# Approximation Opportunities in Edge Computing Hardware: A Systematic Literature Review

HANS JAKOB DAMSGAARD, ALEKSANDR OMETOV, and JARI NURMI, Tampere University, Finland

With the increasing popularity of the Internet of Things and massive Machine Type Communication technologies, the number of connected devices is rising. However, while enabling valuable effects to our lives, bandwidth and latency constraints challenge Cloud processing of their associated data amounts. A promising solution to these challenges is the combination of Edge and approximate computing techniques that allows for data processing nearer to the user. This paper aims to survey the potential benefits of these paradigms' intersection. We provide a state-of-the-art review of circuit-level and architecture-level hardware techniques and popular applications. We also outline essential future research directions.

CCS Concepts: • **General and reference** → **Surveys and overviews**; • **Computing methodologies**; • **Computer systems organization** → *Reconfigurable computing*; *Architectures*; *Distributed architectures*; • **Hardware**;

Additional Key Words and Phrases: approximate computing, edge computing

## 1 INTRODUCTION

Historically, the invention of the smartphone has enabled near-immediate connection of people across the globe and vastly increased the amount of data being produced [13, 26]. The growing interest in the Internet of Things (IoT) and massive Machine Type Communication (mMTC) technologies indicates that this trend is not slowing down. It is key to note that aggregating these data implies considerable computing demands that have not been met outside the Cloud, until recently when rapid developments in computing hardware have distributed high-performance devices at the Edge of the Internet [197]. This distribution enables new types of data processing away from the Cloud, nearer to the users.

The fact that more computations imply greater energy consumption is troublesome in small battery-driven devices such as smartphones and wearables [121]. Many applications in this domain also fail to optimally exploit their error resilience. Instead, they utilize high-precision computations that cause additional energy consumption. Here is an evident improvement opportunity, as users expect low-latency operation and *good enough* quality content in a trade-off for *long enough* battery life [129]. There is, thus, an important balance of time, quality, and energy to be identified.

Computing, as we know it, is based on previous research efforts that have focused on the time-energy trade-off and led to advances in high-capacity battery technologies, efficient (application-specific) processor design, algorithms, and low-overhead communication. Time has, nevertheless, shown that these techniques alone do not suffice. Moreover, we are reaching the end of helpful historic trends like Moore's law and Dennard scaling [46]. Thus, radical changes to the computing paradigm are needed to address future increases in the number of

The authors gratefully acknowledge funding from European Union's Horizon 2020 Research and Innovation programme under the Marie Skłodowska Curie grant agreement No. 956090 (APROPOS: Approximate Computing for Power and Energy Optimisation, http://www.apropos-itn.eu/.)

Authors' address: Hans Jakob Damsgaard, hans.damsgaard@tuni.fi; Aleksandr Ometov; Jari Nurmi, Tampere University, Electrical Engineering Unit, Korkeakoulunkatu 1, Tampere, Finland, 33720.


0360-0300/2022/11-ART
https://doi.org/10.1145/3572772

connected devices and computing demands [169]. We expect two techniques to be crucial in this transition: offloading and Approximate Computing (AxC). While the former of these, offloading, is well-known from Cloud computing in which computations are *offloaded* from constrained, often battery-driven devices to powerful, always-connected servers; the latter, AxC, in which computations are performed *approximately* to save power, energy, or latency, is yet to be fully explored. Combining the two is expected to enable the time-quality-energy trade-off mentioned above. Let us first understand these techniques in more detail.

The offloading technique was historically motivated by the low performance but good connectivity of battery-driven devices. These features effectively forced these devices to focus their energy consumption on capturing sensor data. Thus, rather than processing data locally, the devices would *offload* these tasks to another more powerful computer [8]. Today, offloading is essential in three computing paradigms: Cloud, Fog, and Edge computing, which differ in where data is stored and processed. Cloud computing manages all data in large, centralized data centers. Fog computing is a term coined by CISCO describing an intermediate step between Cloud and local computing utilizing so-called *Fog nodes* [22]. Finally, in Edge computing, data is processed on the nearest Edge node [155]. The Fog and Edge computing terms are often used synonymously, and we will consider them as one. We provide an overview of the computing paradigms and their relations in Fig. 1. It displays examples of devices and their current and projected connections, as well as the compute power and expected approximation benefits in each paradigm. We motivate this expectation by the fact that as more data is efficiently processed (approximately) near the End or Edge layers, less data will incur communication overheads arising from routing through many network layers, reducing overall system energy consumption [51, 134].

The Cloud can be considered as a near-infinite source of computational power relative to data-generating end devices, but communicating with it comes at a high cost of latency and power consumption. The longer networking distances also imply that more links are shared with other devices causing network backbone contention. Hence, this paradigm does not scale with the increasing number of connected devices and data amounts [22]. Yet, as offloading remains beneficial for nearly all compute-heavy tasks [6, 51], research has recently focused on the Edge computing paradigm as a potential solution. While there exist many definitions of which devices constitute the Edge, the common factor is that they possess better computational capabilities than end devices such as wearables, sensor nodes, etc. As such, smartphones, laptops, desktop PCs, and distributed servers (e.g., at base stations) can be considered Edge devices [18], as shown in Fig. 1. The transition to Edge computing and its effects on moving computations and content closer to end users is interesting for multiple reasons: Firstly,

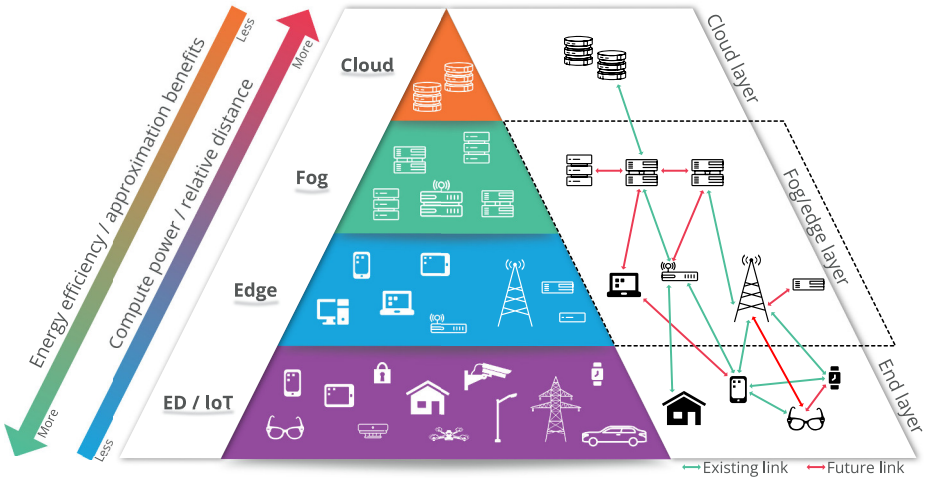


Fig. 1. Overview of the different computing domains and some of their characteristics.

the data-generating devices retain the benefits of offloading, including shorter-range communication. Secondly, communication passes fewer links as less traffic will reach the Cloud, reducing network backbone contention. Finally, data security becomes more easily manageable as data resides in fewer shared links and devices [155].

In parallel to this transition, applications have become more user-centric and solve more complex, less well-defined problems. They challenge the historical notion of application output *correctness* by instead producing *acceptable* outputs. Examples of such are multimedia applications whose outputs are consumed by humans with inherently low sensitivity to noise [44], and Recognition, Mining, and Synthesis (RMS) applications that aggregate large amounts of data and produce some useful output [33]. Both are characterized by being insensitive or *resilient* to computational errors [169]. This characteristic has motivated and remains at the core of the AxC paradigm, enabling energy or latency savings.

In AxC, approximations are applied at different levels of a system to achieve reductions in computational complexity, memory demands, or communication bandwidth [169]. Traditional techniques fall into two classes: computing on approximate data and computing on unreliable hardware [14]. Computing on approximate data essentially means applying approximations at the application or architecture level; the textbook example is quantization of floating-point operations to simpler fixed-point operations. Although quantization is a powerful technique, it should be supplemented by orthogonal techniques such as loop perforation, message skipping, neural approximation, etc. to take full advantage of an application's error resilience [59, 105]. Contrarily, computing on unreliable hardware means introducing errors at the circuit level. This can be done using faulty circuits that would traditionally be discarded due to their inherently non-deterministic behavior but may bed used in approximate systems instead, assuming that the detected faults introduce acceptable errors. Alternatively, designers can manually introduce faulty behavior by over-scaling operating voltage or frequency or by using reduced-voltage Static RAM (SRAM) and reduced refresh-rate Dynamic RAM (DRAM) [10, 105].

Recent research efforts have focused on developing cross-layer techniques to maximize the benefits from approximation while satisfying given quality constraints [120]. Granted that this is still a relatively young research field, existing results are promising [145, 171]. Unfortunately, despite their common goals of achieving energy savings and latency reductions, research into offloading and AxC have mostly been separate fields, underlined by extensive works on either topic but a lack of works on their intersection. The techniques are orthogonal and expected to complement one another well. Specifically, their combination allows for the execution of applications with varying accuracy at different levels of the network. The potential benefits of this include improved computing efficiency and infrastructure flexibility.

In this paper, we focus on the combination of offloading and AxC and provide a systematic literature review on circuit-level and architecture-level approximation techniques and applications of them to areas relevant to Edge computing. We note that several surveys on AxC already exist, but none have a particular focus on Edge computing. The interested reader can refer to [10, 32, 33, 105, 169, 187] for introductions to the topic. Our **main goals** and **contributions** are:

**G1** to give an overview of recent works that implement and evaluate circuit-level and architecture-level AxC techniques,
**G2** to survey works that utilize these techniques in Edge-related applications, and
**G3** to identify open challenges and research directions in the intersection of the AxC and Edge computing paradigms.

The remainder of the paper is structured as follows (also shown in Fig. 2). The next three sections present the reviewed works according to our classification, detailed in Appendix A. First, Sec. 2 covers works on fundamental AxC techniques; second, Sec. 3 summarizes works on AxC-enabled hardware architectures; and third, Sec. 4 presents applications which can benefit from offloading and AxC. In Sec. 5, we contribute by highlighting

challenges and future research directions that have become apparent through the review. Sec. 6 concludes the review. Appendix B presents tabulated, short summaries of all works from Secs. 2, 3, and 4.

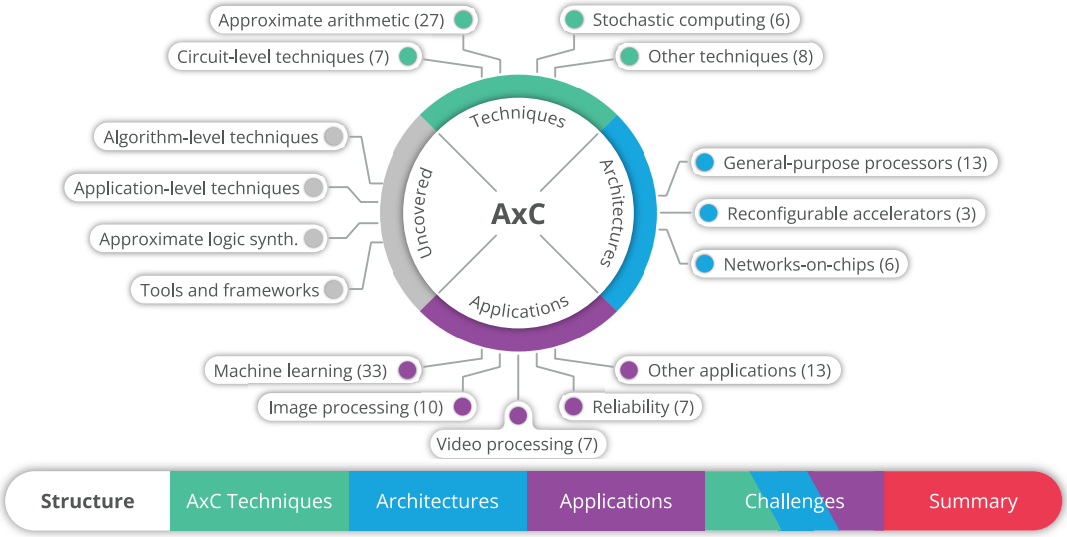


Fig. 2. Overviews of the topics covered in this paper, the number of papers in each category (parenthesized), and the structure of the remaining sections.

## 2 FUNDAMENTAL APPROXIMATE COMPUTING TECHNIQUES

This section reviews fundamental AxC techniques. We first consider circuit-level techniques before moving to designs of arithmetic circuits, stochastic computing, function approximation, and other general techniques, as shown in Fig. 3. The techniques presented here serve as a basis for the architectures and application-specific designs reviewed in the following sections. We provide an overview of commonly used benchmark applications in Tab. 2 and short summaries of all works in the appendix, see Tab. 6. Note that we do not report any numerical results in text due to the difficulty of comparing different evaluation strategies, yet we display select metrics in plots. Regardless, we attempt to provide a qualitative comparison based on the numerical data found in the literature.

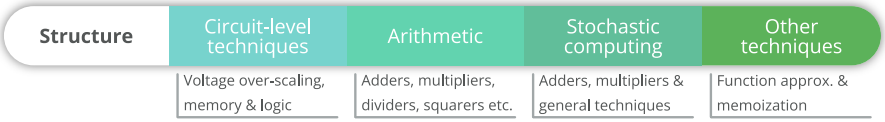


Fig. 3. Coarse classification of publications on fundamental AxC techniques.

### 2.1 Circuit-level Techniques

Traditional digital hardware design focuses on circuits meant to operate in an error-free manner, which is ensured by following process-specific design parameters determining, among others, operating voltage and frequency ranges and their respective safety ranges. However, the error-free operating points are not necessarily the most energy efficient. Recall that power consumption in digital circuits consists of two parts: static power consumption $P_{leakage} \propto I_0 V_{DD}$ from leakage current $I_0$ through open transistors and dynamic power consumption $P_{dyn} = \alpha f C V_{DD}^2$ from charging and discharging of a circuit's intrinsic capacitances [10]. For most designs operating at nominal conditions, dynamic power consumption is dominant [43, 53], and, thus, the greatest gains

can be achieved by reducing operating voltage $V_{DD}$ or frequency $f$, the switched capacitance $C$, or the circuit activity $\alpha$. Deviating from these conditions in a controlled way enables operating with a low error rate, but a high impact on power consumption; for example, reducing voltage leads to linear and quadratic reductions in static and dynamic power consumption. We review six works on this topic.

*2.1.1 Voltage over-scaling.* Two works consider so-called Voltage Over-Scaling (VOS), a technique in which the operating voltage is reduced below its safety margin. Moons and Verhelst [109] propose combining truncation with non-destructive voltage scaling. By truncating operands to a circuit, only parts of its critical paths will be in use, effectively decreasing its latency, an effect lowering the operating voltage can counter. However, the technique does not trivially extend to pipelined designs as their critical paths are not necessarily shortened by truncation. The authors provide two solutions: implementing additional bypassable pipeline registers or multiplexing different circuit paths to the existing pipeline registers. Either solution introduces an overhead proportional to the number of voltage-accuracy configurations wanted. Evaluated on a Discrete Cosine Transform (DCT) algorithm for Joint Photographic Experts Group (JPEG) compression, the technique can halve energy consumption with negligible impact on image quality.

Tziantzioulis *et al.* [166] focus on approximate voltage over-scaled Function Units (FUs) in traditional processor architectures. They describe how FUs are often under-utilized and propose to exploit this feature to let operations issued to the approximate units execute beyond their nominal delay, collecting their results only once equivalent operations are issued to them. This way, the results have more time to converge towards their exact values, reducing experienced error magnitudes in low-activity code regions. They control the approximation with a custom instruction and implement the hardware necessary for lazy write-back and forwarding. The authors demonstrate reduced error rates in three multimedia applications.

A related work by Tagliavini *et al.* [163] proposes a cache architecture implemented partly in SRAM and partly in Standard Cell Memories (SCMs) designed to operate at Near-Threshold Voltage (NTV) – an extreme case of VOS. Operating circuits at NTV reduces both dynamic and static power consumption but makes them more vulnerable to erroneous effects from process and temperature variations [10]. The authors illustrate this with their proposed design, exploiting that while SCMs are more costly in terms of area, they are more reliable at lower voltages than SRAM, enabling compiler-guided placement of MSBs in reliable SCM and LSBs in unreliable SRAM. They demonstrate reduced energy consumption in the cache and the System on Chip (SoC) that comprises it across several applications.

