× and 1.15× compared to the SOTA for VGG3 and VGG7, respectively. As a drawback, the LTA-MU requires up 9.2% (VGG3) and 2.2% (VGG7) more area compared to its LTA counterpart.

D. Impact of Noise on LTA

When n XNOR gates are in a crossbar column, then the analog comparator needs to be able to differentiate between n different states. However, n cannot be arbitrary large. Due to inherent variations of the analog signals an arbitrary number of different states cannot be accomodated.

There are multiple potential sources of noise in our analog hardware design. Although the sources of noise are not limited to the following cases, for explaining the concept, we focus on noise caused by variation sources due to: (1) FeFET-based crossbar columns and (2) resistance value of the resistors. (1) In the crossbar columns, the FeFET-based XNOR gates consist of FeFET devices, which are prone to errors when there are temperature fluctuations during run-time (however, in our study, we assumed no major fluctuations in operating temperature during run-time). In our previous work, we have demonstrated and investigated the impact of temperature in [3] for single FeFET devices. Since FeFET-based XNOR gates consist of two FeFET devices, the current coming out of the XNOR gates will have variations due to high temperature fluctuations as well. Due to the variations of the XNOR gate currents, the summed current also experiences variation. This in turn may lead to errors of the comparator outputs (flips from 0 to 1 or from 1 to 0) that binarize the result of one crossbar column. (2) The resistors connected to the output of the comparators (see R_2 in Fig 5) may suffer from varying resistance values as well (e.g. due to issues in fabrication or temperature), leading to current variation. The effect of the noise from points (1) and (2) above continues to propagate through the circuit, which has two paths: The analog path and the digital path. In the analog path, the result of the majority comparator may be flipped due to the current variations (i.e., the output may also flip from 0 to 1 or from 1 to 0). In the digital path, the ADC may convert the analog signal to an erroneous digital value, which will be accumulated and binarized in the digital domain. In both cases, any variations in the analog computations (e.g. in points (1) and (2) above) will affect the result of the *last binarization*. In the analog path, the output of the last analog comparator will be affected. In the digital path, the output of the digital binarizer will be affected. Therefore, we simulate the noise in the circuit by flipping the output of the last binarization with a certain error probability.

1) Noise Analysis: To conduct a general noise study, we model the noise by using the flip probabilities 1%, 2%, and 5% for the outputs of the last binarizations. Please note that we are not limited to these error rates, they are merely examples. Any other error rate can be applied in our open-source framework. When we write "Noise training" or "Noise+LTA training", we refer to the cases in which we always train from scratch with noise and also the LTA (Alg. 1). In Fig. 7, we show the plots for the accuracy results achieved by the BNNs under the noise in combination with the LTA. We observe that for small noise, the accuracy degradation is also small. The larger the noise, the higher the accuracy degradation. In Tab. VIII, we also show the result of the training together with the LTA and noise in combination for the crossbar column dimension $n = 64$. We also add the results with no noise for reference. We observe that in all cases of our experiments, although the noise causes large accuracy drops, a significant amount of accuracy can be regained by training with the noise.

E. Feasibility of LTA Input Data Flow

To show the feasibility of the input application method, we implement the two types of data flow (SOTA and LTA, as presented in Sec. V-A) in VHDL. The crossbar dimensions are $n = m = 64$. For the SOTA case, we consider that one input is broadcasted to all computing columns. For the LTA case, we assume that each column receives a different input. In both designs, the weights are set once and stay the same throughout the computations to simulate the strided move.

To focus on the dataflow, we simulate the input application to the XNOR gates of all columns. We synthesize the designs using Cadence Genus with 28nm FDSOI technology (as in the previous digital circuits). For the SOTA dataflow, 15928 μm^2 is required, along with 273.55 mW of power consumption. For the LTA dataflow, 24733 μm^2 is required, along with 360.05 mW of power consumption. Both data flows are configured for 300 ps.

