Error Resilient In-Memory Computing Architecture for CNN Inference on the Edge

Marco Rios
Embedded Systems Laboratory, École Polytechnique Fédérale de Lausanne (EPFL), Switzerland

Flavio Ponzina
Embedded Systems Laboratory, École Polytechnique Fédérale de Lausanne (EPFL), Switzerland

Giovanni Ansaloni
Embedded Systems Laboratory, École Polytechnique Fédérale de Lausanne (EPFL), Switzerland

Alexandre Levisse
Embedded Systems Laboratory, École Polytechnique Fédérale de Lausanne (EPFL), Switzerland

David Atienza
Embedded Systems Laboratory, École Polytechnique Fédérale de Lausanne (EPFL), Switzerland

ABSTRACT

The growing popularity of edge computing has fostered the development of diverse solutions to support Artificial Intelligence (AI) in energy-constrained devices. Nonetheless, comparatively few efforts have focused on the resiliency exhibited by AI workloads (such as Convolutional Neural Networks, CNNs) as an avenue towards increasing their run-time efficiency, and even fewer have proposed strategies to increase such resiliency. We herein address this challenge in the context of Bit-line Computing architectures, an embodiment of the in-memory computing paradigm tailored towards CNN applications. We show that little additional hardware is required to add highly effective error detection and mitigation in such platforms. In turn, our proposed scheme can cope with high error rates when performing memory accesses with no impact on CNNs accuracy, allowing for very aggressive voltage scaling. Complementary, we also show that CNN resiliency can be increased by algorithmic optimizations in addition to architectural ones, adopting a combined ensembling and pruning strategy that increases robustness while not inflating workload requirements. Experiments on different quantized CNN models reveal that our combined hardware/software approach enables the supply voltage to be reduced to just 650mV, decreasing the energy per inference up to 51.3%, without affecting the baseline CNN classification accuracy.

CCS CONCEPTS

• **Computing methodologies** → **Machine learning**; • **Computer systems organization** → **Embedded systems**.

KEYWORDS

In-Memory Computing, Fault Tolerant Architectures, Deep Neural Networks.

GLSVLSI '22, June 6–8, 2022, Irvine, CA, USA

ACM ISBN 978-1-4503-9322-5/22/06…$15.00
https://doi.org/10.1145/3526241.3530351

ACM Reference Format:
Marco Rios, Flavio Ponzina, Giovanni Ansaloni, Alexandre Levisse, and David Atienza. 2022. Error Resilient In-Memory Computing Architecture for CNN Inference on the Edge. In *Proceedings of the Great Lakes Symposium on VLSI 2022 (GLSVLSI '22), June 6–8, 2022, Irvine, CA, USA.* ACM, New York, NY, USA, 6 pages. https://doi.org/10.1145/3526241.3530351

1 INTRODUCTION

Edge computing advocates for data to be processed locally by smart sensors. Such approach leads to high responsiveness, dependability and energy efficiency, since applications do not have to rely on slow, unreliable and energy-hungry sensor-to-server links.

Nonetheless, supporting complex processing algorithms, such as Convolutional Neural Networks (CNNs), on the constrained resources of edge devices is an open challenge [27]. Addressing it, research efforts consider a variety of perspectives, as summarized in Section 2. From a hardware viewpoint, novel architectural approaches have been proposed to efficiently execute convolution operations [6]. In-memory computing solutions are particularly promising in this scenario, since they can aptly support the low-bitwidth, highly parallel operations characterizing CNNs at the edge. At the algorithmic level, optimized CNN models [16] have been introduced to decrease workload requirements. Our work builds upon both these research avenues.

We focus on Bit-line Computing (BC) as an energy-efficient in-memory computing strategy to support CNN applications at ultra-low-power levels. BC operations are based on the concurrent activation of multiple memory words. The output logic values depend on a combination of the voltages stored on the accessed cells, hence performing in-memory bit-wise operations [1]. Few additional logic gates in the memory periphery are then employed to derive arithmetic operations from bit-wise ones. Recent works on BC highlight its high parallelism and ease of integration [22][20], either considering SRAM arrays [13], or hybrid schemes employing both SRAM and RRAM cells [20]. Herein, we showcase that a BC approach can be also leveraged to increase robustness against ultra-low voltage supply levels, effectively coping with the ensuing high error rates. To this end, we introduce a novel BC solution able to detect and mitigate memory errors. Error detection and mitigation operates simultaneously with in-memory computing, and borrows most of its components from the BC circuitry, thus requiring minimal hardware overhead.