*2.1.2 Approximate memories.* Another two works also explore the use of unreliable memories. Shoushtari *et al.* [156] describe that typical SRAMs utilize costly redundancy techniques and are kept at high voltage to guarantee error-free data retention, leading to over-design and high static power consumption. Eliminating some of the redundancy and selectively lowering operating voltage can lead to reduced power consumption at the cost of some randomly induced errors. They illustrate their proposal with a cache architecture that implements some unreliable ways, a programmer-controlled criticality-aware replacement policy, and hardware to disable ways that introduce too many errors. Evaluated on two media applications, they show greatly reduced leakage energy at little quality and throughput degradation.

Orthogonally, Ganapathy *et al.* [53] consider embedded DRAMs that are gaining popularity as a higher-density alternative to SRAMs, but whose predictability and data retention time scale poorly with smaller process technologies. To counter these effects, the DRAMs require frequent, costly refresh operations to function reliably. The authors propose relaxing refresh requirements while accepting a low probability of errors, reducing dynamic power consumption and improving availability (as the memory is less busy refreshing itself). Using this technique, they show how different error probabilities affect the output quality of two Machine Learning (ML) applications.

*2.1.3 Alternative logic.* Lastly, Yang *et al.* [192] propose implementing approximate designs in *adiabatic* logic – a clock-powered alternative to Complementary Metal-Oxide Semiconductor (CMOS) that lengthens signal transitions to reduce dynamic power consumption. The authors present two approximate adders and evaluate them at different operating frequencies, reporting an order of magnitude lower power consumption compared to CMOS equivalents.

## 2.2 Arithmetic

Having reviewed circuit-level techniques, it is relevant also to get an overview of AxC applied at a higher level of abstraction. As such, we review and classify 27 works on approximate arithmetic into three main groups based on the operations they focus on. Whenever relevant, we provide pointers to the aforementioned circuit-level techniques.

*2.2.1 Adders.* We first review nine works that present approximate adders, distinguishing designs based solely on inexact full adders from segmented, partially-exact designs with or without error compensation.

The works presenting inexact full adder designs all focus on Ripple-Carry Adders (RCAs) that are characterized by being slow but area-efficient, both traits being results of their single-bit carry propagation [7]. Recall that a full adder takes three inputs: two operand bits $A$ and $B$ and a carry-in-bit $C_{in}$, and calculates a sum-bit $S = A \oplus B \oplus C_{in}$ and a carry-out-bit $C_{out} = A \cdot B + A \cdot C_{in} + B \cdot C_{in}$. Approximate full adders can, thus, be classified by their outputs being **A1**) exact sum $S$ and inexact carry $\hat{C}_{out}$, **A2**) inexact sum $\hat{S}$ and exact carry $C_{out}$, or **A3**) inexact sum $\hat{S}$ and inexact carry $\hat{C}_{out}$. Yang *et al.* [194] first propose three different full adder designs: two of type **A2** and one of type **A3**, all implemented with *pass transistor* logic – another alternative to CMOS. Later [193], they propose another two designs of type **A3** implemented with transmission gates. Allen *et al.* [7] present four designs: two of type **A1** and two of type **A2**. Dutt *et al.* [42] present four designs of type **A1**, two of which coincide with [7]. Lastly, in preliminary work to their more recent one [192], Yang and Thapliyal [191] present two designs, one **A1** and one **A2** implemented in adiabatic logic. Both [7] and [42] motivate their design decisions by combining knowledge about logic gate design parameters and signal probabilities derived from extensive analyses. The five works evaluate their designs differently, but all achieve improvements in one or more key metrics: power [7, 42, 191, 193, 194], area [42, 194], or error [42, 193]. Unfortunately, the results do not clarify whether approximating the the sum-bit $S$ or the carry-out-bit $C_{out}$ is most beneficial.

Focusing on segmented adder architectures, we find that works fall into two categories depending on the type of adder they implement. Dutt *et al.* [42] retain their focus on RCAs, proposing to split the adder into two parts, approximate the least-significant part, but recover quality using error compensation logic. Dalloo [35] proposes a feedback-based error-correction scheme with the same idea in mind. Kim *et al.* [83] propose a carry-lookahead adder whose intermediate carries are approximated by considering only some less significant bits. Yang *et al.* [189] also present a carry-lookahead adder that supports selective masking of certain carries. All four works show improved error metrics, but often at the cost of increased area and delay [42, 83, 189]. Nomani *et al.* [116] propose several approximate adder designs targeting Field-Programmable Gate Array (FPGA) implementation. All designs map several bits of a sum to a pair of 6-input LUTs, common to most modern FPGAs, thereby removing nearly all carry propagation and achieving single-Lookup Table (LUT) logic depth. Like [83], they recover some accuracy by overlapping adders and show how this has minimal impact on the adder's power consumption, area, and delay.

*2.2.2 Multipliers and multiply-accumulators.* The second-most popular arithmetic unit to approximate is the multiplier or its combination with an adder, i.e., the Multiply-Accumulate (MAC) unit. We identify ten works that focusing on these with proposals ranging from an inexact compressor to reduced-complexity partial product encodings. Another three works consider larger multiplication-related circuits.

Like adders, multipliers can be implemented in many ways to achieve specific area and delay characteristics. The simplest designs are sequential: they generate partial products one by one and accumulate them over some cycles. Others generate partial sub-products and add them together in a tree-like architecture using compressors [106, 144]. More advanced designs generate all partial products at once and accumulate (*compress*) them concurrently, typically employing different encodings to reduce the number of partial products needed. The latter category is prominent in current research, as we will see. Zanandrea *et al.* [199] explore both accurate and approximate versions of these designs, showing that array-based multipliers achieve the lowest power consumption, tree-based multipliers have the lowest delay, and approximate Booth-encoded array-based multipliers produce the smallest errors. Following this classification, Mannepalli *et al.* [101] propose utilizing approximate RCAs for partial product accumulation in a sequential multiplier. They report reduced area, delay, and power, with no noticeable impact on output quality in an edge detection application.

Focusing on array-based multipliers, Moaiyeri *et al.* [106] and Salmanpour *et al.* [144] propose each an approximate compressor for partial product compression, one being simplified to only use *majority gates* and the other being approximated without full adders. Orthogonally, Baba *et al.* [17] present a compression scheme implemented with incomplete adders whose carries are not propagated within the partial product array. Instead, they are extracted to form an error recovery vector that is later added to the reduced partial product to form the final result using a RCA with maskable carries. Piri *et al.* [132] propose *input-aware* approximation of a multiplier, truncating its partial product array to produce satisfactory errors for a particular input distribution. The three works have somewhat different objectives, but all report improvements in power, area, and error.

Another four works consider Booth-encoded array-based multipliers. Shao and Li [154] describe a fixed-width multiplier with exact partial product generation but truncated reduction. Rather than simply discarding the truncated bits, they design a low-overhead error compensation unit that approximates the sum of the bits and generates a *signature* from certain partial product bits to identify how the error should be compensated. Along the same lines, He *et al.* [62] propose a probability analysis-based error compensation scheme for approximate partial product compression. Instead of a signature-based solution, they add particular, truncated partial product bits to the LSBs of the accurate partial product array. Both designs achieve reduced area and power at minor errors.

While the above two works use exact Booth encoding, it is also possible to approximate the partial product generation. Liu *et al.* [97] explore this and present two approximate radix-4 encodings that they apply column-wise. Zhu *et al.* [204] propose a radix-256 encoding that avoids the generation of so-called *hard multiples* (e.g., $\pm 3X$ requiring a carry-propagate adder) and apply it to the least-significant partial product rows. Approximating only the Booth encoders does not give rise to significant area reductions, but using a higher radix encoding does, as it decreases the number of partial products. As a result, [204] report reduced area, delay, and power compared to both exact designs and that of [154].

The approximate multipliers described above may also be applied in larger arithmetic circuits. Specifically, Imani *et al.* [64] propose an approximate floating-point multiplier that approximates the product of the mantissas by their sum, selectively re-executing the multiplication exactly if the MSBs of the sum show certain patterns. Osorio and Rodríguez [122] present an approximate Single Instruction Multiple Data (SIMD)-capable multiplier whose partial product array is truncated in a way that ensures reasonable operation in vectorized configurations. Like [62, 154], the multiplier uses exact partial products, but certain bits are implemented with maskable carries so as to enable SIMD operation. Lastly, Xiao *et al.* [183] implement a MAC unit with partial product compression and product reduction based on approximate compressors. Like [132], they apply approximations matching expected input patterns. All three works report improved energy efficiency with little impact on output quality in different applications.

*2.2.3 Others.* A handful of works consider other arithmetic operators. Firstly, in addition to their segmented adder, Kim *et al.* [83] also propose an approximate comparator utilizing their scheme for reduced carry propagation. Compared to its exact counterparts, it achieves a very low error rate but vast improvements in delay and energy consumption.

Secondly, two works propose different approximate squarers, i.e., symmetric multipliers whose operands are identical. Shao and Li [154] re-use their array-based design technique and propose a design with signature-based error compensation. More recently, Reddy *et al.* [136] propose three approximate squarers based on approximate Booth encoding. Their first design only applies this, the second combines the encoding with approximate adders, and the third adds an error recovery module to the second design. While Shao and Li [154] report a reduced mean error at the same hardware costs as comparable designs, Reddy *et al.* [136] achieve reductions in both area, delay, and error.

Thirdly, another two works present approximate dividers. Both propose several inexact subtractors (very similar to full adders) and utilize them in array-based dividers. Chen *et al.* [29] first present three different subtractors based on pass-transistor logic. Afterward, they compare the effects of different schemes for replacing exact subtractor cells with their approximate ones, reporting that triangle replacement, i.e., column-wise substitution of subtractor cells in the LSBs, leads to the best error characteristics. Jha and Mekie [66] compare CMOS-based subtractors with ones based on pass-transistor logic and conclude that the former performs the best, leading them to propose four CMOS-based subtractors. Both works show reductions in power and energy compared to an accurate divider.

Lyu *et al.* [99] describe how traditional implementations of $n^{\text{th}}$ roots based on the Coordination Rotation Digital Computer (CORDIC) algorithm are costly in terms of delay and area. As a solution, they propose using a piece-wise linear approximation combined with segmented and quantized, tabulated approximations of comprised sub-functions. This proposal enables implementation in a simple, pipelined architecture, which achieves similar error magnitudes as state-of-the-art CORDIC-based architectures, but at a much lower area and power consumption.

Finally, most reviewed works fail to account for aggregate errors arising from concatenated approximate arithmetic circuits with the same error polarity. Mazahir *et al.* [103] propose the concept of self-compensation – canceling out approximation errors by evenly mixing circuits with opposite error polarities – to solve this. They demonstrate their proposal on circuits comprising several adders and multipliers, showing decreased errors at negligible overhead.

## 2.3 Stochastic Computing

In some instances, resource constraints might prohibit implementing large arithmetic units operating on binary numbers like the ones reviewed above. Bit-serial processing is an alternative design strategy for such cases that drastically reduces area at the expense of longer execution time. Stochastic computing combines bit-serial architectures with AxC by operating on pseudo-randomly generated, finite-length bitstreams, reducing additions and multiplications to simple OR and AND operations. We find six works presenting stochastic computing techniques.

Firstly, Seva *et al.* [153] and Pamidimukkala *et al.* [127] propose further approximating stochastic computing architectures by truncating operands used for bitstream generation, reducing the size of the (often costly) Linear-Feedback Shift Registers typically used for this. In their first work, they statically truncate operands, keeping only some MSBs, while in their second work, they dynamically select bits following the most significant asserted bit. Both techniques achieve good results when evaluated on an edge detection algorithm.

Secondly, two works describe how stochastic computing does not allow for simple implementations of sub-tractors and dividers. As such, Bharathi *et al.* [20] propose a subtractor unit for so-called *scaled population* arithmetic – and approach in which the bitstreams are paired with an exponent to allow for greater numerical

range. Their implementation uses only well-established negation and addition operations. Yu *et al.* [198] describe a counting-based divider that combines deterministic bitstream generation and early termination. These features enable it to significantly outperform similar designs in a contrast stretch application. Both works report reductions in area, delay, power, and error.

Lastly, two works combine stochastic computing with traditional binary designs. Specifically, Joe and Kim [70] propose implementing approximate adders split in two, performing exact binary addition on the upper part and inexact stochastic addition on the lower part. The resulting design achieves much smaller errors than a lower-part OR-based approximate adder. Faraji *et al.* [48] apply a similar technique to constant-coefficient multipliers, utilizing a unary thermometer encoding for the lower part and a multiplexer for selecting an appropriate bias for the upper part before performing traditional binary addition. The thermometer encoding enables approximate multiplication using only wires. The authors demonstrate reduced area for most of their benchmarked configurations.

## 2.4 Other Techniques

Another eight works present fundamental techniques which do not fall under the above three categories. Regardless, we find they can be grouped into two classes based on their underlying idea: memoization or function approximation.

*2.4.1 Memoization.* Memoization is a technique often used in software to remember prior computations and later reuse their results. The same technique is applicable in hardware and provides opportunities for approximation. We find four works present related techniques: two are *static*, the others *dynamic*. While the static techniques tend to have lower overheads, they lack the adaptability of their dynamic counterparts. Later in Sec. 3.1.2, we will see dynamic memorization applied in General-Purpose Processor (GPP) architectures.

We first explore the static memoization proposals. Parhami [128] describes hardware complexity and difficulty of error analysis as the main drawbacks of existing AxC techniques. Motivated by advances in memory density, the author proposes implementing table-based approximate arithmetic circuits, which are both area- and power-efficient and enable easy error analysis; an interesting alternative to [103]. Jordan *et al.* [71] suggest using K-means to produce clusterings of input-output pairs for approximable code regions at compile-time and selecting the best matching instance using nearest-centroid classification at run-time. Both works describe how hierarchical tables allow for fine-grained quality control. The latter reports lower hardware overhead yet similar error metrics to a neural approximation approach.

Two contrasting proposals focus primarily on dynamic memoization techniques. They both describe how most circuit-level AxC techniques fail to achieve their expected benefits when used in FPGAs, suggesting that memory-based techniques can more efficiently utilize the hardened primitives common to FPGAs. Echavarria *et al.* [43] explore this idea and report benefits in terms of power and resource utilization. Sinha and Zhang [157] propose a High-Level Synthesis (HLS) post-processing step that generates and inserts static or dynamic memoization wrappers on given logic blocks. The static wrapper is configured at synthesis time, while the dynamic wrapper stores input-output pairs at regular intervals at run-time. Despite resource overheads, they also report power savings over an exact baseline.