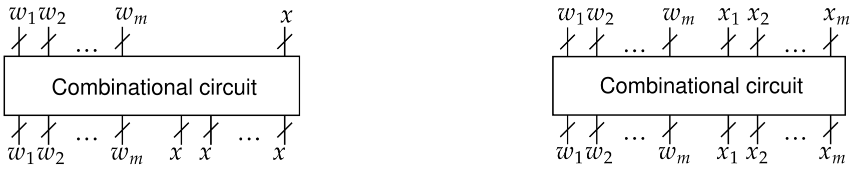
We observe that the LTA data flow uses 31.6% more power and 55.3% more area. However, note that when considering the entire crossbar accelerator, the SOTA requires 128000 μm^2 of area for the ADCs alone, while the LTA only needs 2000 μm^2 of area for the ADCs alone. This is a decrease by 64× (since in the SOTA circuit, $m = 64$ ADCs are needed for $m = 64$ computing columns, and in the LTA only 1 ADC is needed for $m = 64$ computing columns) and the mere increase of LTA by 55.3% is small compared to the 64× area reduction. Calculations in the similar scale can be performed for the energy consumption as well.

The main contribution for area and power in the LTA dataflow implementation is the increased number of wires (for area) and the different signals (power) that need to be driven, compared to the SOTA. We illustrate this by the implementation in Fig. 10 using combinational circuits consisting of wires, which are placed between the memory and the BNN crossbar. Note that memory and the crossbar could also be connected in a different way, Fig. 10 is one example. In the

TABLE VIII

Comparison of Test Accuracy (%) for the Assumed Cases in Our With Crossbar Size 64 × 64. The LTA Approximation Is Applied in Each Column Unless Specified Otherwise. In Cases of “No Noise”/“No LTA”, We Train Without Noise/Without LTA. “Noise” Followed by a Percentange Means That Noise With This Percentage Is Injected, While in “Noise Training”, We Train With the Noise. “Noise+LTA Training” Refers to the Case in Which We Inject Noise During the Training While Simultaneously Applying Alg. 1

Name	No noise, no LTA	No noise	**No noise+LTA training**	Noise 1%	**Noise+LTA training**	Noise 2%	**Noise+LTA training**	Noise 5%	**Noise+LTA training**
FashionMNIST	90.68	81.34	**88.34**	78.79	**87.89**	76.38	**87.41**	65.03	**86.90**
KuzushijiMNIST	93.74	83.05	**89.25**	80.50	**88.55**	76.16	**87.19**	64.31	**85.38**
SVHN	92.84	78.62	**91.88**	73.39	**91.01**	67.13	**90.54**	42.09	**88.14**
CIFAR10	85.50	43.07	**77.85**	35.08	**74.46**	28.23	**73.44**	14.79	**67.31**
IMAGENETTE	72.15	57.02	**72.00**	52.89	**71.69**	48.31	**70.55**	31.75	**70.37**


$w_1 w_2$ … w_m x
Combinational circuit
$w_1 w_2$ … w_m x x … x
$w_1 w_2$ … w_m x_1 x_2 … x_m
Combinational circuit
$w_1 w_2$ … w_m x_1 x_2 … x_m

(a) Input data flow circuit for SOTA.

(b) Input data flow circuit for LTA.

Fig. 10. Examples of using combinational circuits consisting of wires to connect the BNN crossbar to memories (e.g between the FIFOs in Fig. 4 and the BNN crossbar). (a) SOTA and (b) the LTA connections using combinational circuits. The w_i are the weight vectors that are composed of n bits. The weight wires transfer the weights to the ith column of the crossbar. In the SOTA, the input vector (composed of n bits) is the same for each crossbar column (it is replicated). In the LTA, each crossbar column i receives a different input x_i.

SOTA, in Fig. 10(a), a single input bitvector x (with length n) is retrieved, and then it is replicated as many times as there are parallel columns (i.e. m) in the crossbar. This means, to enable the SOTA dataflow, the single input bitvector x is replicated m times with a combinational circuit (i.e. single in, multiple out) and then the m same bitvectors of x are passed to the crossbar. In the LTA method, m different inputs are needed, labelled x_i in Fig. 10(b), which are then applied to the m crossbars (multiple in, multiple out). In summary, the increased area and power usage of the LTA are due to the number of wires and their driving with different signals.