Complementary, robustness towards memory upsets can also be increased with algorithmic-level considerations. In particular, the use of multiple CNN models (termed ensembles of CNNs in literature) effectively increases resiliency [21]. While such a strategy usually impacts the required workload, the authors of [17] recently proved that, by combining ensembling and pruning, new ensembling solutions can be derived at no additional cost. Nonetheless, [17] postulates that memory errors can only occur when accessing CNN parameters weights, while inputs, outputs and intermediate results (including activations) are always error-free. Our in-situ error detection/mitigation strategy does not suffer from this limitation. Conversely, we assume, and cope with, high errors rates in *all* memory accesses, anywhere in the BC array.

Our hardware/software co-optimization strategy ultimately enables a very aggressive scaling of the supply voltage with minimal impact on classification accuracy when executing CNN inferences, but with a tangible benefit in terms of energy efficiency. To the best of our knowledge, this work is the first that investigates the applicability of voltage over-scaling in the context of bit-line computing, and the first that demonstrates the robustness of ensembles of CNNs in the presence of high error rates in memory accesses.

In summary, the paper main contributions are as follows:

- We introduce a novel in-memory BC-based architecture that supports transparent in-situ error detection and mitigation, while requiring minimal additional circuitry with respect to state-of-the-art BC solutions.
- We explore the benefits, from a robustness perspective, deriving from ensembles of CNN models in the presence of bit flips in memory accesses, mitigated by our lightweight management scheme.
- Targeting five different CNN models, we showcase that the proposed combination of resilient optimization at the hardware and algorithmic levels are able to cope with a high rate of memory upsets of 0.01 % with accuracy degradations below 1 %. In turn, this characteristic opens the opportunity to aggressively scale the voltage supply, and ultimately leads to energy efficiency gains of up to 51.3 % in the case of ResNext.

2 BACKGROUND

2.1 Domain-specific and in-Memory architectures for CNN inference

CNNs are both memory and data-intensive. Therefore, their energy-efficient execution requires a tightly coupled integration of storage and computing elements. Such features characterize domain-specific architectures such as Neural Processing Units [7] and systolic arrays [5], as surveyed in [6]. Taking this approach even further, In-Memory Computing (IMC) proposes strategies to entirely merge computation and storage capabilities.

Two notable IMC avenues are crossbar interconnects and bit-line computing. The former stores parameter values as programmable resistances in a matrix of Resistive RAM elements [4]. While such approach has the potential of greatly speeding up the matrix-vector multiplication at the core of CNN computation, it also relies on challenging non-CMOS technologies and must cope with non-obvious

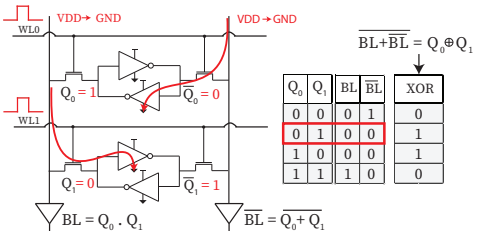

Figure 1: BC operation. BLs behave as the logic gates AND and NOR when concurrently accessing two memory cells on the same BLs (left). A XOR between the memory cell values is derived using an additional NOR gate (right).

implementation problems, e.g., related to noise rejection [26]. Conversely, bit-line computing architectures only require minor modification to SRAM-based memory structures, making them ideal candidates for integration in existing memory hierarchies (e.g. as smart caches [1]), hence minimizing data transfer while enabling a high degree of parallelism.

As in standard SRAMs, in BC arrays each bit-cell is connected to two bit-lines (BL and $\overline{BL}$ in Figure 1) through access transistors. A bit-cell stores a binary value (Q = 0/1) and its complement ($\overline{Q}$ = 1/0) using cross-coupled inverters. To access it, bit-lines must first be pre-charged to the supply voltage. Then, the cell access transistors are activated by asserting the corresponding word-line signal (WL), which causes one bit-line to discharge to ground and the other to hold the charge. The bit-lines are sensed by an amplifier and their logic value assessed. On the other hand, BC operations are performed by asserting two word-line signals simultaneously (as exemplified in Figure 1), providing two possible discharge paths for the bit-lines [22]. Hence, the values sensed on the amplifiers are AND and NOR between the accessed cells on the BL and $\overline{BL}$ lines, respectively. Since word-lines connect to the access transistors of all cells in a memory word, a BC access performs the bitwise AND and NOR among two memory words. The outputs of these operations are then processed at the BC array periphery to realize arithmetic ones, such as additions and multiplications [19].

Crucially, bitwise AND and NOR also provide the basis for efficient in-memory error detection and mitigation, as we show in Section 3. While a recent manuscript presented an error correction mechanism for crossbar IMCs [9], we for the first time propose an error detection approach targeting BC architectures.

2.2 CNN models optimization

Further energy-aware optimization strategies deal with the structure of CNN models. In this context, the research community has proposed hand-crafted high-efficiency models (e.g., MobileNet [11]) that exhibit a lower number of trainable parameters compared to other contemporary state-of-the-art models, while achieving comparable classification accuracy. Techniques have also been proposed to reduce the computational complexity of existing models [16]. Among them, quantization strategies aim at employing integer and small-bitwidth data representations for activations and weights. Pruning approaches instead remove the least critical computation parts, considering granularities ranging from single weights [10] up to entire filters [8], decreasing both workloads at run-time and memory footprints. Unfortunately, aggressive model compressions

achieved via quantization and pruning often incur large accuracy degradations. The ensuing exactness/efficiency trade-off is similar to that derived from voltage overscaling, in which lower supply levels are adopted, decreasing energy consumption but resulting in perturbations due to a non-zero probability of memory access errors [3]. Thus, high model robustness is crucial when performing quantization, pruning and considering very low voltage supplies, as we do in our work.

To this end, the combination of several CNNs performing inference on the same input data (i.e., ensembling) has been shown to be especially effective [2]. Nevertheless, in general, ensembling incurs in large memory and computational overheads because several models are combined to build ensembles. Addressing this issue, the authors of [17] propose a synergic use of quantization, pruning and ensemble methods resulting in models having very high error tolerance, without requiring additional memory and computing resources. In their work, an initial single-instance quantized CNN is first compressed by a factor N via filter pruning. Then, the resulting structure is replicated N times. Each pruned CNN model is individually trained on the target dataset starting from different (random) initial weight values, so that N different trained models are obtained and ensembled.

Herein, we also adopt quantization, pruning and ensembling. As opposed to [17], we do not mandate the presence of an error-free (and high-Vdd) memory region to store activations. Indeed, we show that our approach can cope with non-zero probabilities of error in *all* memory accesses.

3 PARITY-CHECK ON BIT-LINE COMPUTING

3.1 Resilient BC memory overview

Our BC memory are organized as 2-dimensional arrays of SRAM memory cells. Each array row stores a data word and an additional parity bit, which are activated by a word-line (WL) signal. Rows are organized in local groups (LGs), as depicted in Figure 2a. Two address decoders are present. They concurrently assert two WL signals at the same time when performing BC operations requiring two operands. Only one is employed for non-BC memory accesses (reads and writes) and for operations requiring a single operand, such as bit-wise negation. As in [19], two-operands BC operations are possible between any two words, as long as they belong to different LGs. At the array periphery (Figure 2b), inside the Bit-line Computing Unit (BCU), the read/write circuit interfaces with the Bit-wise and Arithmetic Logic (BAL), a circuit that performs in-memory additions, subtractions, shifts and bit-wise operations. These in-memory operation can be leveraged to efficiently perform multiplications [18]. The BAL block is transparent when normal memory reads are performed. The circuitry dedicated to error mitigation also resides in the BCU. Its main components perform parity check and generation, manage the parity bit logic and the detection and mitigation features. We describe this circuit in detail in Section 3.3. External to the BCU, a controller orchestrates its operation dictating which operation is executed at each clock cycle.

3.2 Detection and mitigation strategy

The BC array outlined above provides three features to enhance error resiliency: parity generation, parity check and error mitigation. These are performed entirely in the BCU block, and are hence transparent from a system and application perspective. Jointly, they counter the effect of single bit-flips both on standard memory accesses and during as single- or dual-operands BC operations.

When an error is detected, the value "0" is written to memory (in case of an in-memory operation) or presented at the memory output (in case of a standard memory access). This choice is motivated by the plots in Figure 3, which show the statistical distribution of the computed values (activations) in CNNs is highly skewed towards zero, with only few outliers having a high magnitude. For AlexNet, more than 90% of the activation values are 0, while for RexNext 75% of them are smaller than 0.1% of the representable range.