*2.4.2 Function approximation.* The remaining four works present different approaches to implementing complex function approximation. The iterative nature of software-based implementations often slows down applications that rely heavily on such approximations. Hence, hardware acceleration can provide significant speedup. The reviewed implementations are rather diverse and, thus, difficult to group. del Campo *et al.* [38] notice that the accuracy of quantized non-linear functions is difficult to manage yet highly impactful. To resolve this, they propose using statically memoized, quantized Taylor expansions produced by an error analysis flow to achieve

Table 1. Select characteristics of the reviewed works on fundamental AxC techniques. AE = Autoencoder, SE = Square Error, ED = Error Distance, RED = Relative Error Distance. Prefixes: (M) = Mean, (R) = Root, and (N) = Normalized.

| Class | | Ref. | Evaluation | KPIs | | | | Error metrics | | | | |
|---|---|---|---|---|---|---|---|---|---|---|---|---|
| | | | platform | Power | Energy | Area | Delay | Rate | (M)AE | (R/M)SE | (M/N)ED | (M)RED |
| Circuit-level | **VOS** | [109] | Sim. (ASIC) | | x | x | | | | x | | |
| | | [166] | Sim. (ASIC) | x | | | | x | | | | |
| | | [163] | ASIC | | x | x | | | | x | | |
| | **Mems.** | [156] | Sim. (ASIC) | | x | | | | | | | |
| | | [53] | Sim. (ASIC) | x | | | | | | | | |
| | **Logic** | [192] | Sim. (ASIC) | x | x | | | | | | x | |
| Arithmetic | **General** | [103] | Sim. (anal.) | x | | | | x | | x | x | |
| | **Adders** | [194] | Sim. (ASIC) | x | | x | x | | | | | |
| | | [7] | Sim. (anal.) | x | x | x | | | | x | | |
| | | [193] | Sim. (ASIC) | x | x | | x | x | | | x | |
| | | [83] | Sim. (ASIC) | x | x | x | x | x | x | x | | |
| | | [42] | Sim. (ASIC) | x | | x | x | x | | x | x | x |
| | | [189] | Sim. (ASIC) | x | | x | x | x | | | x | x |
| | | [35] | FPGA | | | x | x | x | x | x | | |
| | | [116] | FPGA | x | | x | x | x | | | x | |
| | | [191] | Sim. (ASIC) | x | x | x | | | | | | |
| | **Mults.** | [154] | Sim. (ASIC) | | x | x | x | | x | | | |
| | | [97] | Sim. (ASIC) | x | | x | x | x | | | x | x |
| | | [106] | Sim. (ASIC) | x | | x | x | | | | x | |
| | | [64] | Sim. (ASIC) | | x | x | x | x | | | | |
| | | [17] | Sim. (ASIC) | x | | x | x | | | x | | x |
| | | [122] | Sim. (ASIC) | | x | x | | | x | | | |
| | | [62] | Sim. (ASIC) | x | x | x | | | x | | | x |
| | | [183] | Sim. (ASIC) | x | | x | x | | | | x | x |
| | | [101] | Sim. (ASIC) | x | | x | x | | x | x | | |
| | | [199] | Sim. (ASIC) | x | | | x | | | | x | |
| | | [144] | Sim. (ASIC) | x | x | x | x | x | | | x | |
| | | [132] | Sim. (ASIC) | x | | x | x | x | | | x | x |
| | | [204] | Sim. (ASIC) | x | | x | x | x | x | | x | x |
| | **Others** | [29] | Sim. (ASIC) | x | | x | | | | | x | |
| | | [66] | Sim. (ASIC) | x | x | | x | | | | | |
| | | [136] | Sim. (ASIC) | x | | x | x | x | | | x | x |
| | | [99] | Sim. (ASIC) | x | | x | x | | x | | | |
| Stochastic | **General** | [153] | Unspecified | | | | | | x | | | |
| | | [127] | Sim. (anal.) | | | | | | | x | | |
| | **Adders** | [70] | FPGA | x | | x | | | | x | x | |
| | | [20] | Sim. (ASIC) | x | | x | x | | | x | | |
| | **Mults.** | [48] | FPGA | x | x | x | x | | x | | | |
| | **Others** | [198] | Sim. (ASIC) | x | x | x | x | | | x | | |
| Others | **Memo.** | [157] | FPGA | x | | x | x | | | x | | |
| | | [128] | Unspecified | | | | | | | | | |
| | | [43] | FPGA | x | | | | | | | | |
| | | [71] | Sim. (ASIC) | | x | x | | x | | | | x |
| | **Func.** | [38] | FPGA | | | x | x | | x | x | | |
| | **appx.** | [142] | Sim. (anal.) | | x | x | | | | | | |
| | | [98] | Sim. (ASIC) | x | | x | x | | x | | | |
| | | [72] | FPGA | x | | x | x | | x | x | | |

given error bounds. They demonstrate the efficacy of their proposal by training Neural Networks (NNs) with approximated sigmoid activations.

Rust *et al.* [142] describe a lack of design methods for multi-variable function approximation. Their technique segments a function's input space, performs linear regression within each segment, quantizes the identified weights, and implements the result in a binary tree-style architecture without multipliers. The authors report large

Table 2. Applications used for evaluating *at least two* reviewed, fundamental AxC techniques. SNR = Signal to Noise Ratio, SSIM = Structural Similarity, RED = Relative Error Distance, SE = Square Error. Prefixes: (P) = Peak, (M) = Mean, and (N) = Normalized.

| **Application** | | **References** | **Application quality metrics** | | | | |
|---|---|---|---|---|---|---|---|
| **Class** | **Name** | | **(P)SNR** | **(M)SSIM** | **(M)RED** | **(N)MSE** | **Class. acc.** |
| **Media** | Edge detection | [64, 70, 101, 127, 153, 156] | x | | | x | |
| | Image smoothing | [43, 103, 156] | x | | | x | |
| | Image sharpening | [17, 189, 193] | x | | x | | |
| | Image multiplication | [97, 106, 144] | x | x | | | |
| | Background removal | [29, 66] | x | x | | | |
| | Change detection | [29, 66] | x | x | | | |
| | DCT for JPEG | [42, 48, 62, 71, 109, 116, 122, 166, 204] | x | x | | | |
| **ML** | *K*-means | [64, 71] | | | | | x |
| | *K*-nearest neighbors | [53, 64] | | | | | x |
| | NN training | [38, 64, 83] | | | | | x |
| | NN inference | [53, 64, 163, 183] | | | | x | x |
| **Others** | DSP tasks | [71, 132, 163] | | | x | x | |

area savings for most benchmarked functions compared to similar works. Similarly, Luong *et al.* [98] combine Lagrange interpolation with piece-wise linear approximations. Their hardware implementations combine statically memoized weights in LUTs with stochastic computing techniques for addition and multiplication, as seen in Sec. 2.3. As a result, they are comparable to binary implementations in terms of errors while being smaller and consuming less power.

Lastly and most recently, Kan *et al.* [72] propose using *bisection* NNs – networks with binary tree-like connections between neurons – for regression-based function approximation and present a reconfigurable hardware accelerator for this. They highlight their proposal's flexibility and area savings over other designs, albeit at the cost of greater errors.

In this section, we have reviewed works on fundamental AxC techniques and highlighted how and with which metrics these techniques are evaluated in Tab. 1. The normalized, reported power and area results are summarized in Fig. 4. The entries are ordered from the oldest to the most recent publication. Most of the techniques reduce overall power consumption, with some showing slight increases over comparable systems. Similarly, most reduce overall area, while [142, 163] trade increases in area for improved performance and energy efficiency.

## 3 APPROXIMATE COMPUTING-ENABLED HARDWARE ARCHITECTURES

Having provided an overview of fundamental AxC techniques, we now review general hardware architectures that implement some of them. We continue to provide a coarse, yet different, classification of works, now into three categories: GPPs, reconfigurable architectures, and Network on Chips (NoCs), as shown in Fig. 5. The works presented in this section emphasize hardware design, while in the subsequent section, the emphasis is instead on algorithms or one or more specific applications. Nonetheless, many techniques frequently re-appear. We provide an overview of commonly used benchmarks in Tab. 4 and short summaries of all works in the appendix, see Tab. 7. As before, we do not report any numerical results in text due to the difficulty of comparison.

### 3.1 Approximate General Purpose Processors

Modern GPPs are complex circuits and one of the most commonly used computing platforms that offer a variety of opportunities for approximation. Therefore, hardware and tool support in GPPs is a requirement for a broad adoption of AxC [82, 170]. The reviewed works apply five approximations: inexact arithmetic, memoization,

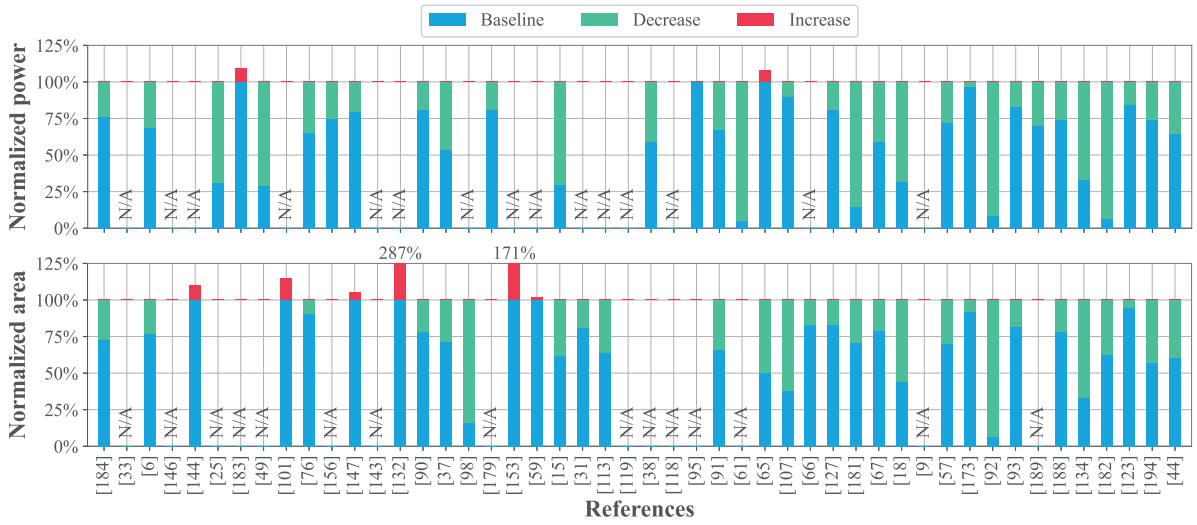


Fig. 4. Normalized power and area metrics from the works on fundamental AxC techniques in chronological order from left to right.

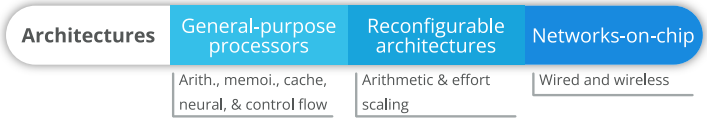


Fig. 5. Coarse classification of publications on AxC-enabled architectures.

approximate caches, neural approximation, and unreliable control flow. Several works combine these with fundamental AxC techniques.

*3.1.1 Arithmetic.* The first approximation opportunity in GPPs is their arithmetic and logic operations. This aspect is explored in two works, both exposing quality control in the Instruction Set Architecture (ISA). Venkataramani *et al.* [170] attribute the limited adoption of AxC in GPPs to the lack of error bounds in typically applied approximations. They present a vector processor comprising a mesh of approximate Processing Elements (PEs) and linear arrays of mixed-precision PEs, all implementing dynamic truncation. Evaluated on a set of compute-intensive applications, they show significant energy reductions. Focusing instead on a traditional Reduced Instruction Set Computer-style processor, Ndour *et al.* [113] explore the energy-error trade-off from using dynamically truncated integer instructions. Based on results from a test chip, they develop a mathematical model of a core's energy consumption analogous to Amdahl's law indicating only small savings can be expected in applications containing little *approximable* code.

*3.1.2 Memoization.* Energy consumption in GPPs is often dominated by memory and control instructions rather than arithmetic [82]. Thus, the benefits of approximating only arithmetic instructions are limited, calling for exploring alternatives; one of such being memoization that, in the context of GPPs, exploits approximate similarity between *computing instances* – e.g., procedure calls or loop iterations – to skip executing one or more instructions. In the context of GPPs, memoization means exploiting approximate similarity between *computing instances* – e.g., procedure calls or loop iterations – to skip executing one or more instructions. Chandrasekharan *et al.* [27] bridge the gap between approximate arithmetic and memoization by targeting costly floating-point instructions. They equip the Floating-Point Unit with dynamic memoization tables and enable/disable approximations using a custom instruction that adjusts the number of LSBs to ignore when performing lookups. With these additions, they report a noticeable speedup.

Approximately reusing results from individual instructions also only leads to limited performance gains. It is generally more beneficial to memoize and skip multi-instruction instances. This is explored in three other works implementing more involved schemes, all requiring ISA support. Firstly, Sato *et al.* [146] aim to solve two issues: a lack of frameworks for applying AxC to different GPP applications and large memory-related overheads of software-based memoization. Based on the design in [164], they propose an auto-memoizing processor that automatically memoizes function calls' inputs and results with minimal additions to the original architecture's tables. The approximations are enabled/disabled as in [27]. Secondly, He *et al.* [61] propose a *significance*-driven scheme consisting of two modules: a regression module of MAC-based PEs that manages linear models for input significance and logistic models for conditional branches; and a ternary Content Addressable Memory (CAM)-based cache of previous computation instances. Lookups in the CAM use exact branch vectors and significance-weighted, truncated distance measures between computing instances. They manage accuracy by maintaining multiple regression models for different input value domains. Lastly, Kim *et al.* [82] propose a full ISA extension, *Value Similarity eXtensions*, with corresponding micro-architectural changes for in-order pipelines. The extension allows for skipping entire sequences of instructions, loop perforation, and memoization, enabled by an extended data cache that compares cache line entries and generates similarity bits accordingly, and an instruction skip controller that extends the core's program counter logic. The skip controller also manages the core's operand selection logic to enable result reuse.

Despite its broader scope, the auto-memoizing processor of [146] only achieves speedup similar to the memoized floating-point instructions of [27]. Both the other designs [61, 82] report many times higher speedup with comparable error bounds, albeit in different applications, but as expected since they reuse larger computing instances.

*3.1.3 Memory.* Three other works explore approximating cores' accompanying memory hierarchies in different ways. Miguel *et al.* [104] approximate the last-level cache, exploiting approximate similarity between data cache lines to reduce cache size. Their architecture measures similarity using hashes of cache lines stored in the tag array and allows for associating multiple tag entries to a cache block. The degree of approximation is, thus, controlled by the number of bits used in the hashes; the fewer, the more cache lines are reused. These changes make reads, writes, allocations, and replacements more problematic, but the authors present solutions to them all.

Nongpoh *et al.* [117] instead approximate the cache coherence policy in many-core GPPs by relaxing coherence on approximable, shared data identified by automated sensitivity analysis. With the protocol implemented, a core can avoid reading updated cache lines owned by other cores and invalidating shared cache lines on writes. The proposal is supported in hardware by a custom store instruction, extensions to the GPP's directory-based coherence scheme, and extra cache(s) for recently invalidated cache lines.

Zhang *et al.* [201] suggest solving degrading restore time challenges in future DRAM by approximation. They propose three precision-aware scheduling techniques that dynamically adjust different row segments' restore times, ensuring that some segments operate reliably while others may give rise to random errors. Their schemes can improve performance by cleverly mapping low-significance bits to unreliable segments but require Operating System (OS) support and introduce hardware overheads for each row in the DRAM controller and the memory management unit.

Though distinct, the three proposals all trade off other metrics to reduce overall energy consumption: the similarity cache increases execution time [104], which is reduced by the approximated DRAM that in turn introduces randomness [201], and the results of [117] are polluted by unclear implementation details of the extra cache(s).

*3.1.4 Neural approximation.* Yet another three works consider neural approximation as an alternative to memoization, substituting compute-heavy kernels with NNs implemented in small, programmable accelerators [47]. Moreau *et al.* [110] attribute limited adoption of NN-based approximation to the complexity and tight integration

of historic accelerators. Instead, they propose a decoupled, FPGA-based NN accelerator based on systolic arrays for heterogeneous SoCs. The accelerator is memory-mapped, comprises a set of PEs, and implements microcode-based evaluation for low-overhead run-time reconfiguration over two Advanced eXtensible Interface-based processor-FPGA interconnects.

Grigorian *et al.* [57] aim to improve the reliability of neural approximations in RMS applications. They propose to ensure output quality at run-time through an iterative flow with increasingly complex NNs whose outputs are evaluated by application-specific Light-Weight Checks (LWCs) (like in [67]). To support their technique, the authors describe a simple accelerator architecture comprising a number of PEs with global configuration logic. This architecture is designed particularly for NoC-based accelerator-rich processors.

Song *et al.* [161] target maximizing invocations of approximations given some error bound. Rather than using LWCs, the authors use tiny NNs to classify computing instances as approximable and other NNs to approximate them, increasing their technique's coverage by using multiple of the latter. They present two strategies: one employing cascaded pairs of classifiers and approximators, and one with a multi-class classifier and a corresponding number of approximators. The latter is emphasized for avoiding the multi-classifier overheads of the former and the multi-approximator overheads of [57]. Like the above works, the authors also describe a parallel, PE-based accelerator that supports having both a classifier and several approximators loaded simultaneously.

Despite their implementation diversity, all three works use small Multi-Layer Perceptrons (MLPs), all rely on developer annotations, and two implement run-time quality management that re-computes approximations of insufficient quality exactly [57, 161]. They all report significant speedups and energy reductions, albeit, when evaluated differently.

*3.1.5 Control flow.* Returning to the core, Nongpoh *et al.* [118] also explore approximating control flow, using custom instructions to skip roll-back on a branch and load-value misprediction in speculative pipelines. Like in [117], they use automated sensitivity analysis to identify approximation opportunities: first, *individually* approximable data and branches, and then, *jointly* approximable branches. They provide hardware extensions needed to support the custom instructions, and their results show energy and execution time reductions at acceptable error bounds.

## 3.2 Approximate Reconfigurable Accelerators

We now turn our attention to generic, reconfigurable accelerator architectures. We review three works that present Coarse-Grained Reconfigurable Array (CGRA)-like designs that balance reconfigurability overheads, performance, and energy efficiency in highly parallel workloads. This renders them suitable for computing environments where bit-level reconfiguration is mostly unnecessary. The three works have many similarities: they consider the CGRA a processor-controlled accelerator, use a centralized configuration unit, support exact operation, and include extended memory architectures for efficiency. Their differences lie primarily in how they integrate AxC.

Specifically, Chippa *et al.* [33] propose a CGRA with run-time quality control, denoted *effort scaling*, implemented using clock gating-based truncation and VOS (refer to Sec. 2.1.1) of its PEs. A global feedback-driven controller manages the effort scaling by estimating the current output quality and tuning approximations towards given error bounds. The authors use a sensitivity analysis flow for detecting approximable kernels of programs, as in [32].

The two other works implement PEs whose FUs comprise both exact and approximate adders and multipliers (refer to Secs. 2.2.1 and 2.2.2) and select between these using configuration bits. They both rely on developer annotation of approximable functions and their error bounds, and both provide tools for searching for optimal approximation configurations. Nevertheless, the works differ in that Akbari *et al.* [2] implement PEs with only one approximate adder and multiplier and control the output quality by enabling/disabling subsequent correction

Table 3. Select characteristics of the reviewed works on AxC-enabled architectures. Note that we specify application *run-time*.

| Class | | Ref. | Evaluation platform | KPIs | | | | Fundamental techniques | | | |
|---|---|---|---|---|---|---|---|---|---|---|---|
| | | | | Power | Energy | Area | Run-time | Circuit-level | Arithmetic | Stochastic | Others |
| GPPs | **Arith.** | [170] | Sim. (ASIC) | x | x | | | x | | | x |
| | | [113] | ASIC | x | | | | | | | x |
| | **Memoiz.** | [146] | Sim. (anal.) | | | | x | | | | x |
| | | [61] | Sim. (ASIC) | | | | x | | | | x |
| | | [27] | FPGA | | | | x | | | | x |
| | | [82] | Sim. (ASIC) | | | | x | | | | x |
| | **Memory** | [104] | Sim. (anal.) | | x | | x | | | | |
| | **appx.** | [201] | Sim. (anal.) | | x | | x | x | | | |
| | | [117] | Sim. (anal.) | | x | x | x | | | | x |
| | **Neural** | [110] | FPGA | | x | | x | | | | |
| | **appx.** | [57] | Sim. (ASIC) | x | x | | x | | | | |
| | | [161] | Sim. (anal.) | | x | | x | | | | |
| | **Control** | [118] | Sim. (anal.) | | x | | x | | | | |
| CGRAs | | [33] | Unspec. | | x | | | x | | | x |
| | | [2] | Sim. (ASIC) | | x | | x | | x | | |
| | | [41] | Sim. (anal.) | x | | | | | x | | |
| NoCs | **Wired** | [111] | Sim. (ASIC) | | x | | | x | | | |
| | | [139] | Sim. (anal.) | | x | | | | | | x |
| | | [185] | Sim. (anal.) | | x | | x | | | | |
| | | [107] | Sim. (anal.) | | x | x | | x | | | x |
| | **Wireless** | [15] | Sim. (anal.) | | x | | | x | | | |
| | | [49] | Sim. (anal.) | | x | | x | | | | |

units, further specified in [3]. Dickerson *et al.* [41] implement several approximate adders and multipliers with different error characteristics and select between them at run-time. Both works also implement dynamic quality control circuitry that monitors the execution and selects between predetermined operating modes.

All three works report significant energy savings and/or power reductions. Notably, the benefits of cross-layer approximations, and VOS in particular, are highlighted in [33]. The works also indicate the necessity of automated tooling for sensitivity analysis.

## 3.3 Approximate Networks-on-Chips

Modern multi-core processors and SoCs often implement high-speed NoCs for core-to-core and core-to-memory communication. Traditionally, these networks have been wired, but increasing core frequencies have made the interconnect a bottleneck, leading to the proposal of using wireless on-chip networks when ultra-low latency operation is needed, e.g., for coherence. Either NoC type offers approximation opportunities, yet we distinguish between the two here, noting that despite the main focus on wireless NoCs in [15] and [49], all six reviewed works include wired NoCs in their proposals. Moreover, the works all agree on their main motivation: poor scaling, both in terms of wire delay and power consumption relative to logic and packet latency in many-core chips. We first consider techniques for wired NoCs.

*3.3.1 Wired NoCs.* Five works present AxC techniques for wired NoCs. Firstly, Reza and Ampadu [139] propose truncating packets containing developer-annotated data before transmission and zero-padding them on arrival. The involved network interfaces employ a per-packet error bound that enables low-complexity resilience evaluation, disregarding accumulated errors. Secondly, Ascia *et al.* [15] propose to selectively VOS (refer to Sec. 2.1.1) the interconnect links when performing stores and loads of developer-annotated data (listing automatic sensitivity analysis as future work in [16]). Thirdly, Momeni and Shahhoseini [107] extend this concept to 3D

Table 4. Applications used for evaluating *at least two* reviewed, AxC-enabled architectures. Abbreviations as in Tab. 2.

| Application | | References | Application quality metrics | | |
|---|---|---|---|---|---|
| **Class** | **Name** | | **PSNR** | **SSIM** | **Class. acc.** |
| **Benchmark suite** | AxBENCH [47] | [41, 61, 104, 107, 110, 113, 118, 185, 201] | x | x | |
| | PARSEC [21] | [49, 104, 107, 110, 118, 185] | x | x | |
| | SPLASH [143, 181] | [15, 49, 117] | x | | |
| **Media** | Image processing | [2, 27] | | x | |
| **Machine learning** | *K*-means | [33, 170] | | | x |
| | SVM inference | [33, 170] | | | x |
| | NN inference | [82, 170] | | | x |

NoCs. In addition to the links, they also apply VOS to the routers and truncate packets based on current network congestion.

Fourthly, orthogonal to [15] and [107], Najafi *et al.* [111] aim to improve the reliability of wired links under VOS while avoiding the overheads of Forward Error Correction (FEC) and Automatic Repeat Request (ARQ). They propose two integer value coding schemes that swap and invert wires and eliminate undesirable bit patterns to reduce crosstalk. Doing so makes neighboring wires less likely to stabilize at wrong logic values.

Finally, Xiao *et al.* [185] extend upon [139]. They propose applying dynamic packet dropping and approximation according to offline error and congestion modeling from developer annotations and managing them at run-time using a set of quality controllers. The controllers sample the network congestion and approximation impact of specific packets to drop or approximate low-impact packets. Dropped packets are estimated as the mean of their predecessor and successor packets, while approximated packets are truncated before transmission and zero-padded on arrival.

Four of the five works report potential for energy and latency savings [15, 107, 139, 185]; the remaining work [111] instead highlighting reduced errors in a set of applications. Neither [139] nor [107] evaluate the errors introduced by their schemes, while [15] demonstrate small application output errors and the thorough error evaluations of [185] show large savings by nearing, but always satisfying, given error bounds. Finally, we note that [139, Fig. 4] is wrong: latency is shown to be consistently lower when using approximation but has a greater mean than the exact system.

*3.3.2 Wireless NoCs.* Two works focus on wireless NoCs. Ascia *et al.* [15] propose adapting their scheme for VOS wired links to instead dynamically select between transmission power levels on wireless links (explored in-depth in [16]). Fernando *et al.* [49] present a more involved technique that extends the architecture of [1], exposing globally-synchronized broadcast memories to the software to enable data sharing via these and their wireless links. Their design measures network contention and uses it to dynamically adapt the transmission protocol to different traffic patterns and enable/disable approximations on developer-annotated data. They support several approximations, including packet dropping, approximate locks, and skipping negligible updates. The two works report significant savings over exact wireless NoCs and approximate wired NoCs, the latter also demonstrates noticeable speedup in several applications [49].

In this section, we have reviewed general AxC-enabled architectures, specifically GPPs, CGRAs, and NoCs. Like in Sec. 2, we have highlighted typically used benchmark applications in Tab. 4 and performance metrics in Tab. 3. Fig. 6 summarizes normalized, reported results in energy and run-time. Essentially all works report improved energy consumption, the exception being [104] that trade off small run-time overheads for reduced energy.

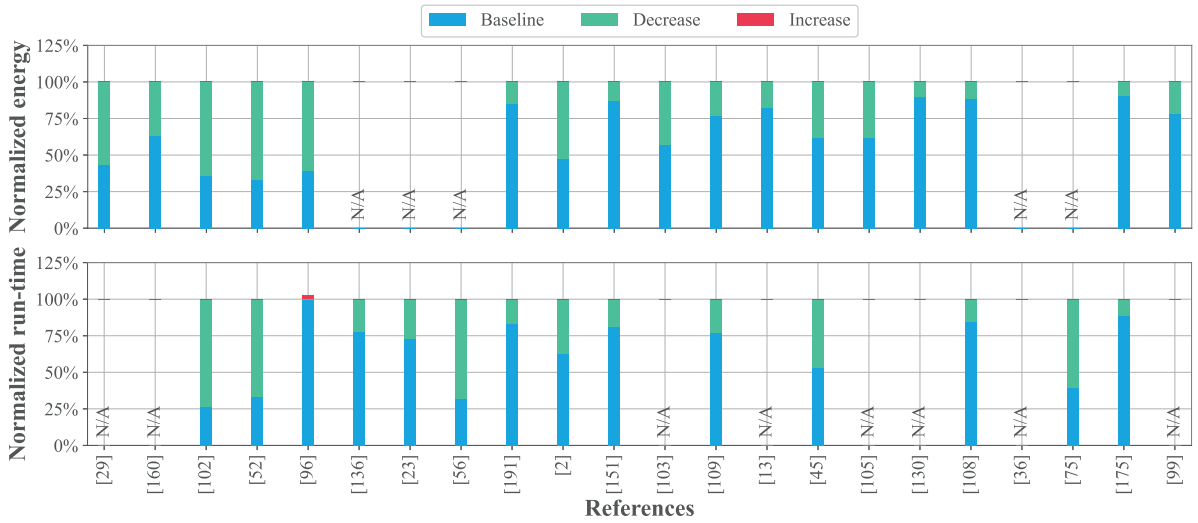


Fig. 6. Normalized energy and run-time metrics from the works on AxC-enabled architectures, in chronological order.

## 4 APPLICATIONS

After reviewing fundamental AxC techniques and AxC-enabled architectures, we now focus on applications of AxC, classifying works into five categories: Machine Learning, image processing, video processing, reliability, and other applications (see Fig. 7). When relevant, we reference works from the prior sections. We give short summaries of all works in Tab. 8 and continue to refrain from reporting numerical results in text.

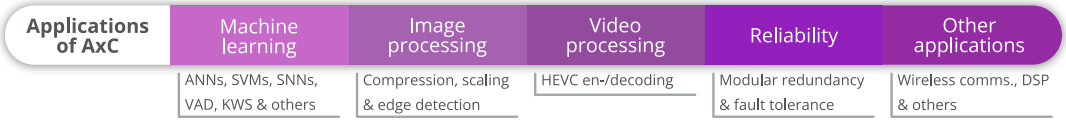


Fig. 7. Coarse classification of publications on applications of AxC.

### 4.1 Machine Learning

The reviewed works reveal that AxC is most frequently applied to ML. This is no surprise, as ML applications are known to be error-resilient and compute-heavy and, thus, provide apparent opportunities for optimization [171]. We also notice a trend toward designing architectures and techniques for the low-power acceleration of ML suitable for Edge computing. In this section, we divide works into three sub-classes: generic accelerators for NNs, accelerators for other ML models, and application-specific accelerators. When possible, we relate these to works in Secs. 2 and 3.1.4.

*4.1.1 Generic NN architectures.* Different NN models require specific architectures to be efficiently accelerated. As such, we distinguish between accelerators for MLPs or Deep Neural Networks (DNNs), Convolutional Neural Networks (CNNs), and Spiking Neural Networks (SNNs), of which the former three are well-known algorithms but the latter less so. As such, we also provide a brief introduction to SNNs before covering related papers.

*Multi-Layer Perceptrons and Deep Neural Networks.* Kim *et al.* [80] propose approximating the often unnecessarily costly MACs used in MLP accelerators. They present a design (like [110, 137]) that combines this with on-chip training, synapse skipping, and dynamic bit-width adaptation of small-gradient synapses. Beyond showing significant power and area savings, they describe how training MLPs at lower precision enables inference with more quantized weights.

Dataflow is a crucial concept for DNN accelerators. It determines in which order memory and arithmetic operations occur, often greatly affecting achievable throughput. Three works explore this in different ways, describing PE-based accelerators like in [110, 161]: Tu *et al.* [165] suggest accelerating individual DNN layers with different dataflow scheduling, specifically *full paralleling*, *neural extension*, or *computation extension*; Reddy *et al.* [137] focus on optimizing memory accesses to minimize initialization latency and maximize bandwidth utilization, targeting matrix-matrix operations implemented in FPGAs; and Wu and Miguel [182] propose supporting different dataflows with extra multiplexers in the PEs. The former two describe in detail how their architectures implement multi-level memory hierarchies, streaming-interfaced external memories, quantized MACs, and global control units. They evaluate their designs with neural approximation workloads and show improved power efficiency [137, 165]. The latter only present their quantized PE design and evaluate their architecture for DNN-based function approximation, showing improved energy efficiency at the expense of accuracy [182].

Sarwar *et al.* [145] explore the benefits of cross-layer AxC techniques. They combine pruning with inexact multipliers and VOS embedded SRAMs, giving rise to drastic energy savings. Venkataramani *et al.* [171] present an industrial perspective on the topic, outlining techniques explored while developing their accelerator (building upon [170] and akin to [33, 165]) and its corresponding tool flow. Their accelerator supports reduced-precision floating-point, integer, and binary operations. Their tool flow enables precision scaling and gradient compression during training.

*Convolutional Neural Networks.* Another five works focus on CNNs. Castro-Godínez *et al.* [24] consider the `im2col` function used to convert convolutions into matrix multiplications. They propose using approximate adders for re-ordering the matrix entries and present an architecture for the operation. Klemmer *et al.* [84] propose an approximate dot product encoding for binary input operands. They split dot products into a crossbar-style encoding of positive and negative accumulation parts, whose connections are determined by network weights. Multiplying inputs by the two binary-entry accumulators and summing the products gives the final result. Like [80, 145, 171], they apply re-training to minimize accuracy degradation. Both report significant speedup at negligible accuracy degradation. Yang *et al.* [190] propose dynamically masking some input pixel LSBs to reduce switching activity, passing in only segments whose MSB is the most significant asserted bit when enabled. This allows for maintaining accuracy while saving some energy.

Lastly, Wang *et al.* [178] and Gong *et al.* [56] present architectures for high-throughput CNN acceleration: the former being a SIMD-style architecture with support for sub-word operations, and the latter implementing PEs with approximate adders and LUT-based multipliers. In addition to their architecture, the former present an offline tool flow for adapting networks to quality constraints by exploring filter resizing, pruning, and precision scaling. The latter proposes early termination (like [67]) in combination with model compression through reduced filter sizes, weight thresholding, quantization, and optimized convolutions. Both works report improved energy efficiency over other accelerators, albeit without implementation details or when evaluated on a relatively smaller technology node.

*Spiking Neural Networks.* SNNs are widely considered the third generation of NNs, replacing first-generation threshold-based and second-generation, non-linear, continuous activation-based networks such as the above [100]. They are event-based, self-triggered networks whose (leaky) Integrate-and-Fire (IF) neurons model the integration of incoming, asynchronous spike events over time, producing outgoing spikes once a certain threshold is reached, well illustrated in [141, Fig. 3]. Owing to their event-based nature, SNNs are expected to enable highly energy-efficient implementations, yet the most popular hardware platform for ML – Graphics Processing Units – fails to leverage these benefits [141]. Instead, researchers and large corporations have proposed bespoke, integrated hardware implementations, the arguably most famous examples are TrueNorth [4] (a chip of parallel *neurosynaptic* cores) and Loihi [36] (a chip of configurable *neuromorphic* cores). Both are highly parallel and capable of simulating millions of neurons at once. Despite encoding spikes in an *address event representation*, the spikes' binary nature

reduces multiplications to simple additions, enabling the architectures to avoid costly multipliers at the expense of more complex control logic [131]. More recent efforts have focused on emerging technology-based designs, particularly in-memory and mixed-signal analog accelerators, both of which are inherently approximate [141].

The architectures of [4] and [36] and their predecessors have undoubtedly inspired new designs, including those reviewed here. Wang *et al.* [174] present a PE-based accelerator that supports online training of a *liquid state machine* SNN. Their architecture implements the leaky IF neuron model, uses approximate adders, and power-gates low-activity PEs to reduce switching power. Sen *et al.* [151] problematize parallelizing acceleration of SNNs owing to their irregular memory access patterns and many negligible-impact neuron updates, inflexibly solved by implementing one PE per neuron in [174]. They propose evaluating neurons in a time-multiplexed manner and assigning them each an approximation level – high levels meaning normal operation and low levels leading to synapse skipping (as in [80]) – that is updated periodically depending on spike activity, enabling early termination in the case of just one active output neuron remaining. Both works report significant energy savings at negligible accuracy degradation over exact baselines.

*4.1.2 Other generic ML architectures.* NNs are not the only ML models that can benefit from AxC, the feature also applies to Hyperdimensional Computing (HDC) and Support Vector Machines (SVMs). Neither model type appears as frequently as NNs, yet the former is currently gaining traction due to its one-shot learning capabilities and potential for in-memory acceleration [76]. Life for SNNs, we provide brief introductions to the models before covering related works.

*Hyperdimensional Computing.* As we described before, the underlying idea of SNNs is to accurately model the neuronal activity in biological brains. HDC abstracts away neurons and models distinct concepts as points in a high-dimensional space [73]. Originally motivated by its tolerance against noise from unreliable memory cells, it also has benefits in terms of training time and operation complexity yet suffers from lower achievable accuracy [54]. In HDC, a basis of random *hypervectors* is first selected to represent all symbols in a dataset and stored in an *item memory*. Classifying an input datum involves encoding it by combining basis vectors with three simple operations: *bundling* (element-wise addition), *binding* (element-wise multiplication), and *permutation* (shuffling), and comparing the result against learned *prototype* vectors, each representing an entire class, stored in an *associative memory* [54]. The high dimensionality and randomness enable learning these prototype vectors from relatively few training examples [73]. Like SNNs, recent efforts have explored in-memory accelerators for classification [76].

The process of searching through an associative memory of prototype vectors is costly, requiring many memory and comparison operations. Imani *et al.* [65] consider this and propose three CAM-like architectures designed to calculate the distances between a search vector and stored vectors. Their designs are fully digital, partially memristive, and fully analog, and their results reveal the fully analog design as the most area and energy efficient. Khaleghi *et al.* [78] instead propose approximating the distance calculations in FPGA-based designs using inexact adders and majority operators. Moreover, they reduce the encoding complexity by using rotated versions of a single vector instead of multiple unique vectors. The authors show significantly reduced area and energy consumption, disregarding this optimization.

*Support Vector Machines.* SVMs are another type of classifier, designed to work with two classes [34]. They model examples as points in a high-dimensional space and train a *soft maximum-margin hyperplane* that best possibly separates the two classes [115]. The plane is represented by a combination of weighted training examples known as *support vectors* [34]. Classification involves computing a *decision function* using the support vectors and a predefined *kernel* that is commonly either linear, polynomial, or Gaussian [25]. SVMs generally require only a few support vectors to achieve reasonable accuracy [34] but suffer from long training times to select these as dataset size grows [25].

van Leussen *et al.* [168] note that SVMs are popular in battery-driven devices that demand high energy efficiency. They propose using quantization and approximate multipliers (like [103]) and present an accelerator architecture that implements these. Interestingly, they notice a greater accuracy degradation from quantization than from approximate multiplications. Zhou *et al.* [203] extend upon this and utilize approximate adders and multipliers. While the former supports linear and higher-order polynomial kernels, the latter's architecture is limited to Gaussian kernels. Lastly, Ferretti *et al.* [50] support several kernels and combine approximations by quantizing data and weights and reducing the number of features and support vectors. All three works report energy and area savings over accurate baselines.

*4.1.3 Application-specific ML architectures.* One application domain appears particularly popular from the reviewed papers: speech recognition. As such, we first cover works in this domain.

*Speech recognition.* Liu *et al.* are motivated by the low-power requirements of always-on Voice Activity Detection (VAD) and present two different accelerator designs targeting this. In [92], they propose a DNN accelerator that uses two types of PEs: some implementing approximate multipliers (like in [145]) and some implementing delay-based adders for accumulating partial results. However, their more recent work [89] proposes an alternative digital accelerator operating on features extracted from an analog signal processing flow. They avoid implementing costly multipliers by utilizing Binarized Weight Networks (BWNs). They further approximate their evaluation using dynamic-precision approximate adders and adaptive bit-widths determined by a SNR estimate derived from the inputs. The authors report improved power efficiency and improved classification accuracy over related works.

In addition to their works on VAD, Liu *et al.* also present several accelerators for Keyword Spotting (KWS). In [93], they propose a PE-based CNN accelerator. They adaptively quantize both inputs and weights during training to find the minimal number of bits needed to achieve satisfactory accuracy. To support these in hardware, they design two types of PEs (like [92]), implementing either digital or approximate voltage-based multipliers. In line with this, they propose a SIMD-style multiplier for quantized CNNs [94]. Their design combines approximate Booth encoding (like in [62, 154]) with approximate adders for partial product reduction. In their other works, they present PE-based accelerators for BWNs. Specifically, in [91], they present a fully digital design that uses fixed-width data and the inexact adder of [89]. The architecture in [86] is very similar, yet it replaces inexact digital adders with analog delay-based ones, mitigating the corresponding effects through a noisy training flow. Lastly, in [87], they extend upon their design from [91] by adding support for dual-voltage operation with error-biased inexact adders. In all five works, the authors report greatly reduced power consumption at the same classification accuracy as related works.

Despite the chronology of these publications, the two last works somewhat combine elements of the above more recent publications. Firstly, Liu *et al.* [90] propose a full speech recognition flow implementing both a BWN-based VAD/KWS module and a Long Short-Term Memory (LSTM)-based speech recognition module. They quantize all data and implement both modules with each their style of PEs. Specifically, they use the analog delay-based adders of [92] for the KWS module's PEs and digital approximate multipliers with error recovery for the speech recognition module. Lastly, Jo *et al.* [69] also propose a hardware accelerator for LSTMs. In addition to using approximate multipliers like [90], they exploit the temporal similarity between inputs to the LSTM cells to determine which updates to approximate or completely skip. Their accelerator implements sparsity-awareness, making it capable of skipping memory accesses to zero-weighted inputs. The two works present significant improvements in power efficiency. We note that the results in most of Liu *et al.*'s works [87, 89–93] are biased by the use of a smaller technology node than in their related works.

*Other ML applications.* The remaining six works focus on applications or techniques that do not naturally fit into the above categories. First, Kim and Shanbhag [81] propose an AxC-enabled stereo image matching

(used in machine vision) accelerator with Algorithmic Noise Tolerance (ANT), an alternative to, e.g., self-compensation [103]. Their message passing architecture uses ANT to mitigate errors from approximate addition and voltage over-scaling. The authors show potential power reductions at the cost of (significant) overheads for implementing ANT-related estimation modules.

Second, Kar *et al.* [75] consider low-power always-on anomaly detection. They propose a hardware accelerator for an ensemble of one-class classifiers, i.e., single-layer NNs with a single output neuron. Their architecture supports online training, replaces weight memories in the first layer with pre-determined pseudo-random number generators, and evaluates neurons in time-division multiplexed MAC units. Furthermore, it can activate only a subset of classifiers and quantize their computations when inputs are easily distinguishable from anomalies. The authors show that the approximations increase energy efficiency at no noticeable drop in detection accuracy.

Third, Chen *et al.* [30] consider communication in NoCs used for image classification. They propose a scheme that scales the contrast of input images to reduce the bit-width of pixels and quantizes floating-point weights and activations to 8-bit integers. They support these approximations in the NoC's network interfaces, thus reducing data transfers. An online quality manager keeps track of output quality and adjusts approximations accordingly. The authors evaluate several image classification NNs and show reduced network latency and power consumption at little accuracy loss.

Fourth, Li *et al.* [85] propose using stochastic computing to reduce the hardware cost of NN accelerators. Specifically, they propose a neuron model that accumulates several incoming bit-streams concurrently and needing no activation function due to how weights are encoded in the stochastic domain. They evaluate their proposal on an MLP and a CNN and show greatly reduced area and power consumption at no accuracy degradation.

Fifth, Wang *et al.* [179] present a PE-based processor architecture designed to accelerate transformer NNs. The design combines self-adapting multipliers with inexact Booth encoding (like [62, 154]) and inexact compressors (like [17, 106]), speculation on weight sparsity, and out-of-order scheduling to maximize PE utilization and fully exploit the transformer's error resilience. These optimizations enable the architecture to achieve improved energy efficiency over related works.

Last, and orthogonal to aiming for power/energy and latency savings through AxC, Guesmi *et al.* [58] propose applying it to reduce CNNs' vulnerability to *adversarial attacks*. Such attacks involve manufactured inputs that closely resemble real inputs but intentionally cause inference failure. AxC introduces input-dependent noise and makes it more difficult to manufacture adversarial inputs efficiently. Specifically, the authors apply an approximate-mantissa floating-point multiplier and find that it greatly reduces attack transferability and success rate.

## 4.2 Image Processing

As seen in the prior sections, highlighted in Tabs. 2 and 4, image processing applications are frequently approximated. Ten works fall into this category, the first four presenting approximate DCT implementations, common to image compression algorithms. As these algorithms are inherently lossy, it is natural to consider approximating parts of them. Almurib *et al.* [9] describe a three-step framework for developing high-efficiency implementations of the DCT, namely: 1) select a low-complexity algorithm, 2) filter out high-frequency components, and 3) apply AxC techniques such as truncation and inexact adders. They report an extensive error analysis of the algorithm using different approximate adders. Similar ideas motivate Snigdha *et al.* [159], who focus on a multiplier-less DCT algorithm. Provided with a quality factor and an error variance budget, they find the optimal number of LSBs to approximate using an ILP solver. Xing *et al.* [186] similarly approximate the constant DCT weights, apply thresholding to intermediate pixel values, and implement inexact adders. Wang *et al.* [180] instead propose replacing the DCT algorithm with neural approximation (refer to Sec. 3.1.4). To accelerate inferences, they present a PE-based architecture (like in [137, 165]) with distributed memories and activation function approximation [38],

all optimized for operation at NTV (refer to Sec. 2.1.1). All four works report that significant energy savings are achievable with little visual impact on compressed images.

Another four works apply AxC to edge detection. de Oliveira *et al.* [37] first explore the use of inexact adders in the Gaussian and gradient filters common to many edge detection algorithms, highlighting the potential savings this enables. Their subsequent work, Soares *et al.* [160], specifies an algorithm to assign approximation levels to adders in the filters based on estimated input magnitudes. The authors also provide a full datapath that approximates gradient magnitude and direction computations to be multiplier-free. Their most recent work, Monteiro *et al.* [108], combines these proposals with varying filter sizes to enable area-power-quality trade-offs. Orthogonally, Usami *et al.* [167] propose using a simple memoization scheme – a single-entry version of [43] – that skips filter convolutions by reusing the most recently computed value if the inputs are approximately similar. The former three works report area and power reductions, while the latter shows some hardware overheads but reduced energy consumption.

The last two works focus on different algorithms. Siva and Jayakumar [158] present a deeply pipelined architecture that implements a bi-linear interpolation scaling algorithm. They approximate the algorithm's edge detection and sharpening filters by using approximate multipliers but fail to achieve improvements over related works. Yao *et al.* [195] consider an architecture for bilateral denoising filters. They estimate filter convolutions using piece-wise linear models and normalize pixel values with an inexact divider, significantly improving throughput compared to related works.

## 4.3 Video Processing

Another seven works present AxC applied to video processing applications. Palumbo and Sau [126] motivate research in this direction by varying user or system requirements regarding quality, power consumption, etc. Having surveyed related works, they argue for flexible software and hardware implementations of video-related algorithms – specifically for Advanced Video Coding (AVC) and High-Efficiency Video Coding (HEVC) – enabling run-time reconfigurability.

In line with this, four works present AxC applied to the HEVC algorithm. The first two consider its *motion estimation* encoding step. El-Harouni *et al.* [44] label this step as cumbersome owing to its many Sum-of-Absolute-Differences (SAD) computations between sub-frames. They exploit its error resilience in a heterogeneous accelerator architecture that implements tiles of unrolled SAD operations with different approximate adders used for some LSBs. A controller switches between the tiles and manages their memory accesses according to user-provided quality constraints, power-gating any unused tiles. Paltrinieri *et al.* [125] present a more static design and evaluate the benefits of replacing subsets of its adders/subtractors with approximate alternatives. Both works report power and area savings, although those of [44] are not matched by [125]. The other two consider the HEVC algorithm's *fractional pixel interpolation* decoding step that involves compute-heavy Finite Impulse Response (FIR) filters for both *chroma* (color) and *luma* (brightness) channels. Sau *et al.* [147] propose a reconfigurable architecture in which a number of filter taps may be skipped. They insert multiplexers and clock-gating logic into an existing interpolation circuit targeting FPGA, enabling low-overhead reconfiguration without reprogramming the FPGA. Preatto *et al.* [133] propose an alternative architecture implementing reconfigurable filters with adapted weights for different numbers of filter taps. Moreover, they use approximate adders in their approximate luma filter implementation. Both works report that power consumption is proportional to the number of filter taps used and note some overheads from the introduced reconfiguration logic.

The remaining two works are much more diverse. Schaffner *et al.* [149] describe how most video processing algorithms boil down to least-squares problems. With this in mind, they present an approximate direct solver architecture based on Cholesky decomposition that prunes insignificant intermediate computations, giving rise to significant run-time savings. Qiao *et al.* [135] instead propose a hybrid cache architecture for approximating

chrominance pixel values in AVC video algorithms. The approximate cache maps several indices to the same entry (like [104]), increasing effective cache size and reducing external memory accesses for chrominance data.

## 4.4 Reliability

Yet another seven works consider applying AxC to fault tolerance, five of which trade off fault detection guarantees for reduced duplication overheads in $N$-Modular Redundancy (NMR) techniques. Chen *et al.* [28] aim to resolve *fail-silent* faults in majority-voting-based NMR systems, i.e., the acceptance of a majority of incorrect results. They propose two approximate voting schemes for Double Modular Redundancy (DMR) and Triple Modular Redundancy (TMR) systems, both involving comparing some most-significant input bits to a given threshold to determine their validity. If no input is classified as valid, the schemes flag an error; otherwise, an approximate result is produced by averaging or voting.

The other four works approximate the redundant modules. Rodrigues *et al.* [140] precision-scale the modules, reducing the width of their datapaths. In addition to reducing their area, doing so also reduces the number of critical wires that lead to output errors if affected by soft errors. Anajemba *et al.* [12] propose replacing the two redundant modules in TMR systems with over- and under-approximated versions of the original. They introduce more 1s or 0s in their truth tables, simplifying the minterms required to represent them in SOP form while satisfying given error constraints. Combining over- and under-approximated modules implies that at least two modules produce exact outputs in the presence of no faults. Deveautour *et al.* [39] propose an approximate Quadruple Modular Redundancy (QMR) scheme. They randomly split a design's outputs into four equally-sized subsets, removing one subset and its fan-in cone from each to generate four approximate circuits. The modules' outputs are combined using four majority voters: one for each subset of outputs. The resulting system guarantees single-fault coverage. Lastly, Nazar *et al.* [112] propose approximating all modules in a DMR system and utilizing the saved area for implementing more modules, improving system throughput. They mitigate the effects of such broad approximations through a heuristic optimization algorithm and by using the voting schemes of [28].

Four works report positive results: the voting schemes of [28] almost consistently achieve higher output quality than an exact TMR scheme; precision scaling is shown to improve reliability against accumulated faults [140]; reduced area and improved energy efficiency is reported in [12]; and decreased error effects from faults are reported by [112]. Contrarily, [39] reveal uncertain benefits of approximate QMR, improving fault tolerance only for *some* circuits.

The two remaining works present orthogonal ideas. Verdeja and Li [172] focus on redundant general-purpose computing systems and propose to exploit application fault tolerance by skipping crashes in non-critical code regions, avoiding the long run-time overheads of restarting the application. Their scheme requires OS and hardware support but achieves significant performance and energy efficiency improvements. Wang *et al.* [177] combine AxC with soft error tolerance by proposing a data format consisting of an error-correctable control part and a parity-protected data part. They suggest truncating 32-bit numbers to their two most significant, non-zero 4-bit segments and assigning them pre-computed control signals. The authors report improved output quality under the influence of soft errors.

## 4.5 Other Applications

Lastly, we review thirteen works that do not naturally fall into any of the above categories. Instead, we group these works relating to wireless communication, Digital Signal Processing (DSP) applications, machine vision, and others.

*4.5.1 Wireless communication.* Six of the ten works focus on different aspects of wireless communication. Sen *et al.* [152] notice a systematic nature of errors introduced at the physical layer of wireless channels and aim to reduce these without resorting to FEC and ARQ. They propose making the transmitter aware of the errors

Table 5. Select characteristics of the reviewed works on applications of AxC. Note that we specify application *run-time*.

| | Class | Ref. | Evaluation | KPIs | | | | Fundamental techniques | | | |
|---|---|---|---|---|---|---|---|---|---|---|---|
| | | | platform | Power | Energy | Area | Run-time | Circuit-level | Arithmetic | Stochastic | Others |
| ML | **NNs** | [174] | FPGA | x | x | x | | | x | | |
| | | [80] | Sim. (ASIC) | x | | x | | | x | | x |
| | | [178] | Sim. (ASIC) | | x | | | | | | x |
| | | [151] | Sim. (ASIC) | x | x | x | | | | | |
| | | [145] | Sim. (ASIC) | | x | | | x | x | | |
| | | [165] | Sim. (ASIC) | x | | x | | | | | |
| | | [56] | Sim. (ASIC) | x | | | x | | x | | x |
| | | [24] | FPGA | | x | x | x | | x | | |
| | | [171] | ASIC | | x | | | | x | | x |
| | | [84] | FPGA | | | x | x | | | | x |
| | | [137] | FPGA | x | | x | x | | x | | x |
| | | [182] | Sim. (anal.) | x | | x | | | | | x |
| | | [190] | Sim. (ASIC) | x | | x | | | | | |
| | **SVMs &** | [168] | Sim. (ASIC) | | x | x | | | x | | x |
| | **HDC** | [203] | Sim. (ASIC) | x | | x | x | | x | | x |
| | | [50] | Sim. (ASIC) | | x | x | | | | | x |
| | | [65] | Sim. (ASIC) | | x | x | | x | | | x |
| | | [78] | FPGA | | x | x | | | x | | x |
| | **VAD** | [92] | Sim. (ASIC) | x | | x | | | x | | x |
| | | [89] | Sim. (ASIC) | x | | | | | x | | x |
| | **KWS** | [93] | Sim. (ASIC) | x | | x | x | | x | | x |
| | | [91] | Sim. (ASIC) | x | | | | | x | | x |
| | | [86] | Sim. (ASIC) | x | | | x | | x | | x |
| | | [94] | Sim. (ASIC) | x | | | x | | x | | |
| | | [87] | ASIC | x | | x | x | x | x | | |
| | **Speech** | [90] | Sim. (ASIC) | x | x | x | | | x | | x |
| | **recog.** | [69] | Sim. (ASIC) | | x | x | | | x | | x |
| | **Others** | [81] | Sim. (ASIC) | x | | x | | x | x | | |
| | | [85] | Sim. (ASIC) | x | x | x | | | | x | x |
| | | [75] | ASIC | x | | | | | | | x |
| | | [58] | Sim. (anal.) | | | | | | x | | x |
| | | [30] | Sim. (anal.) | x | | | x | | | | x |
| | | [179] | ASIC | x | x | x | | | x | | x |
| Image processing | **DCT** | [159] | Sim. (ASIC) | x | | x | | | x | | |
| | | [9] | Sim. (anal.) | | x | | x | | x | | x |
| | | [186] | Sim. (ASIC) | x | x | x | x | | x | | x |
| | | [180] | ASIC | | x | x | x | x | | | |
| | **Edge** | [37] | Sim. (anal.) | x | | x | | | x | | |
| | **detection** | [160] | Sim. (ASIC) | x | x | x | x | | x | | x |
| | | [167] | FPGA | x | x | x | | | x | | x |
| | | [108] | Sim. (ASIC) | x | | x | | | x | | |
| | **Others** | [158] | Sim. (ASIC) | x | | x | x | | x | | |
| | | [195] | FPGA | x | | x | | | x | | x |
| Video processing | **General** | [126] | FPGA | x | | | | | x | | x |
| | **HEVC** | [44] | Sim. (ASIC) | x | | x | x | | x | | |
| | | [147] | FPGA | x | x | x | x | | | | |
| | | [125] | Sim. (ASIC) | x | | x | x | | x | | |
| | | [133] | Sim. (ASIC) | x | | x | x | | x | | |
| | **Others** | [149] | Sim. (anal.) | | | | x | | | | |
| | | [135] | Sim. (anal.) | | | | | | | | x |
| Reliability | **NMR** | [28] | Sim. (ASIC) | x | | x | | | | | |
| | | [140] | FPGA | | | x | x | | | | x |
| | | [12] | Sim. (anal.) | | x | | | | | | |
| | | [39] | Sim. (anal.) | | | x | | | | | |
| | | [112] | Sim. (ASIC) | | | x | | | | | |
| | **Others** | [172] | Sim. (ASIC) | | x | | | | | | |
| | | [177] | Sim. (ASIC) | x | x | x | | | | | x |
| Others | **Wireless** | [152] | Sim. (anal.) | | | | | | | | |
| | **comm.** | [23] | FPGA | | | x | x | | | | x |
| | | [202] | FPGA | | | x | x | | x | | x |
| | | [60] | Unspec. | | | | | | x | | x |
| | | [184] | Sim. (anal.) | | | | | | x | | x |
| | | [63] | FPGA | x | | | | | x | | x |

Continued on the next page

| | | | | | | | | | | | | |
|---|---|---|---|---|---|---|---|---|---|---|---|---|
| *cont.* | | [102] | Sim. (ASIC) | | | x | x | | x | | x | |
| | | [19] | Sim. (ASIC) | | x | | | x | | | | |
| | | [88] | ASIC | x | x | | x | | x | | | |
| | | [40] | FPGA | | | x | x | | | | | |
| | **Machine** | [162] | ASIC | x | x | | | x | | | | |
| | **vision** | [55] | FPGA | x | | x | x | | | | | x |
| | **Others** | [68] | FPGA | x | | x | | | x | | | x |

and the relative importance of bit positions in transmitted packets to transmit important bits in better-protected positions. To support this, they present extended transmitter and receiver architectures. Hao *et al.* [60] explore the use of approximate adders in the Fast Fourier Transform (FFT) and Inverse FFT steps in a conventional wireless communication system.

Idrees *et al.* [63] consider a hypothetical 6G downlink. They propose using FIR filters with approximate fixed-point MACs to minimize the effects of channel noise over a wireless channel used for approximable data. They evaluate several combinations of multipliers and adders applied to different modulation schemes. Sen *et al.* [152] report improved output quality, Hao *et al.* [60] highlight the suitability of fail-rare approximations to their application, and Idrees *et al.* [63] demonstrate potential for significant power savings with little impact on quality.

The three other works focus on more particular technologies. Castaneda *et al.* [23] develop an approximate algorithm for data detection in Multiple-Input Multiple-Output systems. They apply *semidefinite relaxation* to reduce the complexity of the otherwise exponential-scale problem, quantize all intermediate values to fixed-point formats, and provide a PE-based accelerator targeting FPGAs. The accelerator achieves comparable throughput and improved error rates to related works. Xiao *et al.* [184] present three mathematical approximations of the *expectation propagation* step in Sparse Code Multiple Access detection: 1) Jacobi approximation, 2) fewer multiplications and divisions, and 3) max operations instead of some divisions and logarithms. Their algorithm nearly retains the performance of its exact counterpart yet permits simpler hardware implementation. Zhou *et al.* [202] consider the inherently serial successive cancellation decoding algorithm for polar codes an error-resilient DSP task. They propose using approximate comparators, invert-and-add-one modules, and adders, further truncating and quantizing variables to suitable fixed-point formats. The resulting architecture improves throughput at the expense of some error correction performance.

*4.5.2 Digital Signal Processing.* In line with [60, 63, 202], another four works focus on DSP-related applications. Martina *et al.* [102] propose a multiplier-less architecture for approximately computing the Discrete Wavelet Transform. They utilize inexact adders and describe a result-biasing technique for balancing out errors in different stages. Their designs achieve similar output quality as comparable designs. Basu *et al.* [19] focus on bio-DSP applications, describing an architecture that combines task parallelism, vectorization, and approximation. It implements several VOS processor cores and a SIMD-capable CGRA co-processor used only for exact acceleration. An execution monitoring system manages the control, data flow, and reconfiguration, ensuring resets upon non-recoverable memory errors. This approximate acceleration leads to significant energy reductions.

In parallel to their works on speech recognition [86, 87, 89–93], Liu *et al.* [88] present an approximate architecture for pre-processing audio into Mel-scale Frequency Cepstral Coefficients. They implement a reduced-complexity FFT, quantize all computations, utilize inexact adders and multipliers, and apply a dual-voltage supply scheme. The architecture is shown to drastically reduce the power consumption of a VAD system with negligible accuracy degradation. Unrelated to this, Dhaou [40] presents a hardware architecture for fuel consumption estimation in cars, approximating the engine's rotational velocity to save area for memories with negligible loss of accuracy.

*4.5.3 Machine vision.* Two works consider vision-related applications. Tagliavini *et al.* [162] present a NTV parallel architecture for motion detection, closely related to [163]. By dynamically adjusting the voltage of the data memory during different kernels, the authors clearly demonstrate the energy-quality trade-off that their

architecture enables. Gkeka *et al.* [55] consider compute-heavy algorithms used for unmanned aerial vehicles to navigate the world safely. Specifically, they target throughput improvements and propose an extensive list of exact and approximate code and hardware optimizations for accelerating localization and mapping kernels on FPGAs. Their resulting architectures achieve significant speedup over exact baselines at acceptable errors.

*4.5.4 Others.* The work of Jiang *et al.* [68] is not easily grouped with the above. Targeting *imprecise mixed-criticality* systems, they present an AxC-equipped processor architecture implementing an inexact floating-point unit. Supplementing this, they propose a medium-criticality mode for executing otherwise low-criticality applications approximately. With low software and hardware overheads, this mode enables executing more tasks within a given time quantum.

To recap, we have now reviewed fundamental AxC techniques, AxC-enabled hardware architectures, and application-specific designs profiting from AxC. This section clearly shows that ML is the most popular application domain. Yet, AxC has also been applied to image and video processing workloads, reliability and fault detection systems, wireless communication, DSP, and other Edge-related applications. Like before, we summarize frequently used performance metrics in Tab. 5 and reported results in terms of power and area in Fig. 8. Clearly, most works report power savings with only two exceptions instead reporting improved throughput [165] or fault tolerance [68]. The trend is similar for reported area: exceptions improve power consumption [81], energy efficiency [179, 180], or resource efficiency [23].

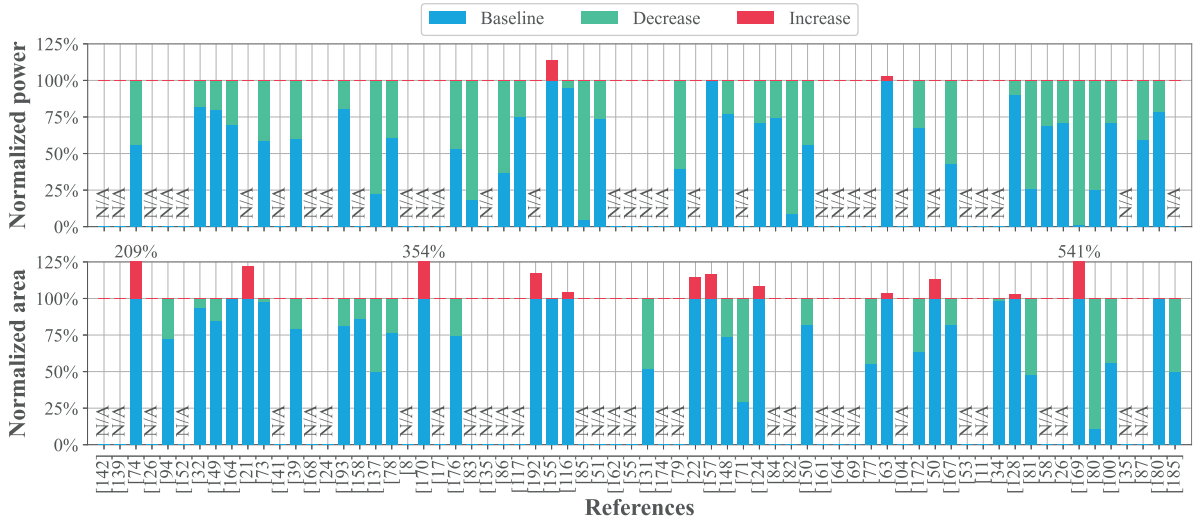


Fig. 8. Normalized power and area metrics from the works on applications of AxC, in chronological order.

## 5 CHALLENGES AND FUTURE WORK

While our technical review reveals a vast amount of work on AxC techniques and their applications, only two explicitly apply to Edge computing [19, 84]. This observation is shared by Barua and Mondal [18], who suggest that Edge computing may be better utilized by appropriately applying AxC techniques, indicating that open research directions remain. We cover the main part of these in the following three sections. Later, we present overarching research directions needed for the mainstream adoption of AxC.

### 5.1 Fundamental Approximate Computing Techniques

The first part of our review revealed a plethora of works on fundamental techniques useful in a wide range of architectures and applications. Despite the variation of works, there are still open research directions related to

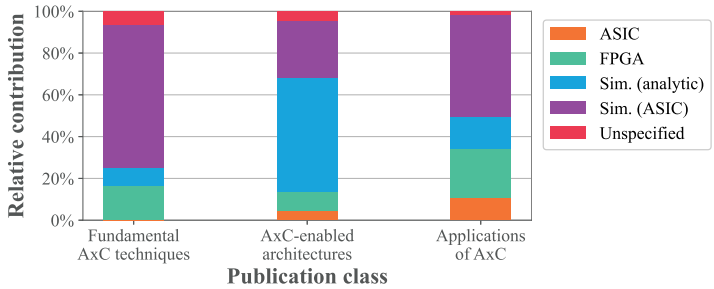


Fig. 9. Distribution of evaluation platforms used across the three classes of publications.

this field. For example, although VOS already appears to be the most effective circuit-level technique, enabling quadratic power reductions, Amanollahi *et al.* [10] highlight it as a direction of future research, especially in combination with dynamic threshold voltage adjustments. In parallel to this, researchers may explore hybrid or emerging technology-based memory architectures [163] or non-volatile memories as alternatives to volatile ones [10].

Although there exist many inexact arithmetic units [96, 105], related authors suggest further research into this area [124]. Some call for units designed with specific applications in mind, which assert either "*fail small*" or "*fail rare*" behavior [32]. The former is necessary for ANT [83], while the latter can be used for ML applications [32]. Others call for the design of reconfigurable units [80, 116, 200], potentially extended with low-overhead error compensation [199].

Architectures subject to strict area requirements have been shown to benefit from stochastic computing techniques. However, while stochastic arithmetic circuits are small, the logic required to generate (pseudo-)random bitstreams is often costly. Future research may consider approximating such circuits further by truncating them and the bitstreams they generate [153] or by using memristive devices in their place [5]. Researchers may also study how to tame memristive architectures' erroneous nature to effectively parallelize stochastic computations, including sequential divisions [5].

Lastly, our review has shown memoization as a low-complexity alternative to neural approximation [71]. Future research may focus on improving its use of local memories [71] and reducing its hardware overheads and lookup and update complexity [61, 157]. Moreover, concerning function approximation, stochastic computing implementations are particularly interesting because of their low area and power consumption. Yet, they require further research to be fully optimized and explored in relevant applications [98].

## 5.2 Approximate Computing-enabled Hardware Architectures

We have also shown how AxC is mostly used for application-specific architectures, likely due to its easier adaptation and potentially greater benefits. Yet, research into generic AxC-enabled architectures is also important, and many techniques are applicable to both [105]. Using AxC efficiently in GPPs is interesting as it offers diverse workloads and widespread application. Yet, its needs for ISA and tool flow changes limit its adoption [124, 187], and the lack of automatic sensitivity analysis tools limits its effectiveness [27]. Other research directions include the approximation of other instruction classes than arithmetic, load/store, and branches [113, 118]; approximation-aware cache replacement and insertion policies [104]; low-overhead memoization techniques [146]; and solutions to DRAM scaling challenges [201].

We find a need for further development of reconfigurable architectures that achieve a good balance of performance and low reconfiguration overhead. The current works focusing on CGRAs [2, 33, 41] fail to report reconfiguration overheads, making it unclear whether they have struck this balance. Furthermore, the potential benefits and implications of SIMD-style extensions to these architectures (like those of [19]) have yet to be fully

explored. Future works may also consider mixed-signal implementations (like [86, 90, 92]) to achieve even greater energy efficiency [124].

Regarding NoCs, research may focus on packet prediction and compression [139], value coding in wired NoCs [111], methods for approximate communication in wireless NoCs [49], as well as more optimal workload mapping and packet compression even for exact operation [107].

Unfortunately, related works mostly avoid hardware simulations opting for simpler, analytical alternatives, as shown in Fig. 9, they typically fail to report power and area estimates, which are often relevant in AxC-equipped systems.

### 5.3 Applications of Approximate Computing

In our review, we saw that AxC techniques have already been applied to a variety of applications. Although many of the reviewed works report improvements, they also outline many directions for future research. Note that the technical sections of our review have not covered many works related to security, as the included works are mainly surveys. Instead, we cover their suggestions here.

ML is undoubtedly the most popular application of AxC techniques. With the potential benefits arising from combining AxC and Edge acceleration and the increasing computational demands of larger models, we expect it to remain so for the near future. Our review covers many works on hardware accelerators for NNs with AxC techniques applied, e.g., [80, 85, 90, 91, 93, 171]. Yet, as the field is evolving rapidly, further research remains interesting. Such research may, for example, seek to understand the synergies between AxC and ML [171] and extend existing techniques to more NN layer types [24, 69]. Attaining the desired power savings with negligible impact on network accuracy requires fully understanding the benefits and drawbacks of AxC techniques applied to ML [80]. Understanding this also enables deterministically evaluating per-layer approximations [200] and supports the evaluation of dynamic AxC as a controlled source of noise, improving robustness against adversarial attacks [58].

In this direction, reviewing the training algorithms used is also relevant. For example, one work calls for application-specific ensemble learning methods [75], while another suggests developing general training algorithms with simplifications for improved execution time [171]. Concurrently, researchers may explore alternative ML model types. The review reveals SNNs as the most likely successor to current DNNs. Initial results are promising, but the networks have yet to see mainstream use and be applied to more complex learning tasks than image recognition [151, 174]. HDC and SVMs also appear as intriguing alternatives allowing for efficient acceleration. Both SNNs, HDC, and SVMs offer many open research directions, particularly in emerging technology-based hardware architectures [25, 54, 65, 141].

Multimedia applications are also often approximated, highlighted by the number of reviewed works in Secs. 4.2 and 4.3, the frequently used benchmark applications in Tabs. 2 and 4, and by Palem and Lingamneni [124]. Most of these works target a specific step in an algorithm (e.g., DCT for JPEG or motion estimation for HEVC). However, some suggest extending their approaches to other steps and evaluating them more carefully in different environments [37, 125, 167].

The reviewed works on reliability and fault tolerance offer many opportunities for improvement. Reducing the overhead of traditional NMR schemes while maintaining a satisfactory level of fault detection is both intriguing and challenging. We find that while some existing works present promising results, there is still a need for further research into, for example, the potential of further approximations in complex DMR, TMR, and QMR systems [12, 39, 112], the benefits arising from truncating module outputs [140], and how to model errors from skipping crashes in non-critical code regions [172]. Moreover, it is perhaps even more interesting to delve into the intersection of these techniques.

Hardware security is becoming increasingly important as more personal data is being stored and processed at a distance from the users. While Edge processing reduces the number of (shared) storage and processing locations, it does not guarantee complete security [119]. Therefore, Edge devices must, apart from being reliable, also contain security features that ensure no attackers can detect and propagate secret information about their users. The reviewed works outline many open challenges in this direction, of which some recur in multiple works. These topics include AxC for generating *physically unclonable functions* [95, 96] usable for authentication and identification; insertion, activation, and detection of hardware trojans [95, 96, 196]; and distinguishing errors caused by AxC from those caused by trojans [95]. It is also relevant to analyze using AxC against fault injection and side-channel attacks [96, 138]. Regrettably, although AxC can obfuscate logic circuits and hide their attack surfaces, it may also bring new attack surfaces whose detection and mitigation is crucial, especially in the context of compound attacks [176]. Applying AxC to post-quantum cryptography algorithms is another interesting research direction yet to be explored [138].

## 5.4 Overarching Research Topics

As we have seen, there are numerous possibilities for further research into the three main technical directions presented in our review. However, some research directions, such as quality management and cross-layer AxC techniques, span a multitude of them. A common notion in related works is that both require more work to enable wider adoption of AxC. We present some ideas for such work here.

Most of the reviewed works on AxC-enabled architectures and applications of AxC statically apply approximations with little-to-none regard for their effects on output quality. This approach is unlikely to maximize potential benefits from AxC, as it implies using techniques conservatively. Instead, quality management systems are required to control approximations effectively [187], existing techniques differing in whether they are applied offline or at run-time.

In the first category, Chen *et al.* [31] propose using developer annotations and static code analysis to determine variables' error sensitivity to truncation, applying their technique to the NoC of [139]. More works fall into the second category, using either LWCs to make circuits *self-tunable* [188] (like [103]) or various regression or NN models to predict errors and adapt approximations accordingly [67, 77, 79, 175]. Unfortunately, most of these techniques are either too conservative [31] or fail to ensure quality constraints [67, 79, 175, 188]. One work stands out, giving thresholds as mean relative error values that are almost fully utilized yet guaranteed [77]. Moreover, although some report promising results, they are often followed by high hardware/software overheads that limit the savings arising from AxC.

Preferably, an application should need minimal overheads to communicate its quality constraints to the underlying hardware, which in turn should monitor and adapt approximations accordingly to guarantee output quality. Research into low-overhead quality monitoring and management at the circuit and architecture levels is needed [14]. Techniques may likely utilize LWCs and NNs as in the reviewed works [79, 188], perhaps in combination with iterative classification and minimal-error SOP approximations [67, 77]. Monitors could also implement traditional fault detection hardware configured to flag only faults known to cause unsatisfactory output quality [14].

Orthogonally, some authors call for tools to formally verify that approximate systems satisfy their quality constraints [169, 187]. Designers of such can benefit from traditional verification tools extended to support quality-oriented properties or approximate equivalence checks [124, 169]. Finally, guaranteeing quality in non-deterministic approximate chips requires developments in in-circuit test flows to ignore acceptable approximation errors [95].

Taking full advantage of AxC requires further development of cross-layer techniques, tools, and frameworks. Mature tools can perform design space exploration and hardware/software co-design, combining various techniques

to satisfy user-specified quality constraints while maximizing energy savings or minimizing latency [114, 124, 200]. Such exploration, however, implies a need for further development of techniques and analysis of their effects. Although some works already combine techniques, e.g., [145, 171], they mostly use truncation with another of the reviewed fundamental techniques, as shown in Tabs. 1, 3, and 5. The tools must also be able to reason about non-deterministic behavior in approximate hardware, for example, through special software libraries or new programming languages [187].

Once well-developed, designers can use these tools for optimizations, a process quickly necessitating computer-aided tools for complex designs. Historically, this has been solved by raising the abstraction level, most recently to HLS. Several works follow this idea and propose extensions to such tools [84, 112, 114, 157, 188]. Alternative tools could be designed specifically to target circuits implemented using probabilistic Boolean logic [124] or memoization [71, 157]. In general, approximate logic synthesis, as surveyed by Scarabottolo *et al.* [148], is very promising but requires great efforts to overcome its current drawbacks in terms of scaling, support for run-time reconfigurability, cross-layer technique exploitation, and dependence on synthesis and simulation [114]. In this space, ML-based design space exploration has recently gained traction. Especially evolutionary algorithms seem well-suited for optimization problems with very large search spaces due to their randomized generational evolution [173]. Such algorithms can efficiently identify near-optimal design parameter configurations for larger designs, or they can be used to optimize smaller components [150]. However, they still need significant research efforts to become functional in practice.

We generally find that techniques should be easily applicable and cause minimal strain on developers, which implies updates to several steps of the development process from compilers, verification, simulation, synthesis tools, and the languages they operate on to tools for sensitivity analysis and formal reasoning. Furthermore, although results arising from AxC applied to various workloads are promising, the combined benefits of AxC and Edge computing are yet to be explored. Lastly, we note that this list is not comprehensive but comprises the ideas we find necessary for a broad adoption of AxC within the Edge computing scope.

## 6 REVIEW SUMMARY

As the number of connected devices that generate data increases, so do the computational demands to aggregate the data. However, many of the devices are battery-driven and, thus, need to perform these computations at minimal energy consumption. Traditionally, they have completely offloaded their computations to far-away centralized Cloud data centers at great costs in terms of communication energy and latency, but the expected future data amounts render this solution unfit. Moreover, this method fails to exploit the inherent error resilience of the applications. This calls for alternatives to be explored. One such alternative is the intriguing combination of the AxC and Edge computing domains. Doing so allows for exploiting the benefits of approximation and offloading at once. This also opens new optimization opportunities by extending the traditional time-energy trade-off by an accuracy aspect.

In this paper, we have presented a systematic literature review of publications about AxC and Edge computing. As such, we have considered works in three classes: fundamental AxC techniques, AxC-enabled architectures, and applications of AxC. From this, we observed that many different techniques, from the circuit level to the architecture level, have found application both in GPPs, CGRAs, and NoCs. However, we also noticed a lack of work exploring cross-layer techniques. We expect that combining them will lead to greater savings but that it also requires further developments in low-overhead quality management.

Our investigation into applications revealed several trends. First, AxC has primarily been utilized for accelerating ML and multimedia workloads, which exhibit the error resilience expected of them. However, we noticed the surprising application of AxC to reduce overheads in fault tolerant systems; a somewhat unexpected though interesting direction. Secondly, despite many works presenting application-specific accelerators, very few apply

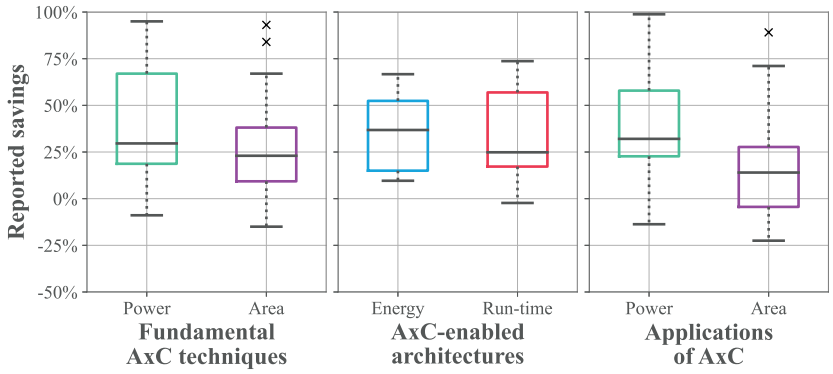


Fig. 10. Box-plots of results in the two most frequently reported performance metrics of each publication class. The savings axis is limited to [−50%, 100%] for clarity, excluding two fundamental AxC technique area outliers (from [142, 163]) and three applications of AxC area outliers (from [81, 179, 180]). Negative values mean increased metrics. Boxes are colored, indicating recurring metrics.

more than one or two AxC techniques simultaneously, once again implying a need for research into cross-layer techniques. And lastly, most publications report savings in power, energy, area, or delay/run-time, while some trade off one for the other. The distribution of reported results, previously presented in Figs. 4, 6, and 8, is shown in Fig. 10, clearly demonstrating the benefits one can expect from using the reviewed techniques.

Despite the amount of work that already exists in the domains of AxC and Edge computing, only relatively few explicitly consider their intersection. We believe that there is still room for significant improvements in these fields. We have outlined ideas for further research to motivate researchers to advance them towards mainstream adoption.

## REFERENCES

[1] Sergi Abadal, Albert Cabellos-Aparicio, Eduard Alarcon, and Josep Torrellas. 2016. WiSync: An Architecture for Fast Synchronization through On-Chip Wireless Communication. *ACM SIGARCH Computer Architecture News* 44, 2 (2016), 3–17.

[2] Omid Akbari, Mehdi Kamal, Ali Afzali-Kusha, Massoud Pedram, and Muhammad Shafique. 2018. Toward Approximate Computing for Coarse-Grained Reconfigurable Architectures. *IEEE Micro* 38, 6 (2018), 63–72.

[3] Omid Akbari, Mehdi Kamal, Ali Afzali-Kusha, Massoud Pedram, and Muhammad Shafique. 2019. X-CGRA: An Energy-Efficient Approximate Coarse-Grained Reconfigurable Architecture. *IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems* 39, 10 (2019), 2558–2571.

[4] Filipp Akopyan, Jun Sawada, Andrew Cassidy, Rodrigo Alvarez-Icaza, John Arthur, Paul Merolla, Nabil Imam, Yutaka Nakamura, Pallab Datta, Gi-Joon Nam, et al. 2015. Truenorth: Design and Tool Flow of a 65 mW 1 Million Neuron Programmable Neurosynaptic Chip. *IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems* 34, 10 (2015), 1537–1557.

[5] Mohsen Riahi Alam, Mohammadreza Hassan Najafi, Nima Taherinejad, Mohsen Imani, and Raju Gottumukkala. 2022. Stochastic Computing in Beyond Von-Neumann Era: Processing Bit-Streams in Memristive Memory. *IEEE Transactions on Circuits and Systems II: Express Briefs* 69, 5 (2022), 2423–2427.

[6] Daria Alekseeva, Aleksandr Ometov, Otso Arponen, and Elena Simona Lohan. 2022. The Future of Computing Paradigms for Medical and Emergency Applications. *Computer Science Review* 45 (2022), 100494.

[7] Christopher Allen, Derrick Langley, and James Lyke. 2014. Inexact Computing with Approximate Adder Application. In *Proc. of National Aerospace and Electronics Conference (NAECON)*. IEEE, 21–28.

[8] Mario Almeida, Stefanos Laskaridis, Stylianos Venieris, Ilias Leontiadis, and Nicholas Lane. 2022. Dyno: Dynamic Onloading of Deep Neural Networks from Cloud to Device. *ACM Transactions on Embedded Computing Systems (TECS)* 21, 6 (2022), 1–24.

[9] Haider Almurib, Thulasiraman Nandha Kumar, and Fabrizio Lombardi. 2017. Approximate DCT Image Compression Using Inexact Computing. *IEEE Trans. Comput.* 67, 2 (2017), 149–159.

[10] Saba Amanollahi, Mehdi Kamal, Ali Afzali-Kusha, and Massoud Pedram. 2020. Circuit-Level Techniques for Logic and Memory Blocks in Approximate Computing Systemsx. *Proc. IEEE* 108, 12 (2020), 2150–2177.

[11] Rida Amjad, Rehan Hafiz, Muhammad Usman Ilyas, Muhammad Shahzad Younis, and Muhammad Shafique. 2019. m-SAAC: Multi-Stage Adaptive Approximation Control to Select Approximate Computing Modes for Vision Applications. *Microelectronics Journal* 91 (2019), 84–91.

[12] Joseph Henry Anajemba, James Adu Ansere, Frederick Sam, Celestine Iwendi, and Gautam Srivastava. 2021. Optimal Soft Error Mitigation in Wireless Communication Using Approximate Logic Circuits. *Sustainable Computing: Informatics and Systems* 30 (2021), 100521.

[13] Cisco and/or its affiliates. 2020. *Cisco Annual Internet Report (2018–2023).* Technical Report C11-741490-01. Cisco. https://www.cisco.com/c/en/us/solutions/collateral/executive-perspectives/annual-internet-report/white-paper-c11-741490.html

[14] Lorena Anghel, Mounir Benabdenbi, Alberto Bosio, Marcello Traiola, and Elena Ioana Vatajelu. 2018. Test and Reliability in Approximate Computing. *Journal of Electronic Testing* 34, 4 (2018), 375–387.

[15] Giuseppe Ascia, Vincenzo Catania, Salvatore Monteleone, Maurizio Palesi, Davide Patti, and John Jose. 2018. Approximate Wireless Networks-on-Chip. In *Proc. of Conference on Design of Circuits and Integrated Systems (DCIS).* IEEE, 1–6.

[16] Giuseppe Ascia, Vincenzo Catania, Salvatore Monteleone, Maurizio Palesi, Davide Patti, John Jose, and Valerio Mario Salerno. 2020. Exploiting Data Resilience in Wireless Network-on-Chip Architectures. *ACM Journal on Emerging Technologies in Computing Systems (JETC)* 16, 2 (2020), 1–27.

[17] Hiroyuki Baba, Tongxin Yang, Masahiro Inoue, Kaori Tajima, Tomoaki Ukezono, and Toshinori Sato. 2018. A Low-Power and Small-Area Multiplier for Accuracy-Scalable Approximate Computing. In *Proc. of Computer Society Annual Symposium on VLSI (ISVLSI).* IEEE, 569–574.

[18] Hrishav Bakul Barua and Kartick Chandra Mondal. 2019. Approximate Computing: A Survey of Recent Trends—Bringing Greenness to Computing and Communication. *Journal of The Institution of Engineers (India): Series B* 100, 6 (2019), 619–626.

[19] Soumya Basu, Loris Duch, Miguel Peón-Quirós, David Atienza, Giovanni Ansaloni, and Laura Pozzi. 2018. Heterogeneous and Inexact: Maximizing Power Efficiency of Edge Computing Sensors for Health Monitoring Applications. In *Proc. of International Symposium on Circuits and Systems (ISCAS).* IEEE, 1–5.

[20] Kunal Bharathi, Jiang Hu, and Sunil Khatri. 2020. Scaled Population Subtraction for Approximate Computing. In *Proc. of 38th International Conference on Computer Design (ICCD).* IEEE, 348–355.

[21] Christian Bienia, Sanjeev Kumar, Jaswinder Pal Singh, and Kai Li. 2008. The PARSEC Benchmark Suite: Characterization and Architectural Implications. In *Proc. of 17th International Conference on Parallel Architectures and Compilation Techniques (PACT).* ACM, 72–81.

[22] Flavio Bonomi, Rodolfo Milito, Jiang Zhu, and Sateesh Addepalli. 2012. Fog Computing and Its Role in the Internet of Things. In *Proc. of 1st Edition of the MCC Workshop on Mobile Cloud Computing.* ACM, 13–16.

[23] Oscar Castaneda, Tom Goldstein, and Christoph Studer. 2016. FPGA Design of Approximate Semidefinite Relaxation for Data Detection in Large MIMO Wireless Systems. In *Proc. of International Symposium on Circuits and Systems (ISCAS).* IEEE, 2659–2662.

[24] Jorge Castro-Godínez, Deykel Hernández-Araya, Muhammad Shafique, and Jörg Henkel. 2020. Approximate Acceleration for CNN-based Applications on IoT Edge Devices. In *Proc. of 11th Latin American Symposium on Circuits & Systems (LASCAS).* IEEE, 1–4.

[25] Jair Cervantes, Farid Garcia-Lamont, Lisbeth Rodríguez-Mazahua, and Asdrubal Lopez. 2020. A Comprehensive Survey on Support Vector Machine Classification: Applications, Challenges and Trends. *Neurocomputing* 408 (2020), 189–215.

[26] Patrik Cerwal et al. 2021. *Ericsson Mobility Report.* Technical Report EAB-21:010887. Ericsson. https://www.ericsson.com/en/reports-and-papers/mobility-report/reports/november-2021

[27] Arun Chandrasekharan, Daniel Große, and Rolf Drechsler. 2017. ProACt: A Processor for High Performance On-Demand Approximate Computing. In *Proc. of Great Lakes Symposium on VLSI (GLSVLSI).* ACM, 463–466.

[28] Ke Chen, Jie Han, and Fabrizio Lombardi. 2017. Two Approximate Voting Schemes for Reliable Computing. *IEEE Trans. Comput.* 66, 7 (2017), 1227–1239.

[29] Linbin Chen, Jie Han, Weiqiang Liu, and Fabrizio Lombardi. 2015. Design of Approximate Unsigned Integer Non-Restoring Divider for Inexact Computing. In *Proc. of Great Lakes Symposium on VLSI (GLSVLSI).* ACM, 51–56.

[30] Yuechen Chen, Shanshan Liu, Lombardi Fabrizio, and Ahmed Louri. 2022. A Technique for Approximate Communication in Network-on-Chips for Image Classification. *IEEE Transactions on Emerging Topics in Computing* (2022). Early access.

[31] Yuechen Chen and Ahmed Louri. 2020. Learning-based Quality Management for Approximate Communication in Network-on-Chips. *IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems* 39, 11 (2020), 3724–3735.

[32] Vinay Kumar Chippa, Srimat Tirumala Chakradhar, Kaushik Roy, and Anand Raghunathan. 2013. Analysis and Characterization of Inherent Application Resilience for Approximate Computing. In *Proc. of 50th Design Automation Conference (DAC).* ACM, 1–9.

[33] Vinay Kumar Chippa, Swagath Venkataramani, Srimat Tirumala Chakradhar, Kaushik Roy, and Anand Raghunathan. 2013. Approximate Computing: An Integrated Hardware Approach. In *Proc. of Asilomar Conference on Signals, Systems and Computers.* IEEE, 111–117.

[34] Corinna Cortes and Vladimir Vapnik. 1995. Support-Vector Networks. *Machine Learning* 20, 3 (1995), 273–297.

[35] Ayad Dalloo. 2018. Enhance the Segmentation Principle in Approximate Computing. In *Proc. of International Conference on Circuits and Systems in Digital Enterprise Technology (ICCSDET).* IEEE, 1–7.

[36] Mike Davies, Narayan Srinivasa, Tsung-Han Lin, Gautham Chinya, Yongqiang Cao, Sri Harsha Choday, Georgios Dimou, Prasad Joshi, Nabil Imam, Shweta Jain, et al. 2018. Loihi: A Neuromorphic Manycore Processor with On-Chip Learning. *IEEE Micro* 38, 1 (2018), 82–99.

[37] Julio de Oliveira, Leonardo Soares, Eduardo Costa, and Sergio Bampi. 2016. Exploiting Approximate Adder Circuits for Power-Efficient Gaussian and Gradient Filters for Canny Edge Detector Algorithm. In *Proc. of 7th Latin American Symposium on Circuits & Systems (LASCAS)*. IEEE, 379–382.

[38] Ines del Campo, Javier Echanobe, Estibaliz Asua, and Raul Finker. 2015. Controlled-Accuracy Approximation of Nonlinear Functions for Soft Computing Applications: A High Performance Co-Proccessor for Intelligent Embedded Systems. In *Proc. of Symposium Series on Computational Intelligence (SSCI)*. IEEE, 609–616.

[39] Bastien Deveautour, Marcello Traiola, Arnaud Virazel, and Patrick Girard. 2021. Reducing Overprovision of Triple Modular Reduncancy Owing to Approximate Computing. In *Proc. of 27th International Symposium on On-Line Testing and Robust System Design (IOLTS)*. IEEE, 1–7.

[40] Imed Ben Dhaou. 2022. Implementation of a Fuel Estimation Algorithm Using Approximated Computing. *Journal of Low Power Electronics and Applications* 12, 1 (2022), 17.

[41] Jonathan Dickerson, Ioannis Galanis, Zois-Gerasimos Tasoulas, Lincoln Kinley, and Iraklis Anagnostopoulos. 2020. Adaptive Approximate Computing on Hardware Accelerators Targeting Internet-of-Things. In *Proc. of 6th World Forum on Internet of Things (WF-IoT)*. IEEE, 1–6.

[42] Sunil Dutt, Sukumar Nandi, and Gaurav Trivedi. 2017. Analysis and Design of Adders for Approximate Computing. *ACM Transactions on Embedded Computing Systems (TECS)* 17, 2 (2017), 1–28.

[43] Jorge Echavarria, Katja Schütz, Andreas Becher, Stefan Wildermann, and Jürgen Teich. 2018. Can Approximate Computing Reduce Power Consumption on FPGAs?. In *Proc. of 25th International Conference on Electronics, Circuits and Systems (ICECS)*. IEEE, 841–844.

[44] Walaa El-Harouni, Semeen Rehman, Bharath Srinivas Prabakaran, Akash Kumar, Rehan Hafiz, and Muhammad Shafique. 2017. Embracing Approximate Computing for Energy-Efficient Motion Estimation in High Efficiency Video Coding. In *Proc. of Design, Automation & Test in Europe Conference & Exhibition (DATE)*. IEEE, 1384–1389.

[45] Elsevier. 2022. Scopus. https://www.scopus.com/home.uri.

[46] Hadi Esmaeilzadeh, Emily Blem, Renée St. Amant, Karthikeyan Sankaralingam, and Doug Burger. 2011. Dark Silicon and the End of Multicore Scaling. In *Proc. of 38th International Symposium on Computer Architecture (ISCA)*. ACM, 365–376.

[47] Hadi Esmaeilzadeh, Adrian Sampson, Luis Ceze, and Doug Burger. 2012. Neural Acceleration for General-Purpose Approximate Programs. In *Proc. of 45th International Symposium on Microarchitecture (MICRO)*. IEEE, 449–460.

[48] Sayed Rasoul Faraji, Pierre Abillama, and Kia Bazargan. 2021. Approximate Constant-Coefficient Multiplication Using Hybrid Binary-Unary Computing for FPGAs. *ACM Transactions on Reconfigurable Technology and Systems (TRETS)* 15, 3 (2021), 1–25.

[49] Vimuth Fernando, Antonio Franques, Sergi Abadal, Sasa Misailovic, and Josep Torrellas. 2019. Replica: A Wireless Manycore for Communication-Intensive and Approximate Data. In *Proc. of 24th International Conference on Architectural Support for Programming Languages and Operating Systems (ASPLOS)*. ACM, 849–863.

[50] Lorenzo Ferretti, Giovanni Ansaloni, Laura Pozzi, Amir Aminifar, David Atienza, Leila Cammoun, and Philippe Ryvlin. 2019. Tailoring SVM Inference for Resource-Efficient ECG-based Epilepsy Monitors. In *Proc. of Design, Automation & Test in Europe Conference & Exhibition (DATE)*. IEEE, 948–951.

[51] Farnaz Forooghifar, Amir Aminifar, and David Atienza. 2019. Resource-aware Distributed Epilepsy Monitoring Using Self-Awareness from Edge to Cloud. *IEEE Transactions on Biomedical Circuits and Systems* 13, 6 (2019), 1338–1350.

[52] Davide Gadioli, Emanuele Vitali, Gianluca Palermo, and Cristina Silvano. 2018. Margot: A Dynamic Autotuning Framework for Self-Aware Approximate Computing. *IEEE Trans. Comput.* 68, 5 (2018), 713–728.

[53] Shrikanth Ganapathy, Adam Teman, Robert Giterman, Andreas Burg, and Georgios Karakonstantis. 2015. Approximate Computing with Unreliable Dynamic Memories. In *Proc. of 13th International New Circuits and Systems Conference (NEWCAS)*. IEEE, 1–4.

[54] Lulu Ge and Keshab Parhi. 2020. Classification Using Hyperdimensional Computing: A Review. *IEEE Circuits and Systems Magazine* 20, 2 (2020), 30–47.

[55] Maria Rafaela Gkeka, Alexandros Patras, Christos Antonopoulos, Spyros Lalis, and Nikolaos Bellas. 2021. FPGA Architectures for Approximate Dense SLAM Computing. In *Proc. of Design, Automation & Test in Europe Conference & Exhibition (DATE)*. IEEE, 828–833.

[56] Yu Gong, Bo Liu, Wei Ge, and Longxing Shi. 2019. ARA: Cross-Layer Approximate Computing Framework based Reconfigurable Architecture for CNNs. *Microelectronics Journal* 87 (2019), 33–44.

[57] Beayna Grigorian, Nazanin Farahpour, and Glenn Reinman. 2015. BRAINIAC: Bringing Reliable Accuracy into Neurally-Implemented Approximate Computing. In *Proc. of 21st International Symposium on High Performance Computer Architecture (HPCA)*. IEEE, 615–626.

[58] Amira Guesmi, Ihsen Alouani, Khaled Khasawneh, Mouna Baklouti, Tarek Frikha, Mohamed Abid, and Nael Abu-Ghazaleh. 2021. Defensive Approximation: Securing CNNs Using Approximate Computing. In *Proc. of 26th International Conference on Architectural Support for Programming Languages and Operating Systems (ASPLOS)*. ACM, 990–1003.

[59] Jie Han and Michael Orshansky. 2013. Approximate Computing: An Emerging Paradigm for Energy-Efficient Design. In *Proc. of 18th European Test Symposium (ETS)*. IEEE, 1–6.
[60] Mingjie Hao, Ardalan Najafi, Alberto García-Ortiz, Ludwig Karsthof, Steffen Paul, and Jochen Rust. 2019. Reliability of an Industrial Wireless Communication System using Approximate Units. In *Proc. of 29th International Symposium on Power and Timing Modeling, Optimization and Simulation (PATMOS)*. IEEE, 87–90.
[61] Xin He, Shuhao Jiang, Wenyan Lu, Guihai Yan, Yinhe Han, and Xiaowei Li. 2016. Exploiting the Potential of Computation Reuse through Approximate Computing. *IEEE Transactions on Multi-Scale Computing Systems* 3, 3 (2016), 152–165.
[62] Yajuan He, Xilin Yi, Ziji Zhang, Bin Ma, and Qiang Li. 2020. A Probabilistic Prediction-based Fixed-Width Booth Multiplier for Approximate Computing. *IEEE Transactions on Circuits and Systems I: Regular Papers* 67, 12 (2020), 4794–4803.
[63] Maryam Idrees, Mohammed Manzar Maqbool, Muhammad Khurram Bhatti, Muhammad Mahboob Ur Rahman, