electronics

MDPI

Article

RapproX: An Adaptive Approximate Adder with Lookback for Efficient Edge AI via Memristive In-Memory Computing

Lukas Rapp [†], **Leandro Borzyk** [†], **Fabian Seiler**, **Nima Amirafshar** and **Nima TaheriNejad** [*]

Institute of Computer Engineering (ZITI), Heidelberg University, 69117 Heidelberg, Germany; lukas.rapp@stud.uni-heidelberg.de (L.R.); fabian.seiler@ziti.uni-heidelberg.de (F.S.); nima.amirafshar@ziti.uni-heidelberg.de (N.A.)

* Correspondence: nima.taherinejad@ziti.uni-heidelberg.de

† These authors contributed equally to this work.

Abstract

As silicon scaling nears its physical limits and digital systems process ever-growing amounts of data, integrating computation directly within memory is emerging as a key strategy to overcome the constraints of conventional Von Neumann architectures. Approximate In-Memory Computation (IMC) with memristors offers a promising path toward energy-efficient processing for data-intensive applications. Recent adaptive approximate adders exploit operand magnitude to dynamically switch between exact and approximate computation, but typically ignore carry propagation across approximation boundaries, which can significantly degrade application-level robustness. This work introduces RapproX, a family of adaptive memristive approximate adders featuring a lightweight carry lookback mechanism that approximates carry interaction between exact and approximate regions. The proposed approach improves arithmetic robustness while introducing only minimal overhead and enabling resource-efficient implementations through memristor reuse. Experimental results demonstrate that the proposed approaches achieve superior arithmetic quality compared to State-of-the-Art (SoA) memristive approximate adders. More importantly, the carry lookback mechanism translates into substantial application-level benefits. In image processing, RapproX reduces energy consumption by up to 30.9% compared to the most competitive SoA design and by 50.3% compared to exact computation while maintaining roughly 43 dB Peak Signal-to-Noise Ratio (PSNR). Across a range of machine-learning workloads, including k-means, AlexNet on MNIST, and multiple CIFAR-10 models, RapproX preserves near-exact inference accuracy for the evaluated models at low-to-moderate k and maintains the energy advantages of adaptive approximation, while SoA approximations degrade markedly under the same conditions. These simulation-based results suggest that lightweight carry-aware approximation can improve the robustness of adaptive approximate in-memory computing with only marginal hardware overhead.

Keywords: in-memory computing; approximate computing; energy-efficient computing; approximate adder; stateful logic; image processing; machine learning; deep learning

Academic Editor: Javid Taheri

Received: 15 June 2026
Revised: 8 July 2026
Accepted: 4 August 2026
Published: 6 August 2026

1. Introduction

Approximate computing has emerged as a promising paradigm for improving the energy efficiency of digital systems by trading off controlled accuracy loss for reductions in energy consumption, latency, and circuit area [1–3]. This trade-off is particularly effective in error-resilient applications such as image processing and machine learning, where

system-level metrics—such as classification accuracy, Peak Signal-to-Noise Ratio (PSNR), and Structural Similarity Index Measure (SSIM)—are often only marginally affected by arithmetic inaccuracies [4–6]. At the same time, the growing disparity between computation and data movement costs has made memory access a dominant contributor to system energy consumption [7]. This has motivated the shift toward memory-centric computing approaches, where processing-in-array [1] stands out as the most efficient approach since operations are performed directly within the crossbar array and data never leaves the memory, reducing the transfer overhead. Memristor-based In-Memory Computation (IMC) is a prominent realization of this paradigm, since they enable both storage and computation through stateful logic operations such as IMPLY [8], FELIX [9], MAGIC [10], and SIXOR [11], offering compact and energy-efficient implementations of arithmetic primitives. Despite their versatility and high parallelization abilities, this requires many steps for each (sub)function, which renders them good targets for approximations. Adders play a central role due to their ubiquitous use in both signal processing and machine-learning workloads. This led to various approximate adder designs using memristive stateful logic primitives [1,12–17]. Recent works, such as the Adaptive Precision Adder Framework (APAF) [18], exploit operand magnitude to dynamically switch between exact and approximate computation, achieving significant energy savings. A key limitation of such magnitude-based approaches is the omission of carry propagation between approximated and exact regions. Though seemingly minor arithmetically, the resulting errors in the most significant bits can propagate non-linearly in application-level workloads such as deep neural networks [1,17].

We introduce RapproX, a family of adaptive approximate memristor-based in-memory adders that address this limitation via a lightweight carry lookback mechanism. By approximating the carry-in at the approximation boundary, RapproX improves arithmetic robustness with minimal overhead. Because RapproX applies this carry approximation inside a magnitude-based adder; the boundary carry and the exact upper computation are performed only when the operand magnitude requires them, so time and energy are spent on the carry only when it is needed. A static carry-aware adder such as S-SINC+ [15] instead computes the full exact section on every addition. On top of this, our XOR variant recomputes the topmost approximated bit with a single-cycle SIXOR, which cleans up the residual error the approximated carry would otherwise leave at the position of highest weight and considerably improves arithmetic quality. Furthermore, we show that improving low-level arithmetic behavior can translate into substantial gains at the application level. In the machine-learning workloads considered here, where operand distributions are highly non-uniform, RapproX effectively exploits sparsity and small-value dominance to skip unnecessary computations. In our simulations, this yields improved energy efficiency compared to the evaluated State-of-the-Art (SoA) designs, while maintaining near-exact inference accuracy for the models considered here. These findings suggest that architectural improvements at the arithmetic level can have a sizeable impact on application-level performance in our experiments. The main contributions of this work are summarized as follows:

- Two novel adaptive approximate architectures that combine magnitude-based adaptation with a carry-lookback, raising energy efficiency by considerably improving arithmetic quality while only slightly increasing energy.
- A novel efficient SIXOR-based XOR complement for the carry approximation that corrects the residual carry error at the topmost approximated bit.
- A detailed error analysis, which outlines why the lookback mechanism leads to significant performance increases compared to SoA designs.

- A comprehensive evaluation at the application level, including image processing, k-means clustering, and deep neural networks evaluated on MNIST and CIFAR-10, which highlights the performance gains using our methodology.

The remainder of this work is organized as follows. Section 2 reviews the relevant literature. Section 3 introduces the proposed RapproX designs. In Section 4, we conduct an analysis of arithmetic quality. Section 5 compares them with SoA approaches in terms of area, speed, and energy consumption. Application workloads spanning image processing and machine learning are evaluated in Section 6, followed by the conclusion in Section 7.

2. Background and Related Work

This section reviews the background needed for the remainder of the paper. Section 2.1 recaps memristor devices and the stateful-logic families used to build in-memory arithmetic, and situates stateful logic within the broader landscape of memristive in-memory computing. Section 2.2 then surveys memristive approximate adders, with a focus on the magnitude-adaptive designs that motivate RapproX.

2.1. Memristors and Stateful Logic

Memristors are two-terminal non-volatile devices whose resistance depends on the history of applied voltage and current, as first theorized by Chua [19] and experimentally demonstrated by Strukov et al. [20]. Their ability to store information in resistive states (typically R_{on} and R_{off}) enables compact and energy-efficient memory implementations. Beyond storage, memristors inherently support in-memory computation by enabling logical operations directly within memory arrays. This eliminates the need for frequent data transfers between memory and processing units, addressing the von Neumann bottleneck [21]. Due to their low power consumption, high density, and compatibility with crossbar architectures, memristors are considered a key enabler for IMC systems [22,23].

The analog behavior of multi-stable memristors is widely used in applications such as neural-network acceleration via analog matrix–vector multiplication performed directly within crossbar arrays [24–26] and neuromorphic spiking neural networks [27–29]. These approaches serve as efficient domain-specific accelerators for linear algebra and neuromorphic workloads. In contrast, this work focuses on digital stateful logic, which stores information in two stable resistive states and enables general-purpose Boolean and arithmetic operations directly within the memory array.

Stateful logic enables computation directly within memory by exploiting the resistive states of memristors. Logical operations are realized through voltage divider circuits that induce conditional switching, allowing multiple memristors to interact and update their states without explicit data movement. Several stateful logic families have been proposed, as illustrated in Figure 1. These include Material Implication (IMPLY) [30], Three Memristor Stateful Logic (TMSL) [31], Fast and Energy-efficient Logic (FELIX) [9], and Single-cycle XOR (SIXOR) [11], which implement fundamental Boolean operations such as implication, AND, OR, and XOR directly within the crossbar. Another well-known approach is Memristor-Aided Logic (MAGIC) [10]; however, it is only compatible with specific device technologies [32,33] and is therefore not considered in this work. Although these logic families differ in latency, device requirements, and implementation complexity, they all rely on the same principle of voltage-driven state transitions.

The compatibility with the crossbar and control parameters of specific stateful logic operations is highly dependent on the underlying device characteristics. Each family requires specific switching voltages and on/off ratios. Since memristors are an emerging class of memories, the pool of suitable devices continues to expand. Beyond the well-established HfO_2- and TaO_x-based memristors [34], newer material systems, such as halide

perovskites [35] and graphene oxide [36], have recently been shown to support the same stateful logic primitives.

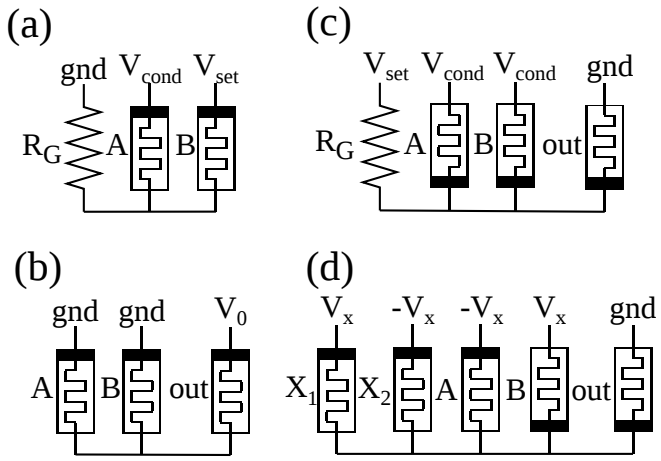

Figure 1. Overview of stateful logic families: (**a**) IMPLY [30] ($B' = A \rightarrow B$), (**b**) TMSL-AND [31] ($out = A \wedge B$), (**c**) FELIX-OR [9] ($out = A \vee B$), (**d**) SIXOR-XOR [11] ($out = A \oplus B$).

Arithmetic units, particularly adders, are key building blocks in memristive IMC. There exist multiple topologies, which can be broadly categorized into serial, parallel, and hybrid topologies [1,15]. Serial adders execute operations sequentially on a single row or column, resulting in minimal hardware requirements at the cost of latency. In contrast, parallel designs distribute computation across multiple rows to improve speed at the cost of increased memristor usage. Hybrid approaches aim to balance these trade-offs. In practice, serial topologies are often preferred, as their low hardware overhead and efficient resource utilization enable higher effective throughput in crossbar-based systems, especially under area and energy constraints [18,37,38].

2.2. Approximate Memristive Adders

Approximate computing has been widely applied to arithmetic units to reduce energy consumption and latency by relaxing accuracy constraints [2,3,39–42]. Recent CMOS approximate adders have likewise incorporated carry prediction and compensation mechanisms to improve arithmetic accuracy while maintaining low hardware overhead and delay [2,39,43–47]. Although these approaches pursue a similar objective of mitigating errors caused by approximated carry propagation, they target conventional CMOS logic. In contrast, RapproX implements a lightweight carry lookback mechanism directly within memristive stateful logic, where computations are executed inside the memory array, and additional logic operations must be carefully minimized to preserve the efficiency of in-memory computing.

While early work focused on CMOS-based designs, these concepts have been increasingly transferred to memristive IMC within the crossbar array [1]. Various approximate adders using IMPLY [12–16] and other approaches [17,48–50] have been presented. Initial memristor-based approximate adder designs [12–14] are based on the relation $Sum \approx \overline{C_{out}}$ to avoid the computationally expensive XOR operation. Subsequent work [15,16] explored different architectural trade-offs, achieving significant improvements in energy efficiency and runtime. A common strategy in these designs is the elimination or simplification of carry propagation, which is a major contributor to computational cost. While effective, this introduces structured errors that can propagate into higher-order bits, limiting robustness. Recently, the Adaptive Precision Adder Framework (APAF) [18] was proposed, which dynamically partitions operands into exact and approximate regions based on their magnitude. The procedure is illustrated in Figure 2. By evaluating the most significant bits using

a FELIX-OR operation, APAF determines whether to skip computation or approximate the lower section. This approach is particularly effective for workloads with skewed data distributions, such as machine learning (especially deep learning approaches), where many operands are small and full-precision computation is often unnecessary. However, a key limitation of APAF and similar approaches is the lack of carry interaction between approximate and exact regions. The carry generated in the approximated section is typically ignored, which can lead to significant errors in the most significant bits.

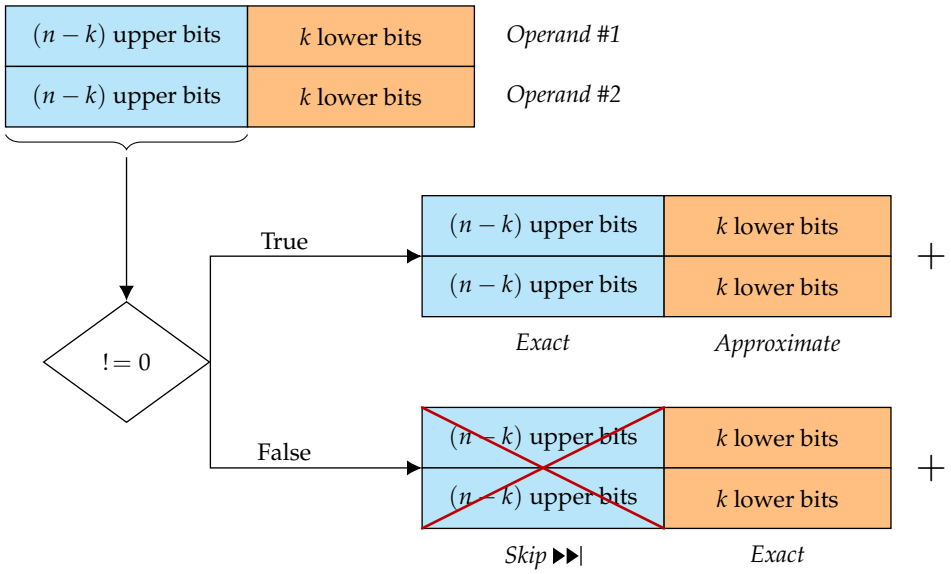

Figure 2. APAF decision mechanism flow chart showing both operands are split into upper and lower sections. If all bits in the upper section are zero, just the lower section is computed exactly. Otherwise, the lower section is approximated, and the upper section is computed exactly.

In contrast, this work introduces RapproX, which addresses this limitation through a lightweight carry lookback mechanism. Instead of discarding carry information entirely, RapproX approximates the carry interaction across region boundaries, reducing error propagation while retaining most of the efficiency gains observed for SoA designs. Prior memristive approximate adders avoid the XOR in the sum because it is costly in IMPLY logic [12,15]. We take a different direction using the single-cycle SIXOR primitive [11], with which our XOR variant computes only the topmost approximated bit with an XOR, correcting the residual error the approximated carry leaves at the highest-weight position at negligible cost. Furthermore, beyond circuit-level improvements, this work emphasizes application-level impact, indicating that improved arithmetic behavior can translate into better simulated results in image processing and, in particular, machine-learning workloads.

3. RapproX Designs

This section introduces the RapproX designs and the architectural decisions that distinguish them from prior magnitude-adaptive approximate adders. Section 3.1 presents the one-bit carry lookback (C1) that reintroduces carry interaction across the approximate and exact boundary. Section 3.2 then generalizes the lookback to a family of m-bit predictors and shows why deeper variants recover little accuracy at a much higher hardware cost, which motivates the single-bit choice. Section 3.3 motivates the sequential scheduling of the approximate and exact sections with the resulting memristor reuse, defines the two proposed variants RapproX_C1_OR and RapproX_C1_XOR that differ in how the lower section is approximated, and formalizes the full procedure of both. Section 3.4 then details how the resulting adder maps onto a single crossbar row.

3.1. Carry Lookback

A notable limitation of recent approximate adders is the omission of carry propagation from the lower (approximated) section to the upper (exact) section. This can lead to significant errors in the most significant bits and, in the worst case, may even affect the sign of the result. To address this limitation, we introduce a lightweight one-bit carry lookback approximation (C1) for the carry-out ($c_{\text{out}}^{\text{C1}}$) generated by the lower section. For a k-bit approximate section, the proposed mechanism derives the carry-out from the two highest-order bits in this segment, a_{k-1} and b_{k-1}, which directly precede the exact region. Specifically, a carry is generated when both bits are equal to one:

$$c_{\text{out}}^{\text{C1}} = (a_{k-1} \wedge b_{k-1}) \tag{1}$$

Prior magnitude-adaptive adders drop this boundary carry and default it to zero. We refer to this choice as the baseline. Whether either prediction is right depends on the two top bits (a_{k-1}, b_{k-1}) together with the true carry c_{in} that propagates up from the lower section. Under uniformly distributed inputs, these three bits span the eight equally likely cases in Table 1, where the true carry out is one whenever at least two of the three bits are one.

Table 1. Carry-out prediction of the baseline and of C1 across the eight equally likely cases of the two top approximate bits and the incoming carry c_{in} under uniformly distributed inputs. A check marks a correct prediction of the true carry out, and a cross indicates a wrong one.

a_{k-1}	b_{k-1}	c_{in}	True c_{out}	Baseline ($c_{\text{out}} = 0$)	C1 ($c_{\text{out}} = a_{k-1} \wedge b_{k-1}$)
0	0	0	0	✓	✓
0	0	1	0	✓	✓
0	1	0	0	✓	✓
1	0	0	0	✓	✓
1	1	0	1	×	✓
1	1	1	1	×	✓
0	1	1	1	×	×
1	0	1	1	×	×

The baseline is correct on the four rows where the true carry is zero, so its correctness is $4/8 = 1/2$. C1 leaves these four rows unchanged and additionally fixes the two $(1,1)$ rows, where a carry is always generated, which adds $2/8$ and raises the correctness to $6/8 = 3/4$. The remaining $1/4$ sits entirely in the two mixed rows $(0,1)$ and $(1,0)$ with an incoming carry. There the true carry out equals the carry propagated up from the lower bits, which the AND in Equation (1) cannot represent because it yields zero whenever either top bit is zero. C1 therefore predicts zero on both mixed states and is right only when that propagated carry is itself zero. This limit is structural, since a single top bit pair cannot observe the propagated carry, and it is also input dependent. Workloads in which the mixed states or lower-section overflows are more frequent than under uniformity shift the effective correctness below $3/4$, whereas workloads in which $(0,0)$ or $(1,1)$ dominate shift it above.

3.2. Ablation of the Carry Lookback to m-Bit

The proposed one-bit carry lookback (C1) corrects the dominant carry-generation pattern at the boundary between the approximate and exact regions. The remaining prediction errors occur when the carry is generated further inside the approximate region and propagates through one or more higher bit positions before reaching the exact region. This naturally motivates extending the lookback depth. More generally, the proposed

carry lookback belongs to a family of m-bit predictors that estimate the boundary carry using progressively larger windows of the approximate region. Following the standard carry-lookahead formulation, the general m-bit predictor is expressed as

$$c_{\text{out}}^{\text{C}m} = G_{k-1} \vee (P_{k-1} \wedge G_{k-2}) \vee (P_{k-1} \wedge P_{k-2} \wedge G_{k-3}) \vee \dots \tag{2}$$

where $G_i = a_i \wedge b_i$ denotes that bit position i generates a carry independently, while $P_i = a_i \vee b_i$ indicates that an incoming carry propagates through position i. The proposed C1 predictor retains only the first generate term ($a_{k-1} \wedge b_{k-1}$), thereby observing only the bit directly preceding the exact region. This already increases the carry prediction correctness from $1/2$ to $3/4$. Extending the lookback to two bits (C2) additionally captures carries generated one position lower that propagate through the Most-Significant Bit (MSB) pair,

$$c_{\text{out}}^{\text{C2}} = (a_{k-1} \wedge b_{k-1}) \vee \big((a_{k-1} \vee b_{k-1}) \wedge (a_{k-2} \wedge b_{k-2})\big). \tag{3}$$

Consequently, C2 removes half of the remaining C1 prediction errors and increases the carry prediction correctness to $7/8$. Applying the same principle one stage deeper yields the three-bit lookback (C3), which again removes approximately half of the remaining errors and reaches a correctness of $15/16$. Thus, each additional lookback bit progressively improves prediction accuracy, but with diminishing returns.

Increasing the lookback depth also increases the implementation cost. Table 2 summarizes the hardware overhead for the three evaluated predictors. For the reference implementation ($n = 8$, $k = 4$), the lookback logic grows from one memristor in C1 to five in C2 and ten in C3, increasing its share of the overall design from 4.5% to 19.2% and 32.3%, respectively. The execution latency and peripheral circuitry scale proportionally because each additional carry-lookahead term requires another stateful-logic evaluation.

Table 2. Correctness against hardware overhead of the m-bit carry lookback, evaluated on the RapproX_C1_OR reference at $n = 8$ and $k = 4$. Correctness is the carry-out prediction probability under uniformly distributed inputs. Overhead columns count only the lookback logic, with the design total in parentheses for the memristor column.

Predictor	Correctness	Memristors (Total)	Cycles	Transistors	Resistors
C1	3/4	1 (22)	1	4	1
C2	7/8	5 (26)	5	20	5
C3	15/16	10 (31)	10	40	10

The improvement in prediction accuracy therefore comes at a rapidly increasing hardware cost. While each additional lookback stage removes roughly half of the remaining prediction errors, the required memristors, execution cycles, and peripheral circuitry grow much faster. For the evaluated designs, the correctness gained per additional execution cycle decreases from 25 percentage points for C1 to approximately 3.1 for C2 and 1.25 for C3. Consequently, C1 represents the knee of the accuracy–hardware trade-off: it captures the dominant carry-generation pattern with minimal hardware overhead, whereas deeper lookback mechanisms provide progressively smaller accuracy improvements at substantially higher implementation cost. This observation motivates the selection of the single-bit carry lookback in the proposed RapproX architecture.

3.3. Algorithmic Implementation

In contrast to prior approaches [18] which compute approximate and exact sections in parallel, RapproX performs the computation sequentially. This enables the reuse of memristors from the exact addition logic, reducing the overall memristor count. Although

the additional operation introduces one extra cycle, the runtime overhead remains small (3.5% for $n = 8, k = 4$), while achieving an 8.3% area reduction. Because the lower-section operations are individually inexpensive, running them in parallel with the exact section saves too little latency to justify the extra work memristors it would require.

On top of this schedule, we propose two variants of the RapproX design that differ in how the lower section is approximated. RapproX_C1_OR approximates every bit of the lower section with an OR, while RapproX_C1_XOR improves arithmetic accuracy by replacing the OR at the topmost approximated bit, s_{k-1}, with an XOR. While the OR maps the case $a_{k-1} = b_{k-1} = 1$ to a sum bit of 1, the XOR yields the correct value of 0, eliminating the largest per-bit OR error at the position with the highest weight in the approximate region. This can translate into performance improvements at end-applications (see Section 6). Based on the SIXOR implementation [11], this operation requires approximately 49 pJ, two additional memristors, and one cycle, discussed in more detail in Section 5.

The RapproX adder operates on two n-bit operands $a_{n-1:0}$ and $b_{n-1:0}$, partitioned into a lower approximate section of k bits and an upper exact section of $n - k$ bits. A magnitude-based decision is performed using a FELIX-OR operation over the MSBs:

$$m_{OR} = \bigvee_{i=k}^{n-1} (a_i \vee b_i). \tag{4}$$

If $m_{OR} = 1$, the upper section is computed exactly, while the sum bits of the k-bit lower section are approximated as $s_i = a_i \vee b_i$ for $i < k$, using FELIX-OR operations [9]. The $c_{\text{out}}^{\text{C1}}$ is calculated by an adapted TMSL AND operation via the one-bit lookback approximation. In the XOR variant, only the topmost approximated bit, which directly precedes the exact region, is computed with $s_{k-1} = a_{k-1} \oplus b_{k-1}$ to reduce the resulting error, while the remaining lower bits $i < k - 1$ still use $s_i = a_i \vee b_i$. For $m_{OR} = 0$, only the lower section is computed exactly, whereas the upper section is omitted, as shown in Figure 2. The detailed procedure descriptions of the OR and XOR variants are shown in Algorithms 1 and 2, respectively.

Algorithm 1: Proposed RapproX_C1_OR Procedure

Inputs: $a_{n-1:0}, b_{n-1:0} \in \mathbb{B}^n, k \in \mathbb{N}_{\leq n}$;
$m_{OR} \leftarrow \bigvee_{i=k}^{n-1}(a_i \vee b_i)$
if $m_{OR} = 1$ **then**
  $c_k \leftarrow a_{k-1} \wedge b_{k-1}$
  for $l := k; l \leq n-1; l \leftarrow l+1$ **do**
    $(S_l, c_{l+1}) \leftarrow ADD(a_l, b_l, c_l)$
  end
  for all $i \in [0, k-1]$ **do in parallel**
    $S_i \leftarrow a_i \vee b_i$
  end
else
  for $m := 0; m \leq k-1; m \leftarrow m+1$ **do**
    $(S_m, c_{m+1}) \leftarrow ADD(a_m, b_m, c_m)$
  end
end
return: $(c_n, S_{n-1:0})$

Algorithm 2: Proposed RapproX_C1_XOR Procedure

Inputs: $a_{n-1:0}, b_{n-1:0} \in \mathbb{B}^n, k \in \mathbb{N}_{\leq n}$;
$m_{OR} \leftarrow \bigvee_{i=k}^{n-1}(a_i \vee b_i)$
if $m_{OR} = 1$ **then**
  $c_k \leftarrow a_{k-1} \wedge b_{k-1}$
  for $l := k; l \leq n-1; l \leftarrow l+1$ **do**
    $(S_l, c_{l+1}) \leftarrow ADD(a_l, b_l, c_l)$
  end
  for all $i \in [0, k-1]$ **do in parallel**
    if $i < k-1$ **then**
      $S_i \leftarrow a_i \vee b_i$
    else
      $S_i \leftarrow a_i \oplus b_i$
    end
  end
else
  for $m := 0; m \leq k-1; m \leftarrow m+1$ **do**
    $(S_m, c_{m+1}) \leftarrow ADD(a_m, b_m, c_m)$
  end
end
return: $(c_n, S_{n-1:0})$

3.4. Crossbar Mapping

The proposed RapproX adders are implemented using mixed memristor-based stateful logic within a 1T1R crossbar architecture, where each cell consists of a memristor and a transistor used to select specific devices. This was employed since it significantly reduces sneak-path effects found in conventional passive arrays. The corresponding circuit realization of the RapproX_C1_OR variant is shown in Figure 3, while its crossbar mapping is illustrated in Figure 4. The RapproX_C1_XOR variant follows the same structure, replacing the last approximate OR computations with SIXOR-based XOR operations. A key advantage of RapproX is that the complete adder can be mapped to a single crossbar row. This allows all rows of the array to operate independently and in parallel, providing a throughput advantage over parallel and hybrid approximate adders that require multiple rows for a single addition [13,23]. To enable such parallel execution, the crossbar is partitioned into independent regions using standard partitioning techniques [9]. The required peripheral circuitry, including voltage sources, switching networks, and support circuitry for IMPLY and TMSL operations, is located outside the crossbar array.

The implementation combines IMPLY logic [30] for exact addition, FELIX-OR [9] for magnitude detection and approximation, and TMSL-AND [31] for the carry lookback operation. The RapproX_C1_XOR variant employs SIXOR logic [11] for the MSB of the approximate range, while RapproX_C1_OR utilizes FELIX-OR for all approximate bits. Unlike fully parallel designs such as ApprOchs [18], RapproX performs magnitude detection, carry lookback evaluation, exact addition, and approximate computation sequentially. This execution model enables extensive memristor reuse across computation phases, reducing hardware requirements while preserving comparable latency due to the single-cycle nature of the additional lookback operations.

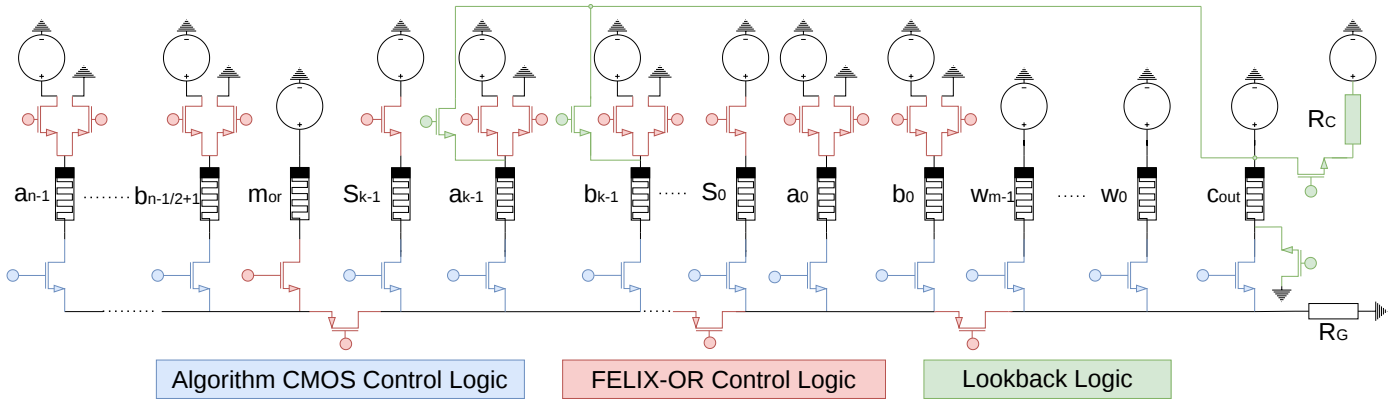

Figure 3. Circuit diagram of an n-bit RapproX_C1_OR adder with approximation level $k = n/2$ and C1 carry lookback. The NMOS and PMOS transistors represent normally open and normally closed switches, respectively. To minimize figure clutter, connecting lines are omitted. The primary functionality of the switches is indicated by color.

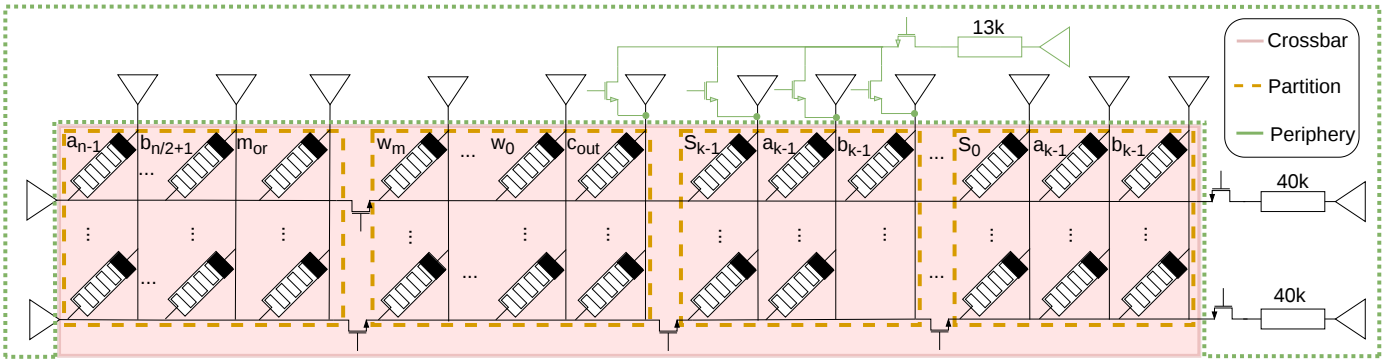

Figure 4. Crossbar array mapping of multiple n-bit RapproX_C1_OR adder with approximation level $k = n/2$ and C1 carry lookback. Transistors corresponding to the 1T1R crossbar implementation were removed to reduce clutter. Only the external periphery is shown.

Importantly, the proposed carry lookback mechanism introduces only a minor hardware overhead. The additional circuitry consists of four transistors and a single resistor located outside the crossbar array, while the single-row mapping property remains unchanged. Consequently, RapproX retains the compactness and scalability of existing adaptive approximate adders while improving arithmetic robustness in our evaluation. The total memristor count scales with the approximation level k, but remains lower than that of fully parallel implementations due to resource reuse. Overall, the proposed mapping provides an attractive trade-off between hardware cost, energy efficiency, and computational quality, making RapproX well-suited for memristive IMC systems.

4. Arithmetic-Error Analysis

This section quantifies the arithmetic quality of the RapproX designs at the adder level and situates them against SoA approximate adders under the same input assumptions. Section 4.1 defines the three error metrics used throughout the analysis and reports the uniform exhaustive sweep of all 2^{16} input combinations of an 8-bit instantiation, giving MED and MRED against SoA designs at several approximation levels. Section 4.2 then looks past these averages at three characteristics of the arithmetic error itself, namely whether the errors are biased or balanced around zero, how they depend on operand sign, and whether the conclusions drawn at $n = 8$ extend to wider adder bit widths.

4.1. Error Metrics and Comparison to SoA Approximations

In the following error analysis, we use three well-established metrics: Mean Error Distance (MED), Normalized Mean Error Distance (NMED), and Mean Relative Error Distance (MRED) [2,13,15,51,52]. Equations (5)–(7) define these metrics, which quantify the error between exact and approximate outputs. Here, $R_{E,i}$ and $R_{A,i}$ represent the i-th exact and approximate output among the full set of 2^{2n} possible n-bit sums. To avoid division by zero in Equation (7), the term corresponding to $R_{E,1}$ (when both operands are zero) is omitted.

$$MED = \frac{1}{2^{2n}} \times \sum_{i=1}^{2^{2n}} |R_{E,i} - R_{A,i}| \tag{5}$$

$$NMED = \frac{MED}{2^{n+1} - 1} \tag{6}$$

$$MRED = \frac{1}{2^{2n} - 1} \times \sum_{i=2}^{2^{2n}} \frac{|R_{E,i} - R_{A,i}|}{R_{E,i}} \tag{7}$$

Having defined the metrics, we evaluate the RapproX variants on the exhaustive sweep of all 2^{16} input combinations of an 8-bit instantiation. This corresponds to a uniform distribution over the operand space. We adopt this baseline because prior memristive approximate-adder work such as S-SINC, S-SINC+, SIAFA, SAID, and ApprOchs reports its arithmetic-error metrics under the same exhaustive sweep convention, so a uniform operand distribution is a prerequisite for a fair, apples-to-apples comparison on the same axis.

We nonetheless acknowledge that a uniform distribution is a limited assumption for application-level analysis, since many target workloads are strongly non-uniform. We revisit this limitation in Section 5.2, where the mean-energy model is extended to a workload-aware form that recovers the uniform baseline as a special case, and again in the machine-learning studies of Section 6. The evaluation is carried out in a behavior-level Python 3.9 simulator that also serves as the basis for the application-level studies in Section 6. Table 3 reports MED and MRED for both RapproX variants alongside several SoA designs at four approximation levels.

Table 3. Exhaustive sweep error metrics MED and MRED for an 8-bit adder evaluated over all 2^{16} input combinations at four approximation levels k. NMED is omitted since it is proportional to MED through NMED = MED/$(2^{n+1} - 1)$. Bold entries mark the best (lowest) value within each column.

Design	n = 8, k = 1		n = 8, k = 2		n = 8, k = 3		n = 8, k = 5	
	MED ↓	MRED ↓	MED ↓	MRED ↓	MED ↓	MRED ↓	MED ↓	MRED ↓
SAID 1 [16]	0.500	0.0213	1.250	0.0519	2.625	0.1111	10.656	0.4411
SAID 2 [16]	0.500	0.0213	1.125	0.0465	2.188	0.0932	8.529	0.3694
SIAFA2 [12]	0.250	0.0093	1.000	0.0412	2.656	0.1070	13.498	0.4539
SIAFA4 [12]	0.500	0.0186	1.250	0.0519	2.625	0.1076	10.656	0.3561
SIAFA1/3 [12]	0.250	0.0093	0.875	0.0345	2.062	0.0842	8.856	0.3369
ApprOchs [18]	0.250	0.0093	0.750	0.0305	1.748	0.0712	7.623	0.2695
S-SINC [15]	0.250	0.0093	0.750	0.0306	1.750	0.0714	7.750	0.2728
S-SINC+ [15]	0.250	0.0093	0.625	0.0253	1.375	0.0566	5.875	0.2121
RapproX_C1_OR	0.250	0.0093	0.625	0.0252	1.374	0.0565	5.783	0.2094
RapproX_C1_XOR	**0.000**	**0.0000**	**0.250**	**0.0093**	**0.749**	**0.0305**	**3.690**	**0.1431**

At low k the designs are largely indistinguishable, since the approximate region is too small to dominate the error budget. As k grows, the spread widens. RapproX_C1_OR matches S-SINC+, the strongest SoA contender, across all metrics. RapproX_C1_XOR reduces MRED by a further 31.7% at $k = 5$ and achieves the lowest values in every column. At $k = 1$, RapproX_C1_XOR is exact, since the XOR over the single approximated bit

reproduces the true sum whenever $c_{\text{in}} = 0$. Under uniformly distributed inputs, RapproX_C1_XOR is therefore the strongest design considered. As the following subsections show, however, this ranking does not fully translate to application-level behavior.

4.2. Error Characteristics

Beyond the averages, three characteristics of the arithmetic error itself shape the achieved quality, namely whether the error is biased or balanced around zero, how it depends on operand sign, and how it behaves as the adder width grows. The averages in Table 3 do not capture how individual errors are distributed across operands. Figure 5 therefore plots the histogram of error distances at $n = 8$ and $k = 5$. S-SINC and RapproX_C1_XOR are one-sided undershoots, whereas S-SINC+ and RapproX_C1_OR are balanced around zero. RapproX_C1_XOR further stands out with 35% of results being exact, against roughly 25% for the others. As Section 6.1 shows, balanced distributions often yield better image quality than the raw metrics suggest.

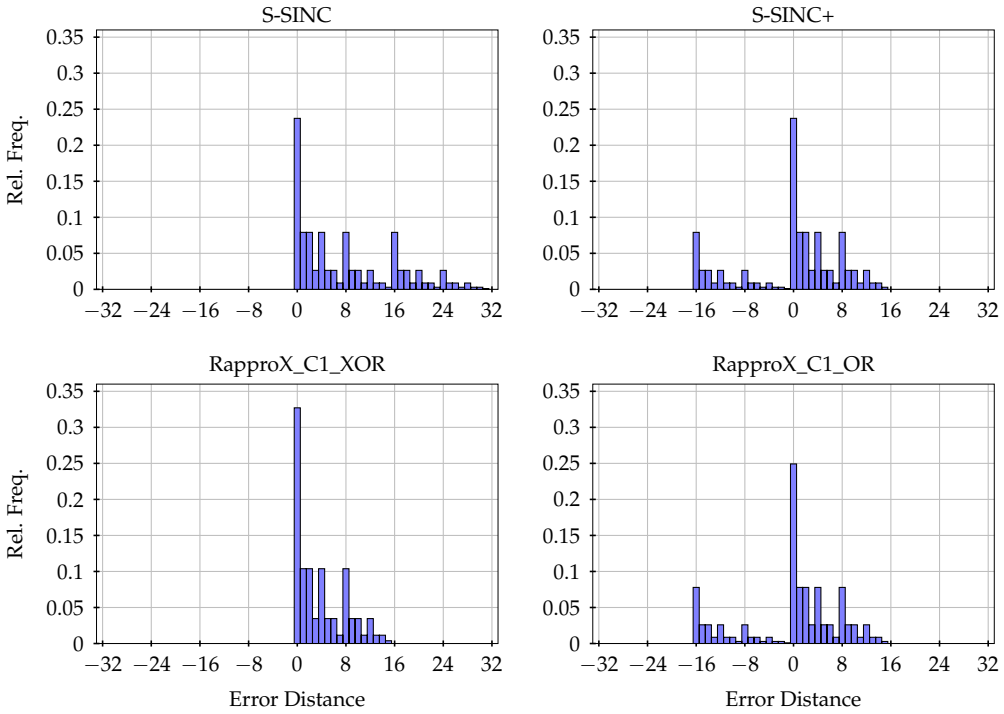

Figure 5. Distribution of error distances ($s_E - s_A$) for S-SINC, S-SINC+ and both proposed RapproX variants for $n = 8$ and $k = 5$. Positive values indicate an underestimation of the sum, while negative values represent an overestimation.

Arithmetic quality also depends on operand sign, an asymmetry that the exhaustive sweep averages hide. Prior ApprOchs work observed degraded quality on negative operands and attributed it to the APAF magnitude switch [18]. Table 4 resolves the MED of the same $n = 8$ sweep into both-negative, mixed-sign, and both-positive sub-cases to identify the actual cause.

Once at least one operand is negative, the designs group into two pairs. ApprOchs and S-SINC report the same MED, and RapproX_C1_OR and S-SINC+ report the same MED across the both-negative and mixed-sign sub-cases. S-SINC+ matches RapproX_C1_OR despite not being APAF-based, which rules out the APAF magnitude switch as the cause of the negative-operand degradation reported for ApprOchs. The pairing instead aligns with whether a design applies a carry lookback at the section boundary, so the boundary-carry mechanism is what discriminates the two pairs. RapproX_C1_XOR achieves the lowest

MED across all three sub-cases. The both-positive sub-block additionally tracks the image-blurring PSNR curves, where RapproX_C1_OR and S-SINC+ stay close through $k = 4$ and separate from $k = 5$ onward.

Table 4. Sign-resolved MED for an 8-bit adder over increasing k, separated into both-positive, both-negative, and mixed-sign operand pairs. Bold entries mark the lowest MED in each row.

Method	Design ↓				
	S-SINC	**S-SINC+**	**ApprOchs**	**RapproX_C1_OR**	**RapproX_C1_XOR**
Both Operands Positive					
$k = 1$	0.2500	0.2500	0.2499	0.2499	**0.0000**
$k = 2$	0.7500	0.6250	0.7493	0.6244	**0.2498**
$k = 3$	1.7500	1.3750	1.7432	1.3696	**0.7471**
$k = 4$	3.7500	2.8750	3.6914	2.8301	**1.7227**
$k = 5$	7.7500	5.8750	7.2656	5.5078	**3.5156**
$k = 6$	15.7500	11.8750	11.8215	8.9062	**5.8125**
$k = 7$ *	31.7500	23.8750	**0.0000**	**0.0000**	**0.0000**
Both Operands Negative					
$k = 1$	0.2481	0.2481	0.2481	0.2481	**0.0000**
$k = 2$	0.7442	0.6202	0.7442	0.6202	**0.2461**
$k = 3$	1.7364	1.3643	1.7364	1.3643	**0.7384**
$k = 4$	3.7209	2.8527	3.7209	2.8527	**1.7230**
$k = 5$	7.6899	5.8295	7.6899	5.8295	**3.6921**
$k = 6$	15.6279	11.7829	15.6279	11.7829	**7.6303**
$k = 7$	31.5039	23.6899	31.5039	23.6899	**15.5068**
Mixed-Sign Operands					
$k = 1$	0.2481	0.2481	0.2481	0.2481	**0.0000**
$k = 2$	0.7442	0.6202	0.7442	0.6202	**0.2499**
$k = 3$	1.7364	1.3643	1.7364	1.3643	**0.7493**
$k = 4$	3.7209	2.8527	3.7209	2.8527	**1.7432**
$k = 5$	7.6899	5.8295	7.6899	5.8295	**3.6914**
$k = 6$	15.6279	11.7829	15.6279	11.7829	**7.2656**
$k = 7$	31.5039	23.6899	31.5039	23.6899	**11.8125**

* The MRED vanishes for APAF-based designs, since the sign bit is zero for all positive inputs, causing the lower section to be computed exactly.

The preceding analyses fix $n = 8$, leaving open whether the same conclusions hold at wider arithmetic. Table 5 therefore reports MRED at $n = 16$, 32, and 64.

Table 5. MRED scaling of the RapproX designs at three bit widths. The bottom row uses the largest meaningful approximation level $k = n/2$, and the two rows above step back one and two bits from that maximum. Since an exhaustive sweep is infeasible at these widths, the possible operand pairs for each bit width are sampled at rate $1/s$ over their value range, with $s = 10^0$ for $n = 16$, $s = 10^{-1}$ for $n = 32$, and $s = 10^{-6}$ for $n = 64$. Higher s corresponds to denser sampling of the operand pairs.

Method	n = 16		n = 32		n = 64	
	OR ↓	**XOR ↓**	**OR ↓**	**XOR ↓**	**OR ↓**	**XOR ↓**
$k = n/2 - 2$	0.3414	0.2455	0.3585	0.2729	0.3582	0.2725
$k = n/2 - 1$	0.5552	0.4123	0.5710	0.4390	0.5707	0.4384
$k = n/2$	0.7305	0.6757	0.7462	0.6984	0.7459	0.6983

For each width, the most aggressive meaningful setting is $k = n/2$, since an n-bit adder computes at most $n/2$ approximate lower bits, and the two rows above step back one and two bits from that maximum. Read horizontally, MRED effectively stays the same across bit widths at each of these three settings, indicating that the RapproX architecture scales to larger operand widths without a notable increase in relative error.

5. Circuit-Level Evaluation

This section characterizes the RapproX_C1_OR and RapproX_C1_XOR adders at the circuit level and places them next to current SoA approximate memristive adders along the three axes that dominate in-memory arithmetic cost: energy, speed, and area. Section 5.1 describes the LT-SPICE simulation setup and the VTEAM memristor model used across all measurements. Section 5.2 derives the per-addition energy of the RapproX variants and compares it against SoA approximate adders at a representative operating point. Section 5.3 reports the processing time in discrete cycle steps, and Section 5.4 the memristor count. Section 5.5 then discusses the scope of the VTEAM-based analysis and the device- and array-level non-idealities that fall outside it. The application-level evaluation of the same designs on representative workloads follows in Section 6.

5.1. Experimental Setup

To accurately assess energy consumption, we conducted circuit simulations in LT-SPICE. These simulations utilize the reliable Voltage-controlled Threshold Adaptive Memristor (VTEAM) model [53], which has been consistently used in prior research [12,13,15,18,38], enabling direct and proper comparisons. The parameters are fitted to a real discrete Knowm memristor, which increases our confidence in the practical relevance of our circuit simulations. Similar to the differences between discrete and integrated CMOS devices, discrete memristors exhibit slower switching behavior and higher power consumption than integrated implementations. Since integrated memristive devices are not yet readily accessible, we employ measurement-fitted models to provide a realistic and practically relevant evaluation of the proposed circuits. For our proposed designs, we simulated all possible adder input combinations in LT-SPICE to measure the energy consumed by the memristors. We then averaged these measurements to determine the average energy per addition, which we linearly scaled for larger adder sizes. Since this work is based on a mixed-logic approach, we compiled the applied control voltages in Table 6, which correspond to the circuits shown in Figure 1, for a cycle time of 30 µs.

Table 6. Parameters for the utilized stateful logic forms.

Logic	V_{set} (V)	V_{cond} (V)	V_{reset} (V)	V_x (V)	R_G (kΩ)
IMPLY	1	0.9	−1	–	40
FELIX-OR	1	–	–	–	–
TMSL-AND *	1.3	0.6	–	–	13
SIXOR	–	–	−1	1.3	–

* TMSL with the specific memristor used requires only a 2 µs pulse.

5.2. Energy Consumption

The energy consumption of the RapproX architectures depends on the executed computation path and therefore strongly varies with the input operands. In particular, the design dynamically adapts its activity depending on whether the upper section of the operands contains non-zero bits. This behavior enables the omission of unnecessary computations and directly impacts the overall energy efficiency. Let a and b denote two arbitrary n-bit input operands, and let k represent the number of approximated lower bits.

Every RapproX addition first runs the FELIX-OR magnitude probe $e_{\mathrm{MAG}} = e_{\mathrm{OR}}(n-k)$ over the $n-k$ upper bits, and then takes one of two execution paths: the large-operand path e_{L} if any upper bit of either operand is set ($m_{OR} = 1$), or the short-circuit small-operand path e_{S} otherwise. The per-pair energy in nJ is therefore

$$e(n,k)_{a,b} = e_{\mathrm{MAG}} + \begin{cases} e_{\mathrm{L}}, & m_{OR} = 1 \\ e_{\mathrm{S}}, & m_{OR} = 0. \end{cases} \tag{8}$$

The per-bit costs of the underlying logic operations are obtained from LT-SPICE simulations of the circuits in Figure 1 driven by the control voltages in Table 6. FELIX-OR costs $e_{\mathrm{OR}} = 0.202\,\mathrm{nJ}$ per bit, one bit of the serial exact adder of [38] costs $e_{\mathrm{ADD}} = 4.0789\,\mathrm{nJ}$ (also reported in Table 7), the SIXOR-based XOR cell costs $e_{\mathrm{XOR}} = 0.049\,\mathrm{nJ}$ per bit, and the C1 carry lookback adds a single TMSL-AND at $e_{\mathrm{C1}} = 0.021\,\mathrm{nJ}$. On the large-operand path, the adder computes the upper $n-k$ bits exactly, runs the C1 lookback, and approximates the lower k bits, at a total cost

$$e_{\mathrm{L}} = e_{\mathrm{ADD}}(n-k) + e_{\mathrm{C1}} + e_{\mathrm{APX}}. \tag{9}$$

On the small-operand path, the upper section is skipped and only the lower k bits are computed exactly, so the total cost reduces to

$$e_{\mathrm{S}} = e_{\mathrm{ADD}}\,k. \tag{10}$$

The two RapproX variants differ only in the lower-section approximation cost e_{APX}. In the OR variant, every lower bit is computed by one FELIX-OR. In the XOR variant, the topmost approximated bit is instead computed by a SIXOR-based XOR while the remaining $k-1$ lower bits stay FELIX-OR-based. The two cases are captured by

$$e_{\mathrm{APX}} = \begin{cases} e_{\mathrm{OR}}\,k, & \text{OR variant} \\ e_{\mathrm{XOR}} + e_{\mathrm{OR}}\,(k-1), & \text{XOR variant.} \end{cases} \tag{11}$$

The mean per-addition energy under uniformly distributed inputs is obtained by weighting the two paths by their respective probabilities. The small-operand path is taken precisely when the upper sections of both operands are all zero, which corresponds to 2^{2k} of the 2^{2n} possible operand pairs, so its probability is $2^{-2(n-k)}$. Weighting Equation (8) accordingly gives the mean energy consumption

$$e(n,k)_{\bar{a},\bar{b}} = e_{\mathrm{MAG}} + \left(1 - 2^{-2(n-k)}\right) e_{\mathrm{L}} + 2^{-2(n-k)}\, e_{\mathrm{S}}. \tag{12}$$

The small-operand probability $2^{-2(n-k)}$ shrinks rapidly as the exact upper section widens. It is already only about 0.4% of all operand pairs at $n = 8$, $k = 4$, and falls further for larger $n-k$. Dropping the negligible small-operand term therefore reduces Equation (12) to $e_{\mathrm{MAG}} + e_{\mathrm{L}}$. Substituting the definitions of e_{MAG}, e_{L}, and e_{APX} into this simplified expression yields the closed forms

$$e(n,k)^{OR}_{\bar{a},\bar{b}} \approx e_{\mathrm{OR}}(n-k) + e_{\mathrm{ADD}}(n-k) + e_{\mathrm{C1}} + e_{\mathrm{OR}}\,k, \tag{13}$$

$$e(n,k)^{XOR}_{\bar{a},\bar{b}} \approx e_{\mathrm{OR}}(n-k) + e_{\mathrm{ADD}}(n-k) + e_{\mathrm{C1}} + e_{\mathrm{XOR}} + e_{\mathrm{OR}}\,(k-1). \tag{14}$$

Substituting these values yields

$$e(n,k)^{OR}_{\bar{a},\bar{b}} \approx 4.2809\,(n-k) + 0.202\,k + 0.021 \text{ nJ}, \tag{15}$$

$$e(n,k)^{XOR}_{\bar{a},\bar{b}} \approx 4.2809\,(n-k) + 0.202\,(k-1) + 0.070 \text{ nJ}. \tag{16}$$

As noted in Section 4, the uniform-input assumption is a limited baseline for application-level analysis, since many target workloads are strongly non-uniform. In ReLU-based neural networks, for instance, a large fraction of activations are exactly zero, and each such activation zeroes out an entire multiplication (see Section 6.3). To make the energy model reusable in these regimes, we replace the uniform-input path probability $2^{-2(n-k)}$ in Equation (12) by a generic small-operand-path fraction f_S that captures how often the short execution path is actually taken. This gives the workload-aware mean energy

$$e(n,k)^{f_S}_{\bar{a},\bar{b}} = e_{MAG} + (1-f_S)\,e_L + f_S\,e_S, \tag{17}$$

which reduces to Equation (12) for $f_S = 2^{-2(n-k)}$ and to the per-addition floor $e_{MAG} + e_S$ for $f_S = 1$. The corresponding theoretical energy-saving ratio relative to the exact serial adder of cost $e_{exact}(n) = e_{ADD}\,n$ is

$$\eta(n,k;f_S) = 1 - \frac{e(n,k)^{f_S}_{\bar{a},\bar{b}}}{e_{exact}(n)}. \tag{18}$$

The fraction f_S can either be estimated from the operand distribution of the workload, or measured directly during a simulation or program run, since it only requires counting how often the short execution path is entered. We exercise this workload-aware model on the quantized AlexNet on MNIST workload and validate it against the measured inference energy in Section 6.3.

Since energy consumption is dependent on technology, we report absolute values rather than relative metrics, which is the common approach in related literature [12–16]. Table 7 provides a snapshot evaluation of several SoA designs at $n = 8$ and $k = 4$. The table shows that both RapproX variants consume around 18 nJ per 8-bit addition, which is 45% less than the serial exact adder [38], 31% less than SIAFA1, 26.5% less than SAID1 [16], and 8% less than S SINC+ [15], and roughly equivalent (<1%) to [18] at the same approximation level. The RapproX designs excel at adapting to input distributions. When properly configured, they omit unnecessary computations to further improve energy efficiency. Hence, a dynamic benefit that snapshot measurements based on uniform operand distributions do not capture. This adaptability, which makes them particularly well-suited for power-sensitive applications, will be discussed in greater detail in the following chapters.

Table 7. Resulting circuit-level metrics for 8-bit adder designs evaluated at $k = 4$.

Design	Energy (nJ) ↓		Speed (Time Steps) ↓		Area (Memristors) ↓	
	e (n,k)	**e (8,4)**	**t (n,k)**	**t (8,4)**	**s (n,k)**	**s (8,4)**
Serial Exact [38] +	$4.0789n$	32.63	$22n$	176	$2n+3$	**19**
Serial Exact 2 [54] +	$4.7092n$	37.67	$23n$	184	$3n+3$	27
Serial Exact 3 [37] +	$4.5190n$	36.15	$18n$	144	$2n+4$	20
Parallel Exact [23] *	$4.0772n$	32.62	$5n+18$	58	$4n+1$	33
Semi-Serial Exact [55] *	$3.8435n+0.8053$	31.55	$10n+2$	82	$2n+6$	22
Semi-Parallel Exact [56] *	$4.8339n$	38.67	$17n$	136	$2n+3$	**19**

Table 7. *Cont.*

Design	Energy (nJ) ↓		Speed (Time Steps) ↓		Area (Memristors) ↓	
	e (n,k)	e (8,4)	t (n,k)	t (8,4)	s (n,k)	s (8,4)
SIAFA 1/3/4 [12] *	$1.71k + 4.83(n-k)$	26.16	$8k + 22(n-k)$	120	$2n+3$	**19**
SIAFA 2 [12] *	$2.51k + 4.83(n-k)$	29.36	$10k + 22(n-k)$	128	$2n+3$	**19**
SAFAN [14] *	$1.66k + 4.83(n-k)$	25.96	$7k + 22(n-k)$	116	$2n+3$	**19**
SEMIS 1 [13] *	$1.66k + 3.84(n-k) + 0.86$	22.86	$5k + 10(n-k) + 3$	63	$2n+6$	22
SEMIS 2, 3, 4 [13] *	$1.66k + 3.84(n-k) + 0.86$	22.86	$5k + 10(n-k) + 3$	63	$2n+6$	22
SINC [15] *	$0.72k + 4.83(n-k)$	22.20	$3k + 22(n-k) + 3$	103	$3k + 4(n-k) + 1$	29
SINC+ [15] *	$0.72k + 4.83(n-k) + 0.78$	22.98	$3k + 22(n-k) + 3$	103	$3k + 4(n-k) + 2$	30
PINC [15] *	$0.72k + 4.08(n-k)$	19.20	$5(n-k) + 18$	**38**	$3k + 4(n-k) + 1$	29
PINC+ [15] *	$0.72k + 4.08(n-k) + 0.78$	19.98	$5(n-k) + 18$	**38**	$3k + 4(n-k) + 2$	30
S-SINC [15] *	$0.57k + 3.84(n-k) + 1.06$	18.70	$2k + 10(n-k) + 3$	51	$2n+6$	22
S-SINC+ [15] *	$0.57k + 3.84(n-k) + 1.87$	19.51	$2k + 10(n-k) + 5$	53	$2n+6$	22
S-PINC [15] *	$0.64k + 4.83(n-k)$	21.88	$3k + 17(n-k) + 3$	83	$2n+3$	**19**
S-PINC+ [15] *	$0.64k + 4.83(n-k) + 0.92$	22.80	$3k + 17(n-k) + 2$	82	$2n+3$	**19**
SAID 1 [16] $^{+}$	$1.23k + 4.83(n-k)$	24.50	$2k + 22(n-k)$	96	$2(n-k) + 2k + 3$	**19**
SAID 2 [16] $^{+}$	$1.55k + 4.83(n-k)$	25.52	$6k + 22(n-k)$	112	$2n + k + 3$	23
ApprOchs [18]	$4.28(n-k) + 0.21k$	17.96	$22 \times \max(k, n-k) + 1$	89	$2n + k + 4$	24
RapproX_C1_OR	Equation (13)	17.98	$22 \times \max(k, n-k) + 3$	91	$2n + (k-2) + 4$	22
RapproX_C1_XOR	Equation (14)	**17.82**	$22 \times \max(k, n-k) + 3$	91	$2n + (k-2) + 6$	24

Bold items indicate the Pareto front. * Obtained from [15] with rounding for energy values. $^{+}$ Simulated with the same simulation setup used in this work via the ATOMIC tool [57].

5.3. Speed

The processing time of an n-bit RapproX adder is mainly driven by the processing time of the underlying exact adder, represented by t_{ex}. The sequential approximation process in the LSB section, which uses FELIX-OR, adds very little delay since the operation only takes one step. Prior to performing the exact addition, the FELIX-OR operation on the MSB section introduces an extra step to determine the operand magnitudes. Additionally, when the MSBs are computed exactly, another extra step is needed for the carry approximation using TMSL-AND. Thus, in the general case where the MSBs are processed exactly, and the LSBs are approximated, the total number of steps $t(n)$ is given by:

$$t(n) \leq t_{ex}(n) \times \max(k, n-k) + 3 \tag{19}$$

where t_{ex} denotes the time taken by the exact adder, n is the total adder bit width, and k specifies the number of LSB bits that are approximated, leaving $n-k$ bits in the MSB section to be computed exactly. For cases where $k \neq \frac{n}{2}$, the processing time $t(n)$ varies depending on the specific inputs. To ensure predictable processing times, the control logic must account for the worst-case scenario. Given $t_{ex} = 22$ from [56] the inequality in Equation (19) is reduced to an equality:

$$t(n) = 22n \times \max(k, n-k) + 3. \tag{20}$$

In the optimal case, where $k = \frac{n}{2}$, the equation simplifies to

$$t^{\text{opt}}(n) = \frac{1}{2} t_{ex}(n) + 3 = 11n + 3. \tag{21}$$

Table 7 presents the processing times for various SoA designs for $n = 8$ and $k = 4$. We report relative speed as discrete time steps instead of absolute latency, as the latter depends on the specific technology used. This approach is common in the literature, where performance is typically measured in cycle counts rather than absolute time, ensuring that our comparisons remain valid even as new technologies emerge [12,13,15–17]. Owing to its use of serial exact addition, the RapproX design delivers moderate yet comparable speed compared to other SoA designs. In terms of timing, RapproX is faster than the SAID [16]

and SIAFA [12] families, but is outperformed by the PINC/SINC [15] family—most notably by S SINC+. However, many alternatives tend to require significantly more area or energy. Overall, S SINC+ stands out as the strongest contender—it is significantly faster, albeit with slightly higher energy consumption than RapproX. However, it is important to note that while this trade-off might seem unfavorable at first, the uniform distribution used for the evaluation prevents RapproX from fully exploiting its power-saving potential, which will be discussed shortly. Quantitatively, at $n = 8$ and $k = 4$ RapproX completes an addition in 91 time steps, which is 48.3% fewer than the serial exact adder [38] at 176 steps and 24.2% fewer than SIAFA1 [12] at 120 steps, and about matches ApprOchs [18] at 89 steps. At the other extreme, the parallel PINC and PINC+ designs [15] are the fastest at 38 steps, which they buy with higher memristor usage and energy. RapproX currently uses more steps than S SINC+ [15] at 53 steps. By replacing the serial exact adder with the faster semi-serial exact adder that S SINC+ builds on, RapproX could reach the speed of S SINC+ while maintaining its energy efficiency, at the cost of increased complexity of the control logic.

5.4. Area Usage

The required area of the RapproX designs can be summarized by the accumulation of the $2n$ input memristors, the carry memristor, the work memristors required by the exact adder algorithm [38] (typically 2–4), and the magnitude sensing memristor m_{OR}. The XOR variant additionally requires 2 stabilization memristors for the SIXOR circuit [11]. When $k > 2$, an extra $k - 2$ memristors are required to approximate the sum bits in the lower section, with two work memristors from the exact adder repurposed to store the FELIX-OR results. The required memristors of the OR and XOR variants can be described through

$$s_{OR}(n) = 2n + (k - 2) + 4 \tag{22}$$

$$s_{XOR}(n) = 2n + (k - 2) + 6 \tag{23}$$

RapproX_C1_OR exhibits a moderate area footprint compared to the parallel exact adder [23] and the SINC, SINC+, PINC, and PINC+ architectures [15]. The RapproX_C1_XOR variant requires a comparable number of memristors to these parallel approaches [15,23], while avoiding the requirement for multirow computation. Both variants incur a larger area overhead than highly compact designs such as the SIAFA family [12] and the S-PINC/S-PINC+ architectures [15]. For consistency with prior work, the comparison is limited to memristor count and does not include peripheral circuitry or layout overhead, as these metrics are not commonly reported by SoA designs. Quantitatively, at $n = 8$ and $k = 4$ RapproX_C1_OR uses 22 memristors and RapproX_C1_XOR uses 24, where the two additional memristors of the XOR variant stem from the SIXOR cell it employs for the topmost approximated bit. Both counts fall in the middle of the design space. The most compact designs, such as the SIAFA family [12] and the serial exact adder [38], use 19 memristors, whereas the parallel exact adder [23] uses 33 and the SINC and PINC families [15] use 29 to 30. This moderate count follows from reusing the exact adder work memristors for the lower section. Relative to the closest adaptive design ApprOchs [18] at 24 memristors, RapproX_C1_OR uses two fewer and RapproX_C1_XOR matches it.

All evaluated architectures require the standard peripheral circuitry commonly employed in memristive stateful-logic IMC systems, including row/column decoders, selectors (switching networks), voltage drivers for the required logic operations, and conventional read/write circuitry. Since these components are largely shared across architectures and are generally not reported quantitatively in the literature, they are excluded from the area comparison. Relative to these common peripherals, the proposed RapproX architecture introduces only a negligible additional overhead consisting of the carry lookback circuitry, which requires four transistors and one resistor located outside the crossbar array while

preserving the single-row mapping of the design. Therefore, memristor count remains an appropriate and fair comparison metric for the evaluated state-of-the-art designs.

5.5. Practical Considerations and Limitations

The circuit-level evaluation is performed using the widely adopted VTEAM behavioral model, enabling fair and reproducible comparisons with prior memristive approximate adders under a common simulation framework. More importantly, the proposed RapproX methodology is independent of the underlying memristor technology and can be implemented using any stateful logic-compatible device. Different memristor technologies exhibit significantly distinct switching voltages, pulse durations, resistive states, and variability characteristics, which influence the cycle time, voltage requirements, and energy consumption. Consequently, the reported energy values should be interpreted primarily as comparative metrics between approaches rather than technology-specific hardware measurements. Practical implementations are further affected by device-to-device and cycle-to-cycle variability, stochastic switching behavior, resistance drift, retention loss, and finite endurance, all of which may influence the reliability and energy efficiency of stateful logic operations. Likewise, large-scale crossbar arrays introduce additional non-idealities such as sneak-path currents, interconnect resistance, parasitic capacitances, and voltage drops, whose impact increases with array size. Since the objective of this work is to introduce and evaluate the proposed carry lookback mechanism and its architectural implications, rather than to comprehensively analyze technology-specific device and array non-idealities, these effects are beyond the scope of the present study. Such investigations are particularly relevant as memristive technologies continue to evolve rapidly and represent an important direction for future research.

6. Application Workloads

We evaluate the RapproX designs on four workloads chosen to cover the typical stages of an edge-AI pipeline. Section 6.1 covers image blurring, a smoothing primitive commonly used as a preprocessing step upstream of edge-AI models, and reports pixel-level quality against approximation level k. Section 6.2 covers k-means clustering as a representative classical, non-neural machine-learning workload whose iterative centroid updates expose the interaction between approximation errors and convergence. Section 6.3 covers quantized AlexNet on MNIST as a compact CNN that keeps the full inference tractable and serves as our detailed energy and quality case study. Section 6.4 then evaluates CIFAR-10 across ResNet, DenseNet, Inception-V3, and VGG as a transfer test that probes how these observations carry over to a range of standard neural-network topologies representative of edge inference. Section 6.5 closes by isolating the application-level accuracy contribution of the C1 carry lookback against a no-lookback baseline.

6.1. Image Blurring

In this subsection, we evaluate the performance of the RapproX designs using the image blurring application, which is a common evaluation method to show the scalability of the approximate adders to more complex workloads [15,58]. Image blurring is performed using a Gauss kernel K, whose positive weights smooth the image and reduce high-frequency noise.

$$K = \begin{bmatrix} 1 & 2 & 1 \\ 2 & 4 & 2 \\ 1 & 2 & 1 \end{bmatrix} \tag{24}$$

The required convolution multiplies K with the corresponding image patch element-wise, sums the products, and normalizes by the kernel's weight sum. Multiplications

are realized as binary long multiplication and reuse the same approximate adder as the summation, so a single 16-bit RapproX instance carries both the partial-product accumulation and the final pixel-sum at the same approximation level k. Per-image energy is obtained by scaling the per-addition energy of Equation (12) by the addition count of the long-multiplication-and-accumulate over the full image, so the energies reported below are stated in mJ rather than the nJ unit of Section 5. The evaluation is based on a dataset of 100 grayscale images (256 × 192 pixels), sampled, resized, and converted from Google's OpenImagesV7 training set [59]. We process these images at various approximation levels k, comparing outputs from the exact adder ($k = 0$) against those from approximate adders ($k > 0$). Image quality is quantified using PSNR, which measures pixel intensity deviations, and the SSIM, which assesses the preservation of structural details. Figure 6 illustrates image blurring using an exact adder ($k = 0$) alongside the proposed RapproX_C1_OR adder with progressively increasing approximation levels. At low levels, the output closely matches the exact result. however, starting at $k = 4$, visual quality begins to deteriorate further as k increases. Despite the decline in image quality, the overall scene remains recognizable, even under the most extreme approximations. This outcome aligns closely with the measurements of median image quality presented in Table 8. They indicate that RapproX_C1_OR sustains high median image quality (>40 dB) up to $k = 5$, a performance that surpasses most other designs.

(**a**) Original (**b**) Exact (**c**) 1-Bit Ax. (**d**) 2-Bit Ax. (**e**) 3-Bit Ax.

(**f**) 4-Bit Ax. (**g**) 5-Bit Ax. (**h**) 6-Bit Ax. (**i**) 7-Bit Ax. (**j**) 8-Bit Ax.

Figure 6. Image quality degradation in image blurring application using 16-bit RapproX_C1_OR with increasing approximation level k: original image, blur using an exact adder ($k = 0$), and blurs with progressively higher k.

For the Gaussian-blurring benchmark considered here, this suggests that higher approximation levels can be used to reduce energy consumption without substantially compromising image quality. We therefore identify $k = 1$ to $k = 5$ as the practical range of k on this workload, within which a higher k lowers energy consumption while keeping image quality high. Similar results were obtained for RapproX_C1_XOR, which only at higher approximation degrees degrade noticeably compared to the OR approach. Please note that RapproX_C1_OR leads to a PSNR result above 31 dB for $k = 7$, while RapproX_C1_XOR results in 5 dB less. This validates our initial analysis proposed in Section 4, that not only error metrics but also the distribution is significant at the application-level.

Table 8. Median image quality (PSNR and SSIM) and energy consumption (mJ) for 16-bit image blurring using the RapproX designs at varying approximation levels k.

Method	RapproX_C1_OR			RapproX_C1_XOR		
	PSNR [dB] ↑	SSIM ↑	Energy [mJ] ↓	PSNR [dB] ↑	SSIM ↑	Energy [mJ] ↓
Exact	∞	1.000	240.36	∞	1.000	240.36
$k = 1$	62.18	0.999	217.56	∞	1.000	217.11
$k = 2$	54.51	0.998	187.82	58.28	0.999	187.42
$k = 3$	52.41	0.997	162.14	50.83	0.998	161.80
$k = 4$	48.55	0.994	142.39	43.80	0.995	142.11
$k = 5$	42.52	0.982	128.69	36.90	0.986	128.47
$k = 6$	36.59	0.951	121.98	31.26	0.960	121.84
$k = 7$	31.33	0.905	123.88	26.33	0.912	123.81
$k = 8$	26.91	0.870	134.34	22.78	0.862	134.30

Figure 7 presents a comparison of various SoA designs based on energy consumption. Across Figures 7–9, the first configuration ($k = 1$) of every curve is marked with a rectangle and its last configuration ($k = 8$) with a diamond. While ApprOchs achieves the lowest energy consumption, RapproX matches it, exhibiting a similar non-linear trend caused by the APAF mechanism, which the three designs RapproX_C1_OR, RapproX_C1_XOR, and ApprOchs share. While their energy curves coincide by construction, the designs' differences become apparent in Figures 8 and 9. On the other hand, S-SINC+ demands significantly more energy. Figure 8 evaluates these designs with respect to image quality, revealing that both RapproX variants follow a steady, nearly linear trend. Starting at PSNR levels above 60 dB, both variants consistently maintain robust signal fidelity. Notably, at $k = 1$, RapproX_C1_XOR achieves perfect image quality by XOR-ing the least significant bit. Moreover, RapproX_C1_OR not only matches but even slightly exceeds the image quality of the competitive S-SINC+ design at higher approximations. In fact, none of the designs listed in Table 7 outperform RapproX_C1_OR in terms of image quality.

Finally, Figure 9 merges these insights by plotting achieved PSNR against energy consumption, yielding a measure of energy efficiency. The plot shows that the RapproX family provides a Pareto front where the RapproX_C1_XOR configuration is dominant at $k = 1$ and $k = 2$, while the RapproX_C1_OR configuration leads from $k = 3$ to $k = 6$. This highlights the improved energy efficiency of RapproX in this benchmark — even against designs already known for high energy efficiency like ApprOchs. RapproX_C1_OR achieves the high image quality of S-SINC+ while matching the low energy consumption of ApprOchs. Notably, the RapproX_C1_OR configuration at $k = 5$ is especially promising for the image blurring application, delivering excellent image quality (approximately 42.5 dB PSNR and 0.982 SSIM) while reducing energy consumption by 46.5% compared to the exact adder. It also uses 9.3% less energy than its sibling RapproX_C1_XOR, 20.2% less than ApprOchs (at $k = 3$), and 30.9% less than S-SINC+ (at $k = 5$), all while maintaining comparable image quality around 43 dB PSNR. This evaluation illustrates the power-saving potential of RapproX for the evaluated blurring benchmark, derived from its adaptability to its inputs. Consequently, the speed-energy trade-off compared to S-SINC+ becomes more favorable. The remaining applications turn from the assumed fixed-pattern image blurring benchmark of Section 6.1 to machine-learning workloads with highly non-uniform input distributions. The first two studies focus on the RapproX_C1_OR variant, allowing a direct comparison against existing memristive approximate adders that employ OR-based approximation mechanisms [15,18]. The CIFAR-10 study additionally includes RapproX_C1_XOR to assess the impact of both proposed lookback variants on more complex deep learning workloads. Together, these experiments provide a comprehensive evaluation of the effectiveness of carry-aware adaptive approximation across diverse machine-learning scenarios.

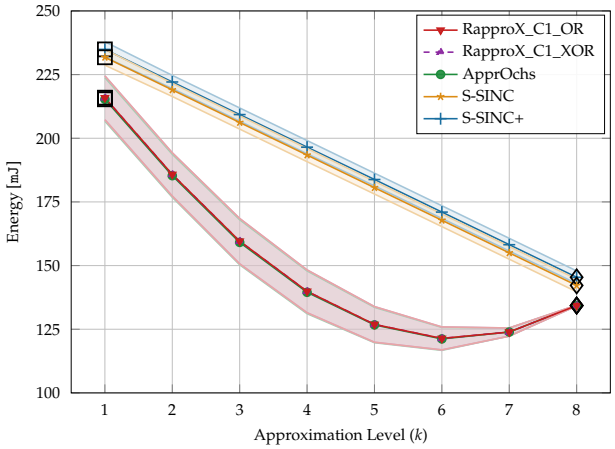

Figure 7. Mean energy consumption per image for RapproX and SoA designs vs. approximation level k. Shaded regions indicate one standard deviation from the mean. The first and last configuration of each curve are highlighted by a rectangle and a diamond, respectively. The three APAF-based designs coincide in energy by construction. Their differences become apparent in Figures 8 and 9.

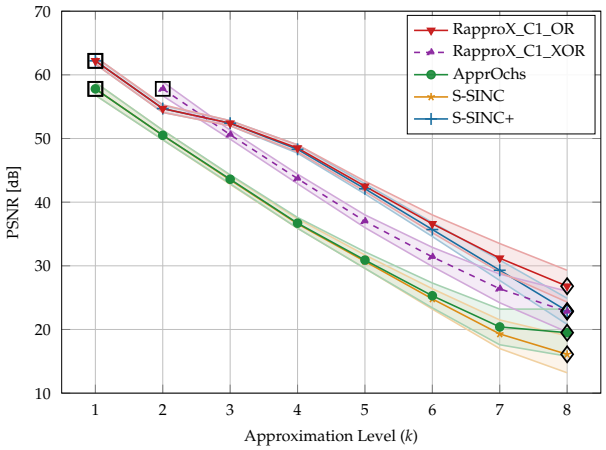

Figure 8. Mean PSNR per image for RapproX and SoA designs vs. approximation level k. Shaded regions indicate one standard deviation from the mean. The first and last configuration of each curve are highlighted by a rectangle and a diamond, respectively. Both RapproX variants degrade almost linearly.

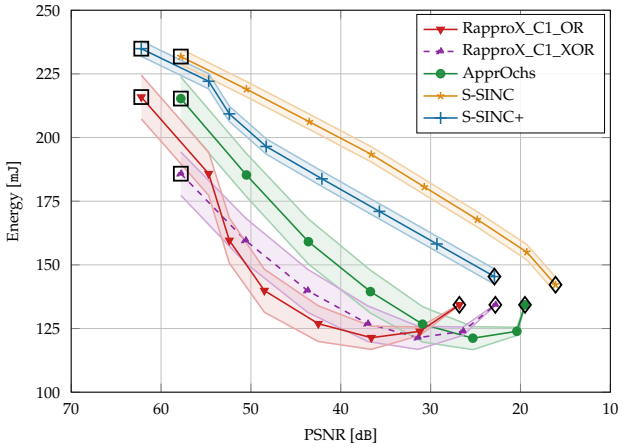

Figure 9. Mean energy consumption per image for RapproX and SoA designs vs. mean PSNR, yielding energy efficiency. Note that the PSNR axis is reversed and image quality decreases from left to right. Shaded regions show one standard deviation from the mean. The first and last configuration of each curve are highlighted by a rectangle and a diamond, respectively. RapproX_C1_XOR at $k = 1$ produces exact output and is therefore off the axis, so its trajectory begins at $k = 2$.

6.2. k-Means Clustering

We assess k-means clustering, a simple yet effective algorithm for unsupervised machine learning. Our implementation employs k-means++ for centroid initialization and uses the Manhattan distance metric, which removes the need for a custom square root function. The synthetic test data, with known ground-truth labels, consists of 36 samples, each containing 400 vectors of 8-bit signed integers. The dataset includes Gaussian blobs with varying cluster centers and variances, along with anisotropically distributed samples and ring structures.

To isolate approximation effects, we first run k-means using an exact adder to establish a fair ground truth, since generated labels can yield outcomes impossible to attain for k-means, even with an exact adder. The approximate runs then use the same initial centroids as the exact run, ensuring that any deviations arise solely from the approximation. We report clustering accuracy relative to this ground truth alongside energy consumption for each approximation level. The k-means algorithm is permitted a maximum of 30 iterations per synthetic example. If convergence is not reached within this limit, the algorithm terminates, and the current results are evaluated as is. Because summing hundreds of vector positions can yield large numbers, we simulate an approximate 32-bit adder in this experiment.

Table 9 shows the performance of the RapproX_C1_OR design across different approximation levels k. As k increases, the average number of iterations required for convergence also rises, particularly beyond $k = 4$, due to reduced arithmetic quality. Additionally, higher k values lead to greater mean energy consumption. Although RapproX consumes less energy per operation compared to an exact adder, the additional iterations counteract these savings. Note that the high standard deviation in energy consumption reflects the variability in convergence, as some experiments complete in just a few iterations while others reach the iteration limit, making energy consumption highly case-dependent.

Table 9. Mean accuracy (%), energy consumption (mJ), and iteration count are measured for k-means clustering with a 32-bit RapproX_C1_OR adder at different k. Mean values are reported with standard deviations using the $\pm$ notation.

Method	Accuracy [%] ↑	Energy [mJ] ↓	Iterations ↓
Exact [38]	100.00 ± 0.00	6.46 ± 3.31	7.92
$k = 1$	98.19 ± 0.05	6.48 ± 3.48	8.28
$k = 2$	97.31 ± 0.06	7.83 ± 6.10	10.50
$k = 3$	95.55 ± 0.09	6.50 ± 4.75	9.03
$k = 4$	92.42 ± 0.08	6.32 ± 5.87	9.11
$k = 5$	86.80 ± 0.13	7.32 ± 6.25	12.17
$k = 6$	83.18 ± 0.14	8.87 ± 6.56	17.00
$k = 7$	69.44 ± 0.13	9.32 ± 5.69	20.58
$k = 8$	46.01 ± 0.17	3.56 ± 3.30	8.64

Figure 10 shows examples from the synthetic test data, each paired with its modified ground truth and classifications at three approximation levels, along with their respective accuracies. As expected, accuracy decreases with increasing k, though the extent varies with the data distribution. In rows two through four, where clusters are in close proximity, accuracy drops significantly because arithmetic errors hinder clear separation. In contrast, rows one and five, featuring well-separated clusters, still yield high accuracy even at higher approximation levels. This sensitivity to cluster separability poses a notable challenge, especially since k-means is typically applied in unsupervised settings without human-annotated ground truth.

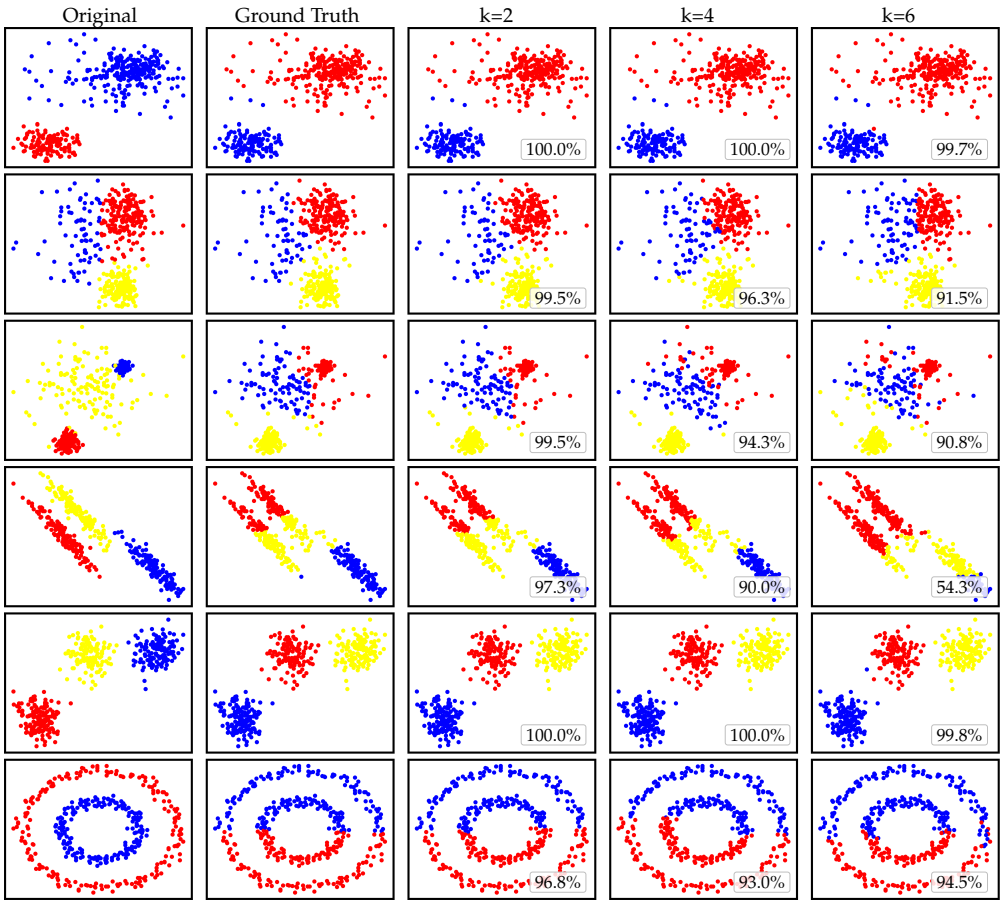

Figure 10. Examples of k-means clustering with a 32-bit RapproX_C1_OR adder at different k. The left column shows the generated ground truth. The second column shows the modified ground truth by applying exact k-means. Clustering results are indicated by color and are given with accuracy relative to the modified ground truth.

In summary, we observe an increase in the mean iteration count, a decline in mean accuracy, a strong dependence of classification quality on the data distribution, and a lack of consistent energy savings. In fact, we see those trends for all examined designs, albeit with noticeable fluctuations in accuracy and iteration count. Together, these findings suggest that approximate designs may not be well-suited for open-ended iterative algorithms that rely on convergence. In such algorithms, arithmetic errors might tend to hinder rather than facilitate convergence. This stands in stark contrast to fixed-pattern applications like image processing or MNIST digit recognition, where increasing the approximation levels merely leads to reduced accuracy and energy consumption. However, in convergence-based settings, approximation may change the underlying dynamics of the algorithm and therefore does not necessarily lead to reduced energy consumption or faster overall computation. Since the benefit here depends on the data distribution rather than on k alone, a general range of k for k-means clustering is of limited practical value.

6.3. Quantized AlexNet on MNIST

Although AlexNet [60] is an early convolutional neural network, its architecture, with multiple convolutional layers followed by dense fully connected layers, remains a good foundational baseline. These components continue to influence modern computer vision models as well as applications in time series analysis and natural language processing. We evaluate both inference accuracy and energy consumption using 1000 randomly selected samples from the MNIST dataset. Employing the original 8-bit quantized model

weights, we substitute the standard convolutional and dense layer operations with custom implementations based on behavioral simulations of our proposed adder design. This setup allows us to directly attribute any performance degradation to approximation and arithmetic errors. In our simulation, multiplications are executed using a 16-bit adder with binary long multiplication. As the operands in both convolutional and dense layers are 8-bit, their products remain within a 16-bit range. However, the subsequent accumulation in these layers can quickly exceed this limit, necessitating the use of a 32-bit adder for summation. Both adders operate at the same approximation level.

Table 10 summarizes the results for S SINC+, ApprOchs, and RapproX_C1_OR. While ApprOchs technically demonstrates the lowest energy consumption, it is followed closely by RapproX_C1_OR, which consumes only 0.2% more energy due to its additional carry approximation. However, ApprOchs exhibited lower stability, with its accuracy deteriorating considerably as k increases. Considering the minute energy disparity and the similar design principle, we will focus on the RapproX_C1_OR design in the subsequent analysis. To obtain 98.9% accuracy, RapproX_C1_OR (at $k = 1$) consumes approximately 530mJ per image classification, only 12.7% of the energy required by an exact computation, which is around 4200 mJ, and just 35% of that used by S SINC+ (at $k = 8$). In fact, even its highest energy consumption remains below the minimum observed for S SINC+. Notably, both designs maintain an inference accuracy of around 99% for all approximation levels. However, the low-approximation configuration of RapproX_C1_OR is the most energetically favorable. The practical range of k on this workload therefore spans $k = 1$ to $k = 8$ for RapproX_C1_OR, whereas ApprOchs drops to 82.6% at $k = 7$ and 27.1% at $k = 8$ so that its practical range is limited to $k = 1$ to $k = 6$. The carry approximation enlarges the practical range compared to ApprOchs in this use case, and the whole range stays near exact accuracy at a large energy saving over the exact computation.

Table 10. Inference accuracy (%) and mean energy consumption per MNIST classification are measured for the S-SINC+, ApprOchs, and RapproX_C1_OR designs. Mean values are reported with standard deviations using the ± notation.

Method	S-SINC+ [15]		ApprOchs [18]		RapproX_C1_OR	
	Acc. [%] ↑	Energy [mJ] ↓	Acc. [%] ↑	Energy [mJ] ↓	Acc. [%] ↑	Energy [mJ] ↓
Exact	98.9	4177.9 ± 0.0	98.9	4177.9 ± 0.0	98.9	4177.9 ± 0.0
$k = 1$	98.9	2265.7 ± 0.0	98.9	532.4 ± 64.7	98.9	533.2 ± 64.9
$k = 2$	98.9	2160.9 ± 0.0	98.9	625.5 ± 57.5	98.9	626.7 ± 56.7
$k = 3$	99.0	2056.2 ± 0.0	98.9	717.5 ± 50.1	99.0	719.2 ± 50.3
$k = 4$	99.0	1951.5 ± 0.0	98.9	808.3 ± 42.3	98.9	811.5 ± 42.9
$k = 5$	99.1	1846.7 ± 0.0	98.7	896.8 ± 33.7	99.1	902.8 ± 35.1
$k = 6$	99.1	1742.0 ± 0.0	97.2	988.0 ± 25.9	99.1	995.5 ± 27.7
$k = 7$	99.1	1637.3 ± 0.0	82.6	1083.7 ± 19.2	99.1	1090.7 ± 20.8
$k = 8$	98.9	1532.5 ± 0.0	27.1	1182.9 ± 13.3	99.0	1188.7 ± 14.5

A major contributor to RapproX's efficiency is its ability to leverage the network's data characteristics. The Rectified Linear Unit (ReLU) activation function zeroes out negative outputs, so many zeroes enter the subsequent convolutional layer. In the tested network, approximately 90% of activations are zero while only 10% are positive, and the weights are normally distributed around zero, often taking values of zero, one, or other small integers. Every RapproX addition takes one of two paths. On the small path, both upper sections are zero, so the upper part is skipped and only the lower k bits are computed exactly. On the large path, at least one upper bit is set, so the upper part is computed exactly and the lower k bits are approximated. A zero activation zeroes an entire binary-long-multiplication inner loop, so the small path is taken far more often than under uniform inputs and lets RapproX omit substantial portions of the computation. Measured directly on one full inference

of roughly 32 M additions per image, the small-path fraction reaches $f_S^{\text{mult}} \approx 92\%$ on the $n = 16$ multiplier stage across its 26.44 M additions per image and $f_S^{\text{acc}} \approx 83\%$ on the $n = 32$ accumulator stage across its 5.56 M additions per image.

This fraction is set by the activation sparsity of the network rather than by k, so it stays near these values across the entire sweep. On the dominant small path, the lower k bits are computed exactly, which is why accuracy holds near 99% at every k. What k controls is only the per-addition cost of the small path, $e_S = e_{\text{ADD}}\, k$ in Equation (10), which grows linearly in k. A larger k therefore leaves the path split and the accuracy essentially unchanged while raising the cost of the most frequently taken path, so within the practical range, low k minimizes energy and preserves accuracy at the same time. On a sparse workload of this kind, the APAF magnitude skip is therefore the driving factor behind the efficiency gain, since it removes the entire upper computation on the vast majority of additions, while the carry lookback contributes only a small refinement at $k = 1$ because the dominant small path never approximates and never invokes the boundary carry. The lookback instead becomes the decisive factor on the non-sparse workloads of Section 6.1, where the large path is taken almost everywhere, and the practical range extends to larger k. The small-path fraction also validates the workload-aware energy model of Section 5.2 on non-uniform data. Substituting the two per-stage fractions into Equation (17) and weighting by the addition counts of each stage gives the closed-form per-image energy predictions in Table 11. Across the full sweep $k = 1, \ldots, 8$, the prediction lies within about 1.2 percentage points of the measured savings ratio, and at $n = 16$, $k = 1$ it predicts $\eta \approx 88\%$, in close agreement with the 87% end-to-end saving measured in Table 10. This is also why the energy of RapproX rises with k while that of S-SINC+ falls. S-SINC+ has no skip path and computes the full exact upper section on every addition, so raising k shifts more bits into the cheaper approximate region and lowers its energy, the opposite of the small-path cost trend that governs RapproX.

Table 11. Closed-form per-image energy prediction of Equation (17) versus the measured AlexNet-MNIST energies from Table 10, evaluated at the empirical per-stage short-circuit fractions $f_S^{\text{mult}} \approx 92\%$ and $f_S^{\text{acc}} \approx 83\%$. All energies are per classification. η is the savings ratio relative to the exact serial adder (4177.9 mJ).

Method	Energy [mJ]		Savings Ratio η [%]		$\lvert\Delta\eta\rvert$ [pp] ↓
	E_{pred} ↓	E_{meas} ↓	η_{pred} ↑	η_{meas} ↑	
$k = 1$	482.6	533.2	88.45	87.24	1.21
$k = 2$	581.6	626.7	86.08	85.00	1.08
$k = 3$	679.4	719.2	83.74	82.79	0.95
$k = 4$	777.4	811.5	81.39	80.58	0.82
$k = 5$	874.9	902.8	79.06	78.39	0.67
$k = 6$	973.1	995.5	76.71	76.17	0.54
$k = 7$	1073.5	1090.7	74.31	73.89	0.41
$k = 8$	1176.3	1188.7	71.85	71.55	0.30

6.4. Quantized Networks on CIFAR10

To broaden our examination of arithmetic quality, we evaluated four more complex model families, as summarized in Table 12. These models include ResNet, DenseNet, Inception-V3, and VGG, and represent diverse and more intricate architectures compared to the simpler AlexNet. More specifically, ResNet leverages residual connections to enable the training of much deeper networks, DenseNet employs dense connectivity for extensive feature reuse across the layers of the network, Inception-V3 performs parallel multi-scale convolutions for computational efficiency and rich feature extraction, and VGG achieves great network depths by stacking numerous small 3x3 convolution filters. In contrast,

AlexNet consists of merely five convolutional layers, followed by three fully connected layers with ReLU activation between them. This architectural difference translates to a significantly lower computational demand for AlexNet at 0.77 Giga MACs (GMACs), whereas the other models range from DenseNet169's 3.4 GMACs to VGG19bn's substantial 19.7 GMACs.

Table 12. Comparison of CIFAR-10 classification accuracy across ten neural-network architectures for varying approximation levels k using RapproX_C1_XOR and RapproX_C1_OR compared to ApprOchs [18]. Cell shading encodes the accuracy drop relative to each column's exact-adder baseline. Cells that remain within a few percentage points of the baseline stay white, while progressively larger drops shade toward salmon. A taller red band therefore indicates a design that degrades over a wider range of k.

Method	Networks ↑									
	ResNet18	**ResNet34**	**ResNet50**	**DenseNet121**	**DenseNet161**	**DenseNet169**	**Inception-V3**	**VGG13bn**	**VGG16bn**	**VGG19bn**
Exact	92.98	93.29	93.63	93.84	93.48	93.70	66.80	94.15	93.97	93.75
RapproX_C1_XOR										
k = 1	92.96	93.30	93.56	93.90	93.43	93.74	66.69	94.02	93.93	93.77
k = 2	92.96	93.29	93.62	93.84	93.45	93.73	66.65	94.08	93.97	93.77
k = 3	92.93	93.27	93.56	93.84	93.48	93.73	66.58	94.09	93.88	93.74
k = 4	92.88	93.26	93.61	93.81	93.45	93.68	66.70	94.09	93.97	93.77
k = 5	92.73	93.19	93.49	93.73	93.40	93.69	66.32	94.12	93.89	93.77
k = 6	92.24	93.01	93.11	93.58	92.98	93.55	65.32	93.97	93.85	93.66
k = 7	90.30	91.76	90.38	92.06	91.54	92.18	60.73	93.42	92.95	92.81
k = 8	82.91	88.14	74.85	90.31	84.67	85.09	37.88	90.17	87.02	88.53
RapproX_C1_OR										
k = 1	92.99	93.21	93.55	93.95	93.47	93.78	66.64	94.03	93.92	93.82
k = 2	92.96	93.24	93.55	93.91	93.47	93.73	66.68	94.09	93.98	93.78
k = 3	92.93	93.24	93.58	93.91	93.43	93.77	66.75	94.08	93.88	93.75
k = 4	92.79	93.20	93.55	93.90	93.48	93.78	67.06	94.10	93.80	93.75
k = 5	92.47	93.06	93.05	93.64	93.01	93.58	66.71	94.08	93.59	93.58
k = 6	91.44	92.03	90.71	92.64	92.13	92.41	65.53	93.65	93.14	93.13
k = 7	86.61	86.99	77.82	88.24	86.98	87.88	59.56	92.44	91.15	91.57
k = 8	52.54	52.02	34.85	47.20	44.38	43.89	37.48	82.90	77.55	83.47
ApprOchs [18]										
k = 1	92.98	93.27	93.58	93.96	93.50	93.72	66.69	94.17	93.92	93.80
k = 2	92.96	93.27	93.59	93.90	93.45	93.65	66.74	94.06	93.92	93.73
k = 3	92.98	93.25	93.64	93.93	93.44	93.77	66.57	94.07	94.00	93.78
k = 4	92.95	93.15	93.50	93.77	93.30	93.71	66.07	94.03	93.89	93.77
k = 5	92.45	92.43	92.42	93.03	92.54	93.17	64.11	93.85	93.74	93.58
k = 6	88.31	87.66	83.11	88.71	88.10	88.86	52.25	92.74	91.63	91.81
k = 7	66.33	58.83	34.74	63.40	58.69	57.39	15.97	83.69	73.00	76.63
k = 8	21.65	11.82	10.05	18.03	16.64	13.62	9.99	30.13	11.16	13.35

The results in Table 12 indicate that the proposed carry lookback mechanism can have a substantial impact on application-level robustness. Across the evaluated CIFAR-10 architectures, both RapproX variants maintain near-exact classification accuracy up to and including $k = 6$, despite the increasing degree of approximation. This behavior is consistent across the evaluated model families, including ResNet, DenseNet, Inception-V3, and VGG, suggesting that the observed improvements are not confined to a specific network architecture within the evaluated set and likely stem from the underlying arithmetic properties of the proposed design. A comparison with ApprOchs highlights the effectiveness of the lookback mechanism. While all designs exhibit comparable accuracy for small approximation levels, the differences become increasingly pronounced as k grows. At $k = 6$, ApprOchs already experiences noticeable degradation across most models, whereas both RapproX variants remain close to exact inference. The gap widens substantially at $k = 7$ and $k = 8$, where RapproX consistently outperforms ApprOchs across all model families. This trend is

further illustrated in Figure 11, which summarizes the accuracy drop of all networks of the proposed designs and ApprOchs [18], compared to the exact baseline. Particularly for the ResNet and DenseNet families, the proposed adders retain a large fraction of the original model accuracy while ApprOchs undergoes a much steeper decline. This asymmetry is also directly visible in the cell shading of Table 12, where each cell is shaded by its accuracy drop relative to the column's exact-adder baseline. For ApprOchs, the shaded band spans both $k = 7$ and $k = 8$, so the design degrades noticeably one full approximation step earlier than either RapproX variant. RapproX_C1_OR shades in only at $k = 7$ and $k = 8$, and with a markedly lighter tone at $k = 7$ than ApprOchs. RapproX_C1_XOR shades in only at $k = 8$ and stays close to the exact baseline everywhere else. The difference between these bands is precisely the accuracy that the C1 carry-lookback mechanism preserves by predicting the boundary carry rather than dropping it.

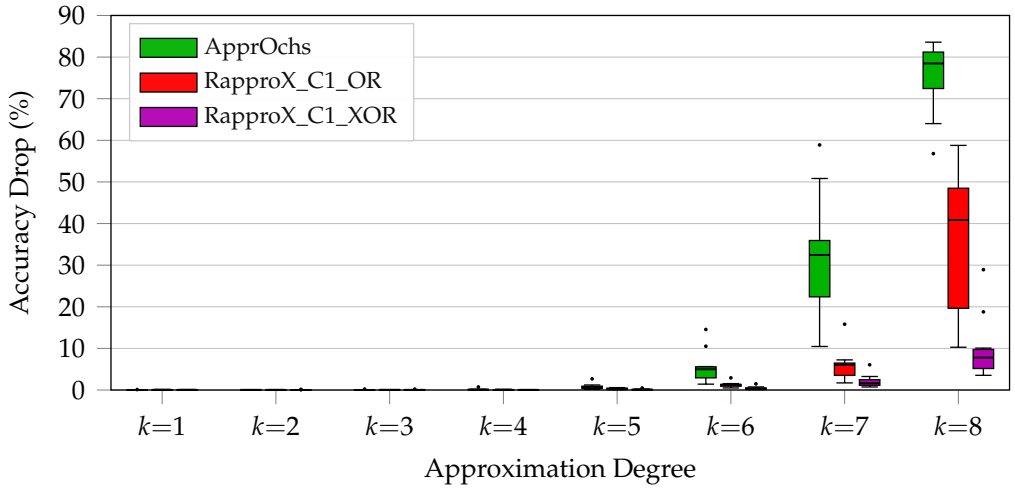

Figure 11. Accuracy drop (in percentage points) relative to the exact-adder reference for RapproX_C1_OR, RapproX_C1_XOR, and ApprOchs [18], summarized as boxplots over the ten CIFAR-10 models of Table 12.

These observations indicate that carry propagation across the approximation boundary represents a major source of application-level error in adaptive approximate IMC adders. Although the arithmetic improvements introduced by the carry lookback mechanism appear moderate when evaluated using conventional error metrics, their impact becomes amplified within deep neural networks. Errors affecting higher-significance bits can propagate through accumulations, activation functions, and multiple network layers, ultimately resulting in a disproportionate loss of inference accuracy. By mitigating these boundary-carry errors, RapproX improves the robustness of the evaluated application-level workloads without sacrificing the efficiency benefits of adaptive approximation. Among the two proposed variants, RapproX_C1_XOR generally exhibits the highest robustness at larger approximation levels, while RapproX_C1_OR already provides substantial improvements over ApprOchs. Together, these results indicate that lightweight carry awareness can translate a small architectural modification into an improvement in the evaluated machine-learning workloads, supporting the central design principle of RapproX.

Although energy measurements were not collected for these larger networks, their extensive use of ReLU activations and quantized arithmetic suggests behavior similar to that observed for AlexNet on MNIST. Consequently, the low-approximation operating points that provide near-exact accuracy are also expected to offer the most favorable accuracy–energy trade-off, making RapproX a promising candidate for efficient edge-AI inference. In terms of the practical range of k on this workload, ApprOchs frequently falls below 90% accuracy already at $k = 6$, so its practical range is limited to $k = 1$ to

$k = 5$, whereas both RapproX variants extend the practical range to $k = 1$ to $k = 6$. By inference from the AlexNet workload for which we did measure energy, the low end of that range is expected to offer the most favorable accuracy–energy trade-off for the evaluated architectures.

6.5. Application-Level Impact of the Lookback Mechanism

This subsection isolates the accuracy contribution of the C1 carry lookback at the application level by comparing each RapproX design against a no-lookback baseline. The natural baseline is a design that shares the same APAF magnitude-switching scaffold and, for the OR variant, the same OR-based approximate section, yet omits the lookback. ApprOchs meets exactly this description, so it acts as the no-lookback baseline here. Its remaining differences from RapproX, namely parallel versus sequential execution and single-row memristor reuse, affect hardware cost but not the arithmetic function computed, so in the OR variant the C1 lookback is the only architectural element that distinguishes RapproX from the baseline. The baseline-to-RapproX gap in Table 13 therefore quantifies the accuracy that the lookback mechanism adds at each approximation level. Only application-level metrics are shown here. The arithmetic-level comparison is in Table 3 and the circuit-level comparison in Section 5. The k-means clustering workload of Section 6.2 is omitted since no evaluated design retains usable accuracy on it.

The three panels tell a consistent story. Panel (a), image blurring, is where the lookback already contributes at mid k, adding more than 10 dB of mean PSNR for the OR variant across the full $k = 5$ to $k = 8$ range. Panel (b), AlexNet on MNIST, shows the no-lookback baseline having difficulties at high k. All three designs stay near the exact 98.9% reference accuracy through $k = 5$, then the baseline drops from 97.2% at $k = 6$ to 27.1% at $k = 8$, while both RapproX variants remain near the exact reference accuracy across the entire sweep, with the OR variant retaining an inference accuracy above 99% for every evaluated k. Panel (c), ResNet-50 on CIFAR-10, is where the baseline already loses roughly 10 percentage points by $k = 6$, and by $k = 7$ the baseline-to-RapproX gap has grown to more than 40 percentage points for the OR variant and more than 55 points for the XOR variant. Across all three panels, the baseline-to-RapproX gap is the accuracy that the lookback adds, with both RapproX variants outperforming the no-lookback baseline at high k while differing only marginally from each other architecturally. The two proposed design variants trade off in a workload-dependent way, with the OR variant leading on image blurring and AlexNet-MNIST and the XOR variant leading on ResNet-50. The same lookback-versus-no-lookback pattern is visible at the arithmetic level in Table 3 and, for CIFAR-10, in the boxplot summary of Figure 11.

The arithmetic-error metrics of Table 3 confirm this contribution below the application level. At $k = 5$ the no-lookback baseline reaches an MED of 7.623 and an MRED of 0.2695, whereas adding the C1 lookback lowers these to 5.783 and 0.2094 for RapproX_C1_OR, a reduction of about 24% in MED and 22% in MRED, and to 3.690 and 0.1431 for RapproX_C1_XOR, a reduction of about 52% in MED and 47% in MRED. NMED tracks MED through the constant factor $1/(2^{n+1} - 1)$ and therefore improves by the same proportion. These arithmetic reductions are the origin of the application-level gains reported in Table 13.

Table 13. Application-level comparison against the no-lookback baseline at the approximation levels where the designs begin to diverge. The baseline column is ApprOchs, which shares the RapproX scaffold and approximate section but omits the C1 lookback. Panel (**a**) reports image blurring mean PSNR, panel (**b**) reports AlexNet-MNIST accuracy, and panel (**c**) reports ResNet-50/CIFAR-10 accuracy as a representative of the ten networks in Table 12. Δ_{base} gives the gain over the baseline, and bold marks the row winner.

(a) Image Blurring (Section 6.1)					
Method	**Mean PSNR [dB] ↑**				
	Baseline	**RapproX_C1_OR**		**RapproX_C1_XOR**	
		Value	Δ_{base}	**Value**	Δ_{base}
Exact	∞	∞	–	∞	–
$k = 5$	30.9	**42.5**	+11.6	37.0	+6.1
$k = 6$	25.3	**36.6**	+11.3	31.4	+6.1
$k = 7$	20.4	**31.2**	+10.8	26.4	+6.0
$k = 8$	19.5	**26.8**	+7.3	22.8	+3.3
(b) AlexNet/MNIST (Section 6.3)					
Method	**Accuracy [%] ↑**				
	Baseline	**RapproX_C1_OR**		**RapproX_C1_XOR**	
		Value	Δ_{base}	**Value**	Δ_{base}
Exact	98.9	98.9	–	98.9	–
$k = 5$	98.7	**99.1**	+0.4	98.9	+0.2
$k = 6$	97.2	**99.1**	+1.9	98.7	+1.5
$k = 7$	82.6	**99.1**	+16.5	97.5	+14.9
$k = 8$	27.1	**99.0**	+71.9	85.6	+58.5
(c) ResNet-50/CIFAR-10 (Table 12)					
Method	**Accuracy [%] ↑**				
	Baseline	**RapproX_C1_OR**		**RapproX_C1_XOR**	
		Value	Δ_{base}	**Value**	Δ_{base}
Exact	93.63	93.63	–	93.63	–
$k = 5$	92.42	93.05	+0.63	**93.49**	+1.07
$k = 6$	83.11	90.71	+7.60	**93.11**	+10.00
$k = 7$	34.74	77.82	+43.08	**90.38**	+55.64
$k = 8$	10.05	34.85	+24.80	**74.85**	+64.80

7. Conclusions

In this work, we presented RapproX, a family of adaptive approximate memristive in-memory adders featuring a lightweight carry lookback mechanism. By incorporating carry awareness into adaptive approximation, RapproX improves arithmetic robustness while maintaining the energy-efficiency advantages of approximate in-memory computing. Furthermore, the sequential design enables efficient resource reuse, resulting in competitive area requirements despite the additional lookback functionality. Our simulation-based experiments indicate that the proposed approach outperforms the evaluated SoA adaptive approximate adders in arithmetic quality and application-level metrics. While the carry lookback mechanism introduces only a minor architectural modification, its impact becomes particularly apparent in machine-learning workloads. Across the evaluated set of neural-network architectures, including AlexNet, ResNet, DenseNet, Inception-V3, and VGG, RapproX preserves near-exact inference accuracy for the evaluated architectures at

low-to-moderate k, and outperforms the compared adaptive approximation schemes at higher k. These results suggest that carry propagation across the approximation boundary is a major source of application-level error in our experiments and that lightweight carry awareness can mitigate this limitation. Beyond machine learning, the proposed designs also show competitive performance in image-processing applications, achieving favorable accuracy–energy trade-offs while maintaining high output quality. Together, these results highlight the importance of evaluating approximate arithmetic not only through conventional error metrics, but also through its impact on representative application workloads. Future work will focus on improving throughput and latency, investigating alternative combinations of stateful logic families, and extending carry-aware approximation techniques to other arithmetic building blocks for energy-efficient edge-AI systems.

Author Contributions: Conceptualization, L.R. and L.B.; Methodology, L.R., L.B. and F.S.; Circuit Design, L.B. and F.S.; Circuit-level Simulation, L.B. and F.S.; Application-level Experiments, L.R. and N.A.; Formal Analysis, L.R., L.B., F.S. and N.A.; Visualization, L.R., L.B. and F.S.; Original Draft Preparation, L.R. and L.B.; Review and Editing, F.S., N.A. and N.T.; Discussion, L.R., L.B., F.S., N.A. and N.T.; Supervision, N.T. All authors have read and agreed to the published version of the manuscript.

Funding: We appreciate the support provided for this work by the Deutsche Forschungsgemeinschaft (DFG, German Research Foundation) under Project 550835853 and the Hector Stiftung.

Institutional Review Board Statement: Not applicable.

Informed Consent Statement: Not applicable.

Data Availability Statement: The evaluation data presented in this study are available on reasonable request from the corresponding author.

Conflicts of Interest: The authors declare no conflicts of interest.

Abbreviations

The following abbreviations are used in this manuscript:

APAF	Adaptive Precision Adder Framework
FELIX	Fast and Energy-efficient Logic
IMC	In-Memory Computation
IMPLY	Material Implication
MAGIC	Memristor-Aided Logic
MED	Mean Error Distance
MRED	Mean Relative Error Distance
MSB	Most Significant Bit
NMED	Normalized Mean Error Distance
PSNR	Peak Signal-to-Noise Ratio
ReLU	Rectified Linear Unit
SIXOR	Single-cycle XOR
SoA	State-of-the-Art
SSIM	Structural Similarity Index Measure
TMSL	Three Memristor Stateful Logic

References

1. Seiler, F.; TaheriNejad, N. Approximate Computing in Memristive Processing-In-Array (PIA). *IEEE Des. Test* **2026**. https://doi.org/10.1109/MDAT.2026.3691674.
2. Jiang, H.; Santiago, F.J.H.; Mo, H.; Liu, L.; Han, J. Approximate Arithmetic Circuits: A Survey, Characterization, and Recent Applications. *Proc. IEEE* **2020**, *108*, 2108–2135. https://doi.org/10.1109/JPROC.2020.3006451.
3. Mittal, S. A survey of techniques for approximate computing. *ACM Comput. Surv. (CSUR)* **2016**, *48*, 62.

4. Bharti, S.; Kumar, A.; Gupta, P. Review of Approximate Computing in Image Processing Applications. In *Proceedings of the 2022 4th International Conference on Artificial Intelligence and Speech Technology (AIST)*; IEEE: New York, NY, USA, 2022; pp. 1–6.
5. Gupta, V.; Mohapatra, D.; Raghunathan, A.; Roy, K. Low-Power Digital Signal Processing Using Approximate Adders. *IEEE Trans. Comput.-Aided Des. Integr. Circuits Syst.* **2013**, *32*, 124–137. https://doi.org/10.1109/TCAD.2012.2217962.
6. Sabetzadeh, F.; Moaiyeri, M.H.; Ahmadinejad, M. A majority-based imprecise multiplier for ultra-efficient approximate image multiplication. *IEEE Trans. Circuits Syst. I Regul. Pap.* **2019**, *66*, 4200–4208.
7. Mutlu, O.; Ghose, S.; Gómez-Luna, J.; Ausavarungnirun, R.; Sadrosadati, M.; Oliveira, G.F. A Modern Primer on Processing in Memory. *arXiv* **2025**, arXiv:2012.03112. https://doi.org/10.48550/arXiv.2012.03112.
8. Lehtonen, E.; Laiho, M. Stateful implication logic with memristors. In *Proceedings of the 2009 IEEE/ACM International Symposium on Nanoscale Architectures*; IEEE: New York, NY, USA, 2009; pp. 33–36.
9. Gupta, S.; Imani, M.; Rosing, T. Felix: Fast and energy-efficient logic in memory. In *Proceedings of the 2018 IEEE/ACM International Conference on Computer-Aided Design (ICCAD)*; IEEE: New York, NY, USA, 2018; pp. 1–7.
10. Kvatinsky, S.; Belousov, D.; Liman, S.; Satat, G.; Wald, N.; Friedman, E.G.; Kolodny, A.; Weiser, U.C. MAGIC—Memristor-Aided Logic. *IEEE Trans. Circuits Syst. II Express Briefs* **2014**, *61*, 895–899. https://doi.org/10.1109/TCSII.2014.2357292.
11. TaheriNejad, N. SIXOR: Single-Cycle In-Memristor XOR. *IEEE Trans. Very Large Scale Integr. (VLSI) Syst.* **2021**, *29*, 925–935. https://doi.org/10.1109/TVLSI.2021.3062293.
12. Fatemieh, S.E.; Reshadinezhad, M.R.; TaheriNejad, N. Fast and Compact Serial IMPLY-Based Approximate Full Adders Applied in Image Processing. *IEEE J. Emerg. Sel. Top. Circuits Syst.* **2023**, *13*, 175–188. https://doi.org/10.1109/JETCAS.2023.3241012.
13. Seiler, F.; TaheriNejad, N. Efficient Image Processing via Memristive-Based Approximate In-Memory Computing. *IEEE Trans. Comput.-Aided Des. Integr. Circuits Syst.* **2024**, *43*, 3312–3323. https://doi.org/10.1109/TCAD.2024.3438113.
14. Asgari, S.; Reshadinezhad, M.R.; Fatemieh, S.E. Energy-efficient and fast IMPLY-based approximate full adder applying NAND gates for image processing. *Comput. Electr. Eng.* **2024**, *113*, 109053.
15. Seiler, F.; TaheriNejad, N. Accelerated Image Processing Through IMPLY-Based NoCarry Approximated Adders. *IEEE Trans. Circuits Syst. I Regul. Pap.* **2024**, *71*, 5141–5154.
16. Kaushik, N.; Krishna, H.; Bodapati, S. High-speed serial and semi-parallel imply-based approximate adders through memristors for in-memory computing. In *Proceedings of the 2024 IEEE International Symposium on Circuits and Systems (ISCAS)*; IEEE: New York, NY, USA, 2024; pp. 1–5.
17. Kaushik, N.; Seiler, F.; Amirafshar, N.; TaheriNejad, N.; Srinivasu, B. High-Speed MAGIC-Based Exact and Approximate Adders for In-Memory Computing. *IEEE J. Emerg. Sel. Top. Circuits Syst.* **2026**, *16*, 312–326. https://doi.org/10.1109/JETCAS.2026.3669637.
18. Ochs, D.; Rapp, L.; Borzyk, L.; Amirafshar, N.; TaheriNejad, N. ApprOchs: A Memristor-Based In-Memory Adaptive Approximate Adder. *IEEE J. Emerg. Sel. Top. Circuits Syst.* **2025**, *15*, 105–119. https://doi.org/10.1109/JETCAS.2025.3537328.
19. Chua, L. Memristor-The missing circuit element. *IEEE Trans. Circuit Theory* **1971**, *18*, 507–519. https://doi.org/10.1109/TCT.1971.1083337.
20. Strukov, D.B.; Snider, G.S.; Stewart, D.R.; Stanley Williams, R. The missing memristor found. *Nature* **2008**, *453*, 80–83. https://doi.org/10.1038/nature06932.
21. Truong, M.S.Q.; Chen, E.; Su, D.; Shen, L.; Glass, A.; Carley, L.R.; Bain, J.A.; Ghose, S. RACER: Bit-Pipelined Processing Using Resistive Memory. In *Proceedings of the MICRO-54: 54th Annual IEEE/ACM International Symposium on Microarchitecture*; MICRO'21; IEEE: New York, NY, USA, 2021; pp. 100–116. https://doi.org/10.1145/3466752.3480071.
22. Borghetti, J.; Snider, G.S.; Kuekes, P.J.; Yang, J.J.; Stewart, D.R.; Stanley Williams, R. 'Memristive' switches enable 'stateful' logic operations via material implication. *Nature* **2010**, *464*, 873–876. https://doi.org/10.1038/nature08940.
23. Karimi, A.; Rezai, A. Novel design for a memristor-based full adder using a new IMPLY logic approach. *J. Comput. Electron.* **2018**, *17*, 1303–1314.
24. Huang, Y.; Ando, T.; Sebastian, A.; Chang, M.F.; Yang, J.J.; Xia, Q. Memristor-based hardware accelerators for artificial intelligence. *Nat. Rev. Electr. Eng.* **2024**, *1*, 286–299. https://doi.org/10.1038/s44287-024-00037-6.
25. Wang, S.; Luo, Y.; Zuo, P.; Li, Y.; Ielmini, D.; Sun, Z. In-memory analog computing for non-negative matrix factorization. *Nat. Commun.* **2026**, *17*, 1881. https://doi.org/10.1038/s41467-026-68609-8.
26. Greco, L.; Lico, A.; Gahem, F.; Franchi Scarselli, E.; Pasotti, M.; Antolini, A. Current Subtraction for Signed MVM Analog in-Memory Computation on Resistive Arrays. In *Proceedings of the Proceedings of SIE 2025*; Pavan, P., Campopiano, S., Eds.; Springer: Cham, Switzerland, 2026; pp. 11–19.
27. Chen, P.; Zhang, B.; He, E.; Xiao, Y.; Liu, F.; Lin, P.; Wang, Z.; Pan, G. Towards scalable memristive hardware for spiking neural networks. *Mater. Horiz.* **2025**, *12*, 2820–2839. https://doi.org/10.1039/D4MH01676A.
28. Shooshtari, M.; Serrano-Gotarredona, T.; Linares-Barranco, B. Review of Memristors for In-Memory Computing and Spiking Neural Networks. *Adv. Intell. Syst.* **2026**, *8*, e202500806. https://doi.org/https://doi.org/10.1002/aisy.202500806.

29. Gao, Z.; Wang, L.; Duan, S. A Memristor-Based Neural Network Circuit with Classical Conditioning and Fear Generalization. *IEEE Trans. Consum. Electron.* **2026**, *72*, 3213–3224. https://doi.org/10.1109/TCE.2026.3663359.
30. Kvatinsky, S.; Kolodny, A.; Weiser, U.C.; Friedman, E.G. Memristor-based IMPLY logic design procedure. In *Proceedings of the 2011 IEEE 29th International Conference on Computer Design (ICCD)*; IEEE: New York, NY, USA, 2011; pp. 142–147. https://doi.org/10.1109/ICCD.2011.6081389.
31. Huang, P.; Kang, J.; Zhao, Y.; Chen, S.; Han, R.; Zhou, Z.; Chen, Z.; Ma, W.; Li, M.; Liu, L.; et al. Reconfigurable Nonvolatile Logic Operations in Resistance Switching Crossbar Array for Large-Scale Circuits. *Adv. Mater.* **2016**, *28*, 9758–9764. https://doi.org/10.1002/adma.201602418.
32. Hoffer, B.; Rana, V.; Menzel, S.; Waser, R.; Kvatinsky, S. Experimental Demonstration of Memristor-Aided Logic (MAGIC) Using Valence Change Memory (VCM). *IEEE Trans. Electron Devices* **2020**, *67*, 3115–3122. https://doi.org/10.1109/TED.2020.3001247.
33. Truong, M.S.Q.; Shen, L.; Glass, A.; Hoffmann, A.; Carley, L.R.; Bain, J.A.; Ghose, S. Adapting the RACER Architecture to Integrate Improved In-ReRAM Logic Primitives. *IEEE J. Emerg. Sel. Top. Circuits Syst.* **2022**, *12*, 393–407. https://doi.org/10.1109/JETCAS.2022.3171765.
34. Liu, X.; Pan, Z.; Xia, Z.; Li, L.; Deng, Q.; Pan, Y.; Li, J.; Huo, N. Memristors Based on 2D Materials: Bridging Device Theory and Potential Applications. *Adv. Funct. Mater.* **2026**, *36*, e20816. https://doi.org/https://doi.org/10.1002/adfm.202520816.
35. Shooshtari, M.; Kim, S.Y.; Pahlavan, S.; Rivera-Sierra, G.; Través, M.J.; Serrano-Gotarredona, T.; Bisquert, J.; Linares-Barranco, B. Advancing Logic Circuits With Halide Perovskite Memristors for Next-Generation Digital Systems. *SmartMat* **2025**, *6*, e70032. https://doi.org/https://doi.org/10.1002/smm2.70032.
36. Kim, Y.; Jeon, S.B.; Jang, B.C. Graphene Oxide-Based Memristive Logic-in-Memory Circuit Enabling Normally-Off Computing. *Nanomaterials* **2023**, *13*, 710. https://doi.org/10.3390/nano13040710.
37. Seiler, F.; Gulafshan; Taherinejad, N. An Efficient Robust Serial IMPLY-based In-Memristor Adder. In *Proceedings of the IEEE International Conference on Cross-Disciplinary Conference on Memory-Centric Computing (CCMCC)*; IEEE: New York, NY, USA, 2025; pp. 1–7.
38. Rohani, S.G.; TaheriNejad, N. An improved algorithm for IMPLY logic based memristive Full-adder. In *Proceedings of the 2017 IEEE 30th Canadian Conference on Electrical and Computer Engineering (CCECE)*; IEEE: New York, NY, USA, 2017; pp. 1–4. https://doi.org/10.1109/CCECE.2017.7946813.
39. Mahdiani, H.R.; Ahmadi, A.; Fakhraie, S.M.; Lucas, C. Bio-inspired imprecise computational blocks for efficient VLSI implementation of soft-computing applications. *IEEE Trans. Circuits Syst. I Regul. Pap.* **2009**, *57*, 850–862.
40. Napoli, E.; Zacharelos, E.; Strollo, A.G.M.; Di Meo, G. Approximate Full-Adders: A Comprehensive Analysis. *IEEE Access* **2024**, *12*, 136054–136072. https://doi.org/10.1109/ACCESS.2024.3463182.
41. Mrazek, V.; Sarwar, S.S.; Sekanina, L.; Vasicek, Z.; Roy, K. Design of power-efficient approximate multipliers for approximate artificial neural networks. In *Proceedings of the 35th International Conference on Computer-Aided Design*; ICCAD'16; IEEE: New York, NY, USA, 2016. https://doi.org/10.1145/2966986.2967021.
42. Farahmand, E.; Mahani, A.; Hanif, M.A.; Shafique, M. Design and Analysis of High Performance Heterogeneous Block-based Approximate Adders. *ACM Trans. Embed. Comput. Syst.* **2023**, *22*, 106. https://doi.org/10.1145/3625686.
43. Leon, V.; Hanif, M.A.; Armeniakos, G.; Jiao, X.; Shafique, M.; Pekmestzi, K.; Soudris, D. Approximate Computing Survey, Part I: Terminology and Software & Hardware Approximation Techniques. *ACM Comput. Surv.* **2025**, *57*, 185. https://doi.org/10.1145/3716845.
44. Ebrahimi-Azandaryani, F.; Akbari, O.; Kamal, M.; Afzali-Kusha, A.; Pedram, M. Block-Based Carry Speculative Approximate Adder for Energy-Efficient Applications. *IEEE Trans. Circuits Syst. II Express Briefs* **2020**, *67*, 137–141. https://doi.org/10.1109/TCSII.2019.2901060.
45. Dalloo, A.; Najafi, A.; Garcia-Ortiz, A. Systematic Design of an Approximate Adder: The Optimized Lower Part Constant-OR Adder. *IEEE Trans. Very Large Scale Integr. (VLSI) Syst.* **2018**, *26*, 1595–1599. https://doi.org/10.1109/TVLSI.2018.2822278.
46. Lee, J.; Seo, H.; Seok, H.; Kim, Y. A Novel Approximate Adder Design Using Error Reduced Carry Prediction and Constant Truncation. *IEEE Access* **2021**, *9*, 119939–119953. https://doi.org/10.1109/ACCESS.2021.3108443.
47. Joshi, V.; Mane, P. Novel approximate adaptive carry lookahead adder for error resilient applications with generic method for error analysis. *Sci. Rep.* **2025**, *15*, 19215. https://doi.org/10.1038/s41598-025-03865-0.
48. Muthulakshmi, S.; Dash, C.S.; Prabaharan, S. Memristor augmented approximate adders and subtractors for image processing applications: An approach. *AEU-Int. J. Electron. Commun.* **2018**, *91*, 91–102.
49. Muthulakshmi, S.; Dash, C.S.; Prabaharan, S. Memristor-based approximate adders for error resilient applications. In *Proceedings of the Nanoelectronic Materials and Devices: Select Proceedings of ICNETS2*; Springer: Berlin/Heidelberg, Germany, 2018; Volume III, pp. 51–59.
50. Fatemieh, S.E.; Asgari, S.; Reshadinezhad, M.R. Fast and low-energy approximate full adder based on FELIX logic. *Comput. Electr. Eng.* **2026**, *136*, 111220. https://doi.org/https://doi.org/10.1016/j.compeleceng.2026.111220.

51. Fatemieh, S.E.; Farahani, S.S.; Reshadinezhad, M.R. LAHAF: Low-power, area-efficient, and high-performance approximate full adder based on static CMOS. *Sustain. Comput. Inform. Syst.* **2021**, *30*, 100529.
52. Jiang, H.; Liu, C.; Liu, L.; Lombardi, F.; Han, J. A review, classification, and comparative evaluation of approximate arithmetic circuits. *ACM J. Emerg. Technol. Comput. Syst. (JETC)* **2017**, *13*, 60.
53. Kvatinsky, S.; Ramadan, M.; Friedman, E.G.; Kolodny, A. VTEAM: A General Model for Voltage-Controlled Memristors. *IEEE Trans. Circuits Syst. II Express Briefs* **2015**, *62*, 786–790. https://doi.org/10.1109/TCSII.2015.2433536.
54. Teimoory, M.; Amirsoleimani, A.; Shamsi, J.; Ahmadi, A.; Alirezaee, S.; Ahmadi, M. Optimized implementation of memristor-based full adder by material implication logic. In *Proceedings of the ICECS2014*; IEEE: New York, NY, USA, 2014; pp. 562–565.
55. Radakovits, D.; TaheriNejad, N.; Cai, M.; Delaroche, T.; Mirabbasi, S. A Memristive Multiplier Using Semi-Serial IMPLY-Based Adder. *IEEE Trans. Circuits Syst. I Regul. Pap.* **2020**, *67*, 1495–1506. https://doi.org/10.1109/TCSI.2020.2965935.
56. Ganjeheizadeh Rohani, S.; Taherinejad, N.; Radakovits, D. A Semiparallel Full-Adder in IMPLY Logic. *IEEE Trans. Very Large Scale Integr. (VLSI) Syst.* **2020**, *28*, 297–301. https://doi.org/10.1109/TVLSI.2019.2936873.
57. Seiler, F.; Hinkel, P.M.; Jantsch, A.; TaheriNejad, N. ATOMIC: Automatic Tool for Memristive IMPLY based Circuit-level Simulation and Validation. In *Proceedings of the 2025 21st International Conference on Synthesis, Modeling, Analysis and Simulation Methods and Applications to Circuit Design (SMACD)*; IEEE: New York, NY, USA, 2025; pp. 1–4.
58. Zacharelos, E.; Nunziata, I.; Saggese, G.; Strollo, A.G.; Napoli, E. Approximate Recursive Multipliers Using Low Power Building Blocks. *IEEE Trans. Emerg. Top. Comput.* **2022**, *10*, 1315–1330. https://doi.org/10.1109/TETC.2022.3186240.
59. OpenImagesV7 Image Dataset. Available online: https://storage.googleapis.com/openimages/web/download_v7.html#download-manually (accessed on 12 September 2024).
60. Krizhevsky, A.; Sutskever, I.; Hinton, G.E. Imagenet classification with deep convolutional neural networks. *Adv. Neural Inf. Process. Syst.* **2012**, *25*, 1097–1105.