Error detection itself is straightforward when only one operand is involved, requiring the XORing ($\oplus$) of all BL lines, including that of the parity bit. When instead a two-operand BC operation is performed e.g., between two words $\mathbf{A} = \{A_{n-1}; A_{n-2}; ...; A_0\}$ and $\mathbf{B} = \{B_{n-1}; B_{n-2}; ...; B_0\}$), only the values ($BL_i = A_i \ . \ B_i$) and ($\overline{BL}_i = \overline{A_i \ + \ B_i}$) are available, but not A_i and B_i themselves. The expression for $A_i \ \oplus \ B_i$ can nonetheless be computed from the bit-line values with a NOR gate (as depicted in Figure 1)

$$A_i \ \oplus \ B_i = \overline{BL_i \ + \ \overline{BL_i}} \tag{1}$$

Hence, parity checking in can be performed based on *both* bit-line signals, as follows:

$$\begin{aligned} Parity &= (A_{n-1} \oplus B_{n-1}) \oplus ... \oplus (A_0 \oplus B_0) \\ &= \overline{(BL_{n-1} + \overline{BL_{n-1}})} \oplus ... \oplus \overline{(BL_0 + \overline{BL_0})} \end{aligned} \tag{2}$$

3.3 Detection and mitigation circuit

At the center of the detection/mitigation circuitry is a tree of XOR gates (named Error Detection and Mitigation Unit, or EDMU, in Figure 2). At each memory access, the EDMU is employed to detect if a parity error occurred. If an error is detected, the "Clear" signal is assessed to zero the value at the output of the read/write block. The EDMU inputs are either the bit-line signals (BL) for normal memory reads and single-operand BC operations, or the bitwise NOR between BL and $\overline{BL}$ signals for two-operands ones, as discussed in the previous section. The selection of proper inputs is dictated by a dedicated multiplexer, itself governed by the memory controller.

In case of a BC operation, a new parity bit must be generated. To this end, the EDMU is traversed again, having at its input the value computed by the arithmetic logic block i.e., the result obtained by a given in-memory operation. The output parity bit is then written back to memory at the same time as the data bits.

All the actions outlined above are executed in a single clock cycle. The timing behaviour of our resilient BC computing solution is depicted in Figure 4. First, the bit-lines are pre-charged to Vdd. Then (a), the word-lines are activated to access the memory cells in the desired word or words. Once the voltage on the bit-lines are stable, the parity check is performed by the EDMU on the read value (b). During phase (c), the word-lines are deactivated and the arithmetic unit computes the desired in-memory operation. Its result is stored in a dedicated register if no parity error occurred, otherwise the state of the register is cleared (set to zero). Finally, in phase (d) the EDMU is employed again to generate the output parity bit. In case of in-memory operation, the result must be stored back to memory. This operation is performed in a further clock cycle.

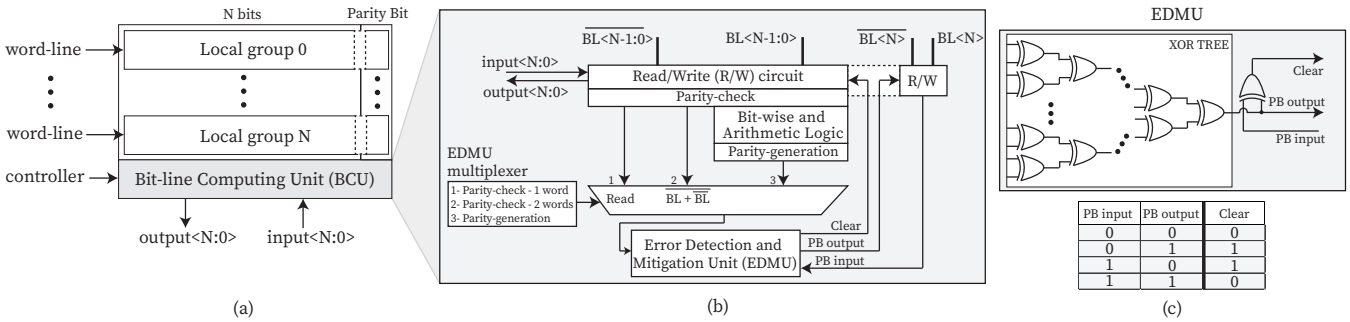

Figure 2: (a) BC memory architecture comprising N-bits words and one parity bit. (b) the Bit-line Computing Unit (BCU), which comprises the read/write circuit, the Bit-wise and Arithmetic Logic, and the Error Detection and Mitigation Unit (EDMU). (c) The EDMU is composed of the XOR-tree and an extra XOR gate to compute the Clear signal.

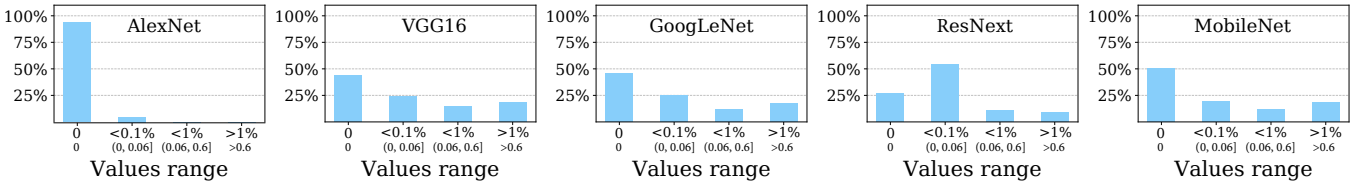

Figure 3: Activations' distribution in single-instance CNNs on the CIFAR-100 dataset. More than 75 % of activations assume values within 1 % of the representable range, with a significant fraction of them being exactly 0.

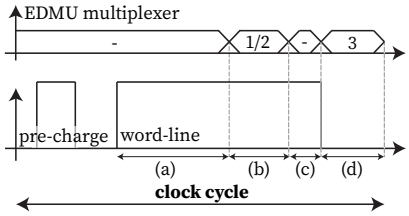

Figure 4: EDMU multiplexer states at run-time in one clock cycle. (a) Read of one or two words. (b) Parity check and in-memory operation. (c) IMC result available. (d) Parity calculation of the new IMC output word.

While the calculation of parity check is performed in parallel with in-memory arithmetic operation in phase (b), parity generation phase (d) does incur in an additional delay. Such delay is nonetheless marginal, being below 10% of the critical path across the experiments we present in Section 5.

4 EXPERIMENTAL SETUP

4.1 Single-instance and ensemble benchmarks

We evaluate our proposed architecture and framework methodology on AlexNet [15], VGG16 [23], GoogLeNet [24], ResNext [25] and MobileNet [11] on the CIFAR-100 dataset [14]. For each of these benchmarks, we consider single-instance, 2- and 4-instances ensemble implementations. To obtain ensembles, we apply the methodology described in [17] compressing a single-instance CNNs via filter pruning and replicating their structure. This approach allows to obtain ensemble models that exhibit equal (or smaller) memory and computational requirements with respect to the corresponding single-instance CNN, but increased accuracy and error robustness (as we show in Section 5) ultimately achieving better accuracy/energy trade-offs.

Table 1: Energy consumption per access and bit error rate for an SRAM built on a 40 nm CMOS process at different voltage levels (fJ/bit) [3].

	Read	Write	BC op.	Error Rate
800 mV	62.7	81.1	101.0	
750 mV	46.9	50.9	74.1	1×10^{-5}
700 mV	36.0	34.9	54.3	1×10^{-4}
650 mV	23.7	24.8	42.3	7×10^{-4}
600 mV	18.6	18.3	32.1	2×10^{-3}

CNNs are trained in PyTorch, using a fake-quantization approach [12] for the last 20 training epochs, at a quantization level of 8 bits for weights and 16 bits for activations. Such setting leads to negligible accuracy drops compared to floating point implementations. Instances of ensembles are trained independently, resulting in compressed models with similar accuracy but slightly different weight values. At run-time, each CNN instance composing an ensemble independently processes the input data, producing separate classification probabilities. These are then averaged together to compute the ensemble output.

4.2 Accuracy evaluation

To explore the accuracy achieved by different CNNs in the presence or absence of our error mitigation strategies, we considered as a starting point the bit-flip probabilities reported in [3] for different supply voltage levels in 40*nm* technology. We then constructed an error model targeting stuck-at faults on a bit-level basis, i.e., assuming a non-zero probability that a bit is always set as a '1' or as a '0', irrespectively of its intended value.

Faults cause observable errors if they affect the representation of the accessed data. Assuming an equal probability of stuck-at-0 and stuck-at-1 faults, the probability of having an observable error in a bit in a memory access is as follows:

$$P_e = \frac{1}{2} P_{stuck-at} \tag{3}$$

Considering a word of n bits (possibly including a parity bit), the probability of having k bit-flips during an access is then:

$$P_{(num-err==k)} = \binom{n}{k} P_e^k (1 - P_e)^{n-k} \tag{4}$$

We implemented Equation 4 as part of an inference solver (written in the C language). When executing multiply-accumulate operations, a non-zero probability of bit-flips is assumed when computing the result[1]. Without any error mitigation schemes, all bit-flips are propagated to successive computations. When instead simulating the strategy outlined in Section 3, results are set to zero in presence of an odd number of bit-flips. The probability of error detection is:

$$P_{error-detection} = \sum_{k=1,3,5,\ldots}^{k=n-1} \left[\binom{n}{k} P_e^k (1 - P_e)^{n-k} \right] \tag{5}$$

The probability of having multiple errors (e.g., 2, 3, 4, ...) in the same memory access decreases exponentially, motivating our choice of only addressing single-error mitigation.

4.3 Energy and area evaluation

We implemented the bit-line computing architecture described in Section 3 in 40nm CMOS technology. Hence, our results are consistent with the bit-error data reported in [3] and presented in Table 1. The array of 16-bits words is composed of 4 LGs of 128 words. The addresses are interleaved in 4 ways and each LG has 32 word-lines. Thus, each subarray stores 1kB. In order to account for the parasitic effects of the physical layout, we implemented RC networks based on the width, length and metal layer. The energy required to read, write and perform an in-memory computing operation is reported in Table 1. We considered as a baseline an iso-size BC array as in [22], which does not feature error mitigation.

To assess energy requirements for different benchmarks, we investigated the number of read, write and BC operation required by an inference on each of the considered benchmarks. To this end, we employed a cycle-accurate simulator[2] as in [20]. The simulator calculates for each CNN layer the number of cycles required to load the inputs, perform the required BC operations and stream out the results. Inputs are tiled and channel-wise separation is performed if data size exceeds the available storage in the BC memory. In this case, additional operations are reported to stream partial convolutions data to the memory array, perform the aggregation operations in-memory, and retrieve full convolutions outputs.

5 RESULTS

5.1 Area, energy and performance breakdown

Figure 5 shows the floor plan of our subarray implementation. The total area of the subarray is 3448 μm^2, of which 76.9% (2651 μm^2) is occupied by the high-density SRAM bit-cells from TSMC and the sense-amplifiers of the 4 LGs, each bit-cell has a surface of 0.253 μm^2. The SRAM bit-cells occupy most of the LG surface, with the

[1]MAC operations are executed as a sequence of shift-adds among two operands, in which each of the two may be affected by stuck-at faults.
[2]https://www.epfl.ch/labs/esl/research/open-source-tools/cnn2blade/

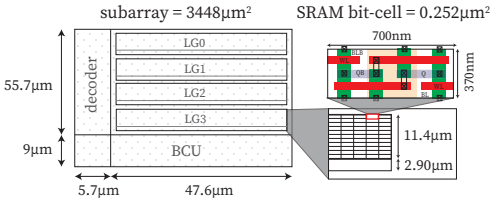

Figure 5: Area breakdown of a single BC subarray.

LG sense amplifier representing 20% of its total area. Moreover, an overhead of 5.9% (1/17) of the 4 LGs is necessary to store the parity bit. Finally, the overhead of the BCU to enable arithmetic operations and our error mitigation strategy is just 12.4%.

The energy values per bit for the read, write and BC operations are presented in Table 1. Reducing the supply voltage from 800mV to 700mV reduces the energy of each operation by 47%, while at 600 mV the total energy reduction reaches 72%. However, considering our implementation with parity-bit, the read and BC operation energy increases by 15%. This overhead is due to the parity bit access and the EDMU, which is used twice during the same cycle in these operations, as described in Section 3.3.

Operating at sub-nominal voltages forces a reduction of the working frequency. Indeed, a supply voltage of just 650 mV reduces the maximum operating frequency by more than 40 % compared with the 800 mV baseline. Nevertheless, in this condition, our architecture is still able to run at 300 MHz, a relatively high frequency in the field of ultra-low power devices. Moreover, in-memory operations are parallelized by employing multiple subarrays to increase throughput, which compensates the frequency reduction.

5.2 Accuracy/energy trade-off

The accuracy achieved at different sub-nominal voltages is presented in Figure 6, where the energy cost of inference for different benchmarks is shown on the x-axis. Black curves correspond to baseline single-instance CNNs and blue ones represent the same models, in which our error mitigation strategy is applied. Finally, green and yellow lines correspond to ensemble-based solutions, also including the proposed error mitigation strategy. Different markers highlight different supply voltages (hence error rates) as introduced in Table 1. We considered supply ranging from 800mV (a level in which no error occurs) down to 600 mV. These results indicate that voltage scaling dramatically impacts the accuracy of baseline solutions. In particular, by slightly reducing the voltage from 800mV to 750mV the accuracy of the considered benchmarks is reduced to 59.8% on average. Our experiments report similar effects in ensembles (i.e., where our error mitigation strategy is not implemented), highlighting that, despite their increased accuracy and robustness, errors affecting activations are still critical [17].

On the other hand, Figure 6 also demonstrates the accuracy improvements of our error mitigation approach at any evaluated sub-nominal voltage. More precisely, star points show that more aggressive voltage scaling can be applied in our BC architecture, which they retain a better accuracy level. The minimal energy overhead due to the additional circuitry performing the parity check is largely compensated by the possibility to reduce the supply voltage. Thus, it ultimately results in a more favorable accuracy/energy trade-off. On average, energy savings of 41.2 % can be achieved with

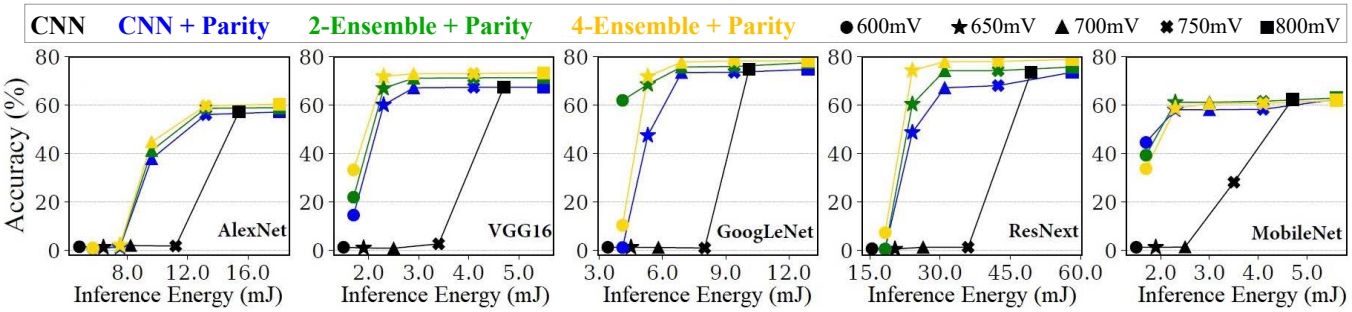

Figure 6: Energy-per-inference vs. accuracy, varying the supply voltage. Black line: baseline BC implementation. Blue-green-yellow lines: BC array featuring error detection and mitigation, employing different ensemble sizes.

our methodology, while preserving the baseline accuracy. Additionally, combining our error mitigation strategy with the described ensembles results in even more advantageous energy/accuracy trade-offs. In fact, our error mitigation strategy reduces the inexactness introduced by memory errors in the activations, thus limiting their impact on the accuracy of ensembles, that can achieve, on average, 8.2 % higher accuracy than single-instance CNNs, when our proposed solution is applied. In this context, ensembles serve two purposes: on one side, they increase the initial inference accuracy at nominal voltage (i.e., error-free executions), and, on the other side, their additional robustness against errors is exploited to enable more aggressive voltage scaling. This effect is particularly evident in Figure 6 for VGG16, GoogLeNet, and ResNext, where the curves of ensembles exhibit a smoother accuracy degradation due to voltage reduction compared to single-instance alternatives.

In GoogLeNet, the 2-Ensemble option offers significantly higher accuracy than the 4-Ensemble one at 650 mV, while the latter configuration outperforms the former one at any other voltage level. This behavior has already been discussed in [17], where the authors underline that larger ensembles require more aggressive pruning on the initial CNN. The result is a lower accuracy of the individual pruned CNNs that, in presence of high error rates, may not be recovered by the ensemble. Nevertheless, combining our error mitigation technique with ensemble-based solutions allows the supply voltage to be reduced to just 650 mV, resulting in energy savings up to 51.3 % with minimal impact on the initial CNNs accuracy.

6 CONCLUSION

In this paper, we have proposed a new bit-line computing architecture designed to support aggressive supply voltage scaling when running CNN inference, thanks to the implementation of a transparent error mitigation technique. Additionally, we have demonstrated how the synergic use of constrained ensembles of CNNs and the proposed error mitigation method can improve the robustness against memory errors. Our results have shown that voltage can be reduced from the nominal 800 mV down to just 650 mV, enabling energy savings up to 51.3 % without affecting the initial CNN accuracy.

ACKNOWLEDGMENTS

This work has been supported by the EC H2020 WiPLASH (GA No. 863337), the EC H2020 FVLLMONTI (GA No. 101016776), the ERC Consolidator Grant COMPUSAPIEN (GA No. 725657) and the Swiss NSF ML-Edge RTD (GA No. 20002018200) projects.

REFERENCES

[1] K. C. Akyel et al. 2016. DRC2: Dynamically Reconfigurable Computing Circuit based on memory architecture. *IEEE ICRC.*
[2] M. Amin-Naji et al. 2019. Ensemble of CNN for multi-focus image fusion. *Elsevier, Information Fusion.*
[3] D. Bortolotti et al. 2014. Approximate compressed sensing: Ultra-low power biosignal processing via aggressive voltage scaling on a hybrid memory multi-core processor. *ISLPED.*
[4] G. Burr et al. 2015. Experimental demonstration and tolerancing of a large-scale neural network (165 000 synapses) using phase-change memory as the synaptic weight element. *IEEE TED.*
[5] YH. Chen et al. 2016. Eyeriss: An energy-efficient reconfigurable accelerator for deep convolutional neural networks. *IEEE J. Solid-State Circuits.*
[6] Y. Chen et al. 2020. A survey of accelerator architectures for deep neural networks. *Elsevier, Engineering.*
[7] H. Esmaeilzadeh et al. 2014. Neural acceleration for general-purpose approximate programs. *Commun. ACM.*
[8] Y. He et al. 2020. Learning filter pruning criteria for deep convolutional neural networks acceleration. *Proc. IEEE Int. Conf. Comput. Vis.*
[9] B. Feinberg et al. 2018. Making memristive neural network accelerators reliable. In *IEEE HPCA.*
[10] S. Han et al. 2015. Learning both weights and connections for efficient neural networks. *arXiv:1506.02626.*
[11] A. G. Howard et al. 2017. Mobilenets: Efficient convolutional neural networks for mobile vision applications. *arXiv:1704.04861.*
[12] B. Jacob et al. 2018. Quantization and training of neural networks for efficient integer-arithmetic-only inference. In *IEEE ICCV.*
[13] S. Jeloka et al. 2016. A 28 nm configurable memory (TCAM/BCAM/SRAM) using push-rule 6T bit cell enabling logic-in-memory. *IEEE JSSC.*
[14] A. Krizhevsky et al. 2009. Learning multiple layers of features from tiny images. *University of Toronto.*
[15] A. Krizhevsky et al. 2012. Imagenet classification with deep convolutional neural networks. In *Advances in neural information processing systems.*
[16] T. Liang et al. 2021. Pruning and Quantization for Deep Neural Network Acceleration: A Survey. *arXiv:2101.09671.*
[17] F. Ponzina et al. 2021. E2CNNs: Ensembles of Convolutional Neural Networks to Improve Robustness Against Memory Errors in Edge-Computing Devices. *IEEE Trans. Comput.*
[18] F. Ponzina et al. 2021. A Flexible In-Memory Computing Architecture for Heterogeneously Quantized CNNs. In *IEEE Computer Society Annual Symp. on VLSI.*
[19] M. Rios et al. 2019. An Associativity-Agnostic in-Cache Computing Architecture Optimized for Multiplication. *IEEE VLSI-SoC.*
[20] M. Rios et al. 2021. Running Efficiently CNNs on the Edge Thanks to Hybrid SRAM-RRAM In-Memory Computing. *IEEE/ACM DATE.*
[21] H. Sagha et al. 2013. On-line anomaly detection and resilience in classifier ensembles. *Elsevier, Pattern Recognition Letters.*
[22] W. A. Simon et al. 2020. BLADE: An in-Cache Computing Architecture for Edge Devices. *IEEE Trans. Comput.*
[23] K. Simonyan et al. 2014. Very deep convolutional networks for large-scale image recognition. *arXiv:1409.1556.*
[24] C. Szegedy et al. 2015. Going deeper with convolutions. *IEEE CVPR.*
[25] S. Xie et al. 2017. Aggregated Residual Transformations for Deep Neural Networks.
[26] T.H. Yang et al. 2019. Sparse ReRAM engine: Joint exploration of activation and weight sparsity in compressed neural networks. *Proceedings of the 46th International Symposium on Computer Architecture.*
[27] Z. Zhou et al. 2019. Edge intelligence: Paving the last mile of artificial intelligence with edge computing. *Proc. IEEE.*