F. Discussion of Other Methods

We have also evaluated other methods to improve the accuracy of BNNs under LTA. However, the only method that worked consistently is the LTA-train method, as explained above. We summarize these ideas here.

For each layer and every of its neurons in the BNN we use the same method: To acquire the local thresholds of a neuron, we divided the original threshold of that neuron by the number of windows N (that have size n), as shown in Eq. (4). We have also evaluated other methods for acquiring local thresholds. Instead of using the same threshold for all windows, we have attempted to tune the local thresholds. We have first incremented or decremented all the local thresholds by the same small numbers, i.e. $+1$, -1, $+2$, -2, etc., but noticed that when manually changing the local thresholds this way, the inference accuracy was severely dropping. Then, we modified the local threshold settings based on the mean popcount value within the windows of size n. For this, we considered the local thresholds T^* from Eq. (4) as a starting point and considered to manipulate them with constants. We attempted to add constants to T^* by increasing the T^* of the windows that have a larger mean, while decreasing the T^* of the windows that have a smaller mean. The thresholds were adapted based on the data in the training set. However, it did not lead to higher test accuracy, and even lead to higher drops of accuracy the higher the change in the local thresholds. This indicates that different thresholds for different windows do not lead to superior results. The reason may be the following: We observed that most popcount results in the windows of size n have similar values that are closely around the mean, and that there are not any patterns to exploit for different thresholds. We show the histograms of the popcount values for the layers in VGG3 for FashionMNIST training dataset in Fig. 11. The frequencies are acquired by recording the popcount values in windows of size $n = 64$ over the training data. We observe that the values follow a normal distribution, where most values are close to the mean of 32 for $n = 64$ (note that the y-axes are logarithmic). For other n and other datasets, we also observe the mean $\frac{n}{2}$ and similar distributions. In general, modifying the thresholds in BNNs manually to optimize certain properties has been reported to be unsuccessful in other work as well [27].

Please note that it is highly challenging to manually tune the threshold configurations, as there are thousands of neurons in our BNN architectures, which all have individual thresholds. Furthermore, the number of local thresholds is approximately one order of magnitudes larger than the total number of neurons in the BNN architectures.

We observed a similar case for the majority vote. In our framework, the majority can be shifted, meaning it can be configured that for a “1” (instead of “0”) as output, there needs to be a majority and a certain number of additional local thresholds that are “1”, for a “1” in the final output. Introducing majority vote shifts based on the oberserved mean “1”s in the local thresholds also lead to none or when the shifts become high, to poorer accuracy results.

Another idea for the LTA is to use a smaller stride than n by moving the windows such that there are overlaps, leading to more local thresholdings. In our explorations, this only alleviated the sharp drops in accuracy by a small amount (see the accuracy drops in the black plots in Fig. 7). It did not lead to an increase in accuracy for the peaks. With a smaller stride, more computations need to be performed, taking up crossbar space, which is a high cost.

Note the operation with custom thresholds, the shifted majority votes, and using different strides are all implemented in our framework as command line parameters and can be can be evaluated by the users to perform more research.

G. Discussion of Other BNN Models

To demonstrate the proof of concept of the LTA approximation, we assume in Sec. IV-A that the layers have the following structure, without any operations inbetween: Convolution, batch norm, activation. Currently, our proposed LTA cannot be used with architectures that do not comply with this assumption. One notable example are skip connections, such as those in ResNets or MobileNets. For computing the skip connection, a convolution is computed, then the result of another convolution is added to the previous result, after which an activation is applied. To enable the LTA for other structures, such as the skip connections, the BNN structure needs to be modified, which may also require the modification of the BNN acceleration hardware and the training procedure. We plan to extend the LTA for other BNN structures in future work.

H. Discussion of ADC Modifications

In Tab. VI, we observe that the ADC resource usage is significantly higher than the other components. Therefore, we discuss the expected impact of using an ADC with smaller resolution compared to our selected 8-bit ADC from [21].

The ADC we have selected is not configurable with respect to the resolution. Therefore, we rely on the data in the study in [28] (Fig. 7), where the resource consumption (area, latency, power, energy) of ADCs are shown as a function of the number of bits (3-6 bits) at the 40 nm technology node. The data for Successive Approximation Register (SAR) ADC and Flash ADC are shown in [28] (Fig. 7). SAR ADCs are more suitable for low-resource scenarios such as efficient inference in BNNs, since the Flash ADCs require a significantly larger number of comparators when compared to SAR ADCs. For SAR ADCs, we observe in [28] (Fig. 7) that when the number of bits is halved (e.g. from 6 bits to 3 bits), then the area, latency, and energy are approximately halved as well. More generally, from the data in [28] (Fig. 7), we observe a linear trend in resource usage when the number of bits is changed. To summarize, based on the study in [28], we expect that when a smaller number of bits are used in the ADC, the resource costs for the ADC will be proportionally smaller.

However, when an ADC with a smaller resolution is used for our proposed LTA method, then the analog path will be triggered in less cases. This opposes our main design idea to trigger the analog path in as many cases as possible. Recall that in the LTA method, the condition for triggering the analog path (i.e. ADC-less execution) for a layer is $\beta \leq mn$. In our study, the required ADC resolution depends on the crossbar dimension m (number of crossbar columns), i.e. the required ADC resolution is $\lfloor \log_2(m) \rfloor + 1$. Therefore, the smaller the m, the smaller the required resolution of the ADC. The drawback of this is that when m is reduced, then the analog path will be triggered in less cases, because the right side of the condition $\beta \leq mn$ for triggering the analog path will be smaller. In addition to that, a smaller number columns increases the inference latency approximately by the factor of the crossbar column reduction.

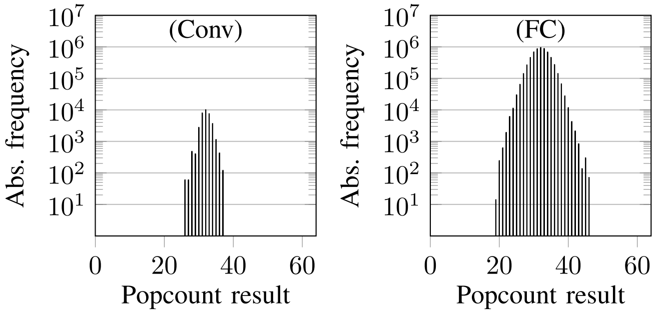


Fig. 11. Histograms of average popcount values (absolute frequency, over the training dataset) of a layer in sampling windows of size $n = 64$ for the FashionMNIST baseline BNN.

VII. Conclusion

We proposed a novel BNN inference scheme, called Local Thresholding Approximation (LTA), which approximates the global thresholdings in BNNs by local thresholdings. In BNN crossbar accelerators, this enables the use of only analog comparators through most of the execution, which significantly increases the interface circuit efficiency compared to the state of the art. However, employing the LTA without any countermeasures to the approximations results in a significant accuracy drop. To retain the original accuracy, we propose a training scheme that accounts for the degradation induced by the LTA, which consistently achieves high accuracy under approximations. Our results for two BNN models show that using the LTA reduces the area by factors of 42× and 54×, the energy by 2.7× and 4.2×, and the latency by 3.8× and 1.15×, compared to state-of-the-art crossbar-based BNN accelerators.

One possible future direction to improve the LTA accuracy is to explore methods that attempt to automatically find local thresholds and majority vote shifts in the LTA (note that in our framework, local thresholds and majority vote shifts can be manipulated conveniently). This can be done in a fine-grained manner, i.e. window or neuron based, or in a more coarse-grained manner, i.e. on the layer level. However, automatically finding or training the local thresholds and majority vote shifts introduces many additional parameters, as there can be many thousands or more neurons in a BNN, while the number of local thresholds is approximately one order of magnitude larger than the total number of neurons in the BNN. Furthermore, the optimization of thresholds also competes with the function of the batch norm layer, as the traditional (global) thresholds are derived from the batch norm parameters, without which BNNs are reported to achieve poor training performance [15].

References

[1] I. Hubara, M. Courbariaux, D. Soudry, R. El-Yaniv, and Y. Bengio, "Binarized neural networks," in *Proc. Adv. Neural Inf. Process. Syst.*, 2016, pp. 4107–4115.

[2] T. Hirtzlin et al., “Outstanding bit error tolerance of resistive RAM-based binarized neural networks,” in *Proc. IEEE Int. Conf. Artif. Intell. Circuits Syst. (AICAS)*, Mar. 2019, pp. 288–292, doi: 10.1109/AICAS.2019.8771544.

[3] M. Yayla, S. Thomann, S. Buschjäger, K. Morik, J.-J. Chen, and H. Amrouch, “Reliable binarized neural networks on unreliable beyond von-Neumann architecture,” *IEEE Trans. Circuits Syst. I, Reg. Papers*, vol. 69, no. 6, pp. 2516–2528, Jun. 2022.

[4] M.-L. Wei, M. Yayla, S.-Y. Ho, J.-J. Chen, C.-L. Yang, and H. Amrouch, “Binarized SNNs: Efficient and error-resilient spiking neural networks through binarization,” in *Proc. IEEE/ACM Int. Conf. Comput. Aided Design (ICCAD)*, Nov. 2021, pp. 1–9.

[5] X. Chen, X. Yin, M. Niemier, and X. S. Hu, “Design and optimization of FeFET-based crossbars for binary convolution neural networks,” in *Proc. Design, Autom. Test Eur. Conf. Exhib. (DATE)*, Mar. 2018, pp. 1205–1210.

[6] T. Soliman et al., “Efficient FeFET crossbar accelerator for binary neural networks,” in *Proc. IEEE 31st Int. Conf. Appl.-Specific Syst., Archit. Processors (ASAP)*, Jul. 2020, pp. 109–112.

[7] A. Shafiee et al., “ISAAC: A convolutional neural network accelerator with in-situ analog arithmetic in crossbars,” *ACM SIGARCH Comput. Archit. News*, vol. 44, no. 3, pp. 14–26, 2016.

[8] G. W. Burr et al., “Experimental demonstration and tolerancing of a large-scale neural network (165,000 synapses), using phase-change memory as the synaptic weight element,” in *IEDM Tech. Dig.*, Dec. 2014, pp. 29.5.1–29.5.4.

[9] B. Liu, H. Li, Y. Chen, X. Li, Q. Wu, and T. Huang, “Vortex: Variation-aware training for memristor X-bar,” in *Proc. 52nd ACM/EDAC/IEEE Design Autom. Conf. (DAC)*, Jun. 2015, pp. 1–6.

[10] M. Hu et al., “Dot-product engine for neuromorphic computing: Programming 1T1M crossbar to accelerate matrix-vector multiplication,” in *Proc. 53rd ACM/EDAC/IEEE Design Autom. Conf. (DAC)*, Jun. 2016, pp. 1–6.

[11] Y. Umuroglu et al., “FINN: A framework for fast, scalable binarized neural network inference,” in *Proc. ACM/SIGDA Int. Symp. Field-Program. Gate Arrays*, Feb. 2017, pp. 65–74, doi: 10.1145/3020078.3021744.

[12] U. Saxena, I. Chakraborty, and K. Roy, “Towards ADC-less compute-in-memory accelerators for energy efficient deep learning,” in *Proc. Design, Autom. Test Eur. Conf. Exhib. (DATE)*, Mar. 2022, pp. 624–627.

[13] H. Kim, Y. Jung, and L.-S. Kim, “ADC-free ReRAM-based in-situ accelerator for energy-efficient binary neural networks,” *IEEE Trans. Comput.*, early access, Nov. 25, 2022, doi: 10.1109/TC.2022.3224800.

[14] S. Buschjäger et al., “Margin-maximization in binarized neural networks for optimizing bit error tolerance,” in *Proc. Design, Autom. Test Eur. Conf. Exhib. (DATE)*, Feb. 2021, pp. 673–678.

[15] E. Sari, M. Belbahri, and V. P. Nia, “How does batch normalization help binary training?” 2019, *arXiv:1909.09139*.

[16] M. Yayla et al., “FeFET-based binarized neural networks under temperature-dependent bit errors,” *IEEE Trans. Comput.*, vol. 71, no. 7, pp. 1681–1695, Jul. 2022.

[17] H. T. Kung, “Why systolic architectures?” *Computer*, vol. 15, no. 1, pp. 37–46, Jan. 1982.

[18] Y.-H. Chen, J. Emer, and V. Sze, “Eyeriss: A spatial architecture for energy-efficient dataflow for convolutional neural networks,” in *Proc. ACM/IEEE 43rd Annu. Int. Symp. Comput. Archit. (ISCA)*, Jun. 2016, pp. 367–379.

[19] K. Simonyan and A. Zisserman, “Very deep convolutional networks for large-scale image recognition,” in *Proc. Int. Conf. Learn. Represent. (ICLR)*, 2015.

[20] A. T. Ramkaj, M. S. J. Steyaert, and F. Tavernier, “A 13.5-Gb/s 5-mV-sensitivity 26.8-ps-CLK–OUT delay triple-latch feedforward dynamic comparator in 28-nm CMOS,” *IEEE Solid-State Circuits Lett.*, vol. 2, no. 9, pp. 167–170, Sep. 2019.

[21] D.-R. Oh, K.-J. Moon, W.-M. Lim, Y.-D. Kim, E.-J. An, and S.-T. Ryu, “An 8b 1GS/s 2.55 mW SAR-flash ADC with complementary dynamic amplifiers,” in *Proc. IEEE Symp. VLSI Circuits*, Jun. 2020, pp. 1–2.

[22] T. Soliman et al., “Ultra-low power flexible precision FeFET based analog in-memory computing,” in *IEDM Tech. Dig.*, Dec. 2020, pp. 29.2.1–29.2.4.

[23] M. Jerry et al., “Ferroelectric FET analog synapse for acceleration of deep neural network training,” in *IEDM Tech. Dig.*, Dec. 2017, p. 6.

[24] A. Gupta, K. Ni, O. Prakash, X. S. Hu, and H. Amrouch, “Temperature dependence and temperature-aware sensing in ferroelectric FET,” in *Proc. IEEE Int. Rel. Phys. Symp. (IRPS)*, Apr. 2020, pp. 1–5.

[25] K. Ni, S. Thomann, O. Prakash, Z. Zhao, S. Deng, and H. Amrouch, “On the channel percolation in ferroelectric FET towards proper analog states engineering,” in *IEDM Tech. Dig.*, Dec. 2021, pp. 15.3.1–15.3.4.

[26] B. Murmann. *ADC Performance Survey 1997–2021*. Accessed: May 23, 2022. [Online]. Available: http://web.stanford.edu/~murmann/adcsurvey.html

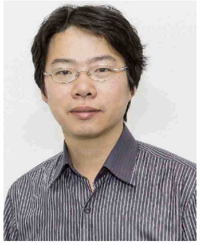
[27] S. Buschjäger et al., “Towards explainable bit error tolerance of resistive ram-based binarized neural networks,” 2020, *arXiv:2002.00909*.

[28] S. Yu, H. Jiang, S. Huang, X. Peng, and A. Lu, “Compute-in-memory chips for deep learning: Recent trends and prospects,” *IEEE Circuits Syst. Mag.*, vol. 21, no. 3, pp. 31–56, 3rd Quart., 2021.

Mikail Yayla is currently pursuing the Ph.D. degree with the Informatics Chair “Design Automation for Embedded Systems,” Technical University of Dortmund, under the supervision of Prof. Jian-Jia Chen. He has published in major EDA conferences and journals, including DATE, ICCAD, DAC, IEEE TRANSACTIONS ON COMPUTERS, and IEEE TRANSACTIONS ON CIRCUITS AND SYSTEMS—I: REGULAR PAPERS. His research interests include robust and efficient machine learning for emerging resource-constrained systems. He has one Best Paper Nomination at DATE’21.

Fabio Frustaci (Senior Member, IEEE) received the M.S. and Ph.D. degrees in electronic engineering from the University Mediterranea of Reggio Calabria, Italy, in 2003 and 2007, respectively. In 2006, he was a Visiting Scholar at the ECE Department, University of Rochester, Rochester, NY, USA. From 2011 to 2013, he was a Visiting Researcher at the EECS Department, University of Michigan, Ann Arbor, MI, USA. He is an Associate Professor with the Computer Science, Electronics, Modeling and Systems Department, University of Calabria, Rende, Italy. He has authored more than 60 articles in the field of VLSI design. Currently, he is an Editorial Board Member of *Microelectronics Journal*. His research interests include low-power and high-performance VLSI circuits, design techniques for emerging technologies, reconfigurable architectures, and embedded systems.

Fanny Spagnolo (Member, IEEE) was born in Belvedere Marittimo, Cosenza, Italy, in April 1991. She received the master’s degree in electronics engineering and the Ph.D. degree in information and communication technologies from the University of Calabria, Italy, in 2016 and 2019, respectively. She is currently appointed as an Assistant Professor with the University of Calabria. She has coauthored more than 30 articles in the field of VLSI design. Her research interests include VLSI architectures for image processing, high-performance reconfigurable circuits, embedded systems design, emerging technologies, and approximate computing techniques for low-power deep neural networks. In June 2016, she won a research grant funded by the Department of Informatics, Modeling, Electronics and System Engineering, University of Calabria.



Jian-Jia Chen (Senior Member, IEEE) received the B.S. degree from the Department of Chemistry, National Taiwan University, Taiwan, in 2001, and the Ph.D. degree from the Department of Computer Science and Information Engineering, National Taiwan University, in 2006. Between January 2008 and April 2010, he was a Postdoctoral Researcher at ETH Zürich, Switzerland. He was a Junior Professor at the Department of Informatics, Karlsruhe Institute of Technology (KIT), Germany, from May 2010 to March 2014. He is currently a Professor at the Department of Informatics, TU Dortmund University, Germany. His research interests include real-time systems, embedded systems, energy-efficient scheduling, power-aware designs, temperature-aware scheduling, and distributed computing. He received the European Research Council (ERC) Consolidator Award in 2019. He received more than ten best paper awards and outstanding paper awards and has been involved in technical committees at many international conferences.

Hussam Amrouch (Member, IEEE) received the Ph.D. degree (summa cum laude) from the Karlsruhe Institute of Technology (KIT) in 2015. He is currently a Professor and heading the Chair of AI Processor Design, Technical University of Munich (TUM), Munich, Germany. He is also with the Munich Institute of Robotics and Machine Intelligence (MIRMI), Munich. Furthermore, he is the Head of the Semiconductor Test and Reliability (STAR), University of Stuttgart, Germany. Prior to that, he was a Research Group Leader at the KIT, where he was leading the research efforts in building dependable embedded systems. He has more than 220 publications in multidisciplinary research areas (including more than 90 journals) across the entire computing stack, starting from semiconductor physics to circuit design all the way up to computer-aided design and computer architecture. His research in HW security and reliability has been funded by the German Research Foundation (DFG), Advantest Corporation, and the U.S. Office of Naval Research (ONR). His main research interests are design for reliability and testing from device physics to systems, machine learning for CAD, HW security, approximate computing, and emerging technologies with a special focus on ferroelectric devices. He holds eight HiPEAC paper awards and three best paper nominations at top EDA conferences, such as DAC'16, DAC'17, and DATE'17, for his work on reliability. He has served in the technical program committees of many major EDA conferences such as DAC, ASP-DAC, and ICCAD, and as a reviewer for many top journals like *Nature Electronics*, IEEE TRANSACTIONS ON ELECTRON DEVICES, IEEE TRANSACTIONS ON CIRCUITS AND SYSTEMS—I: REGULAR PAPERS, IEEE TRANSACTIONS ON VERY LARGE SCALE INTEGRATION (VLSI) SYSTEMS, IEEE TRANSACTIONS ON COMPUTER-AIDED DESIGN OF INTEGRATED CIRCUITS AND SYSTEMS, and IEEE TRANSACTIONS ON COMPUTERS. He currently serves as an Editor for *Scientific Reports* journal (Nature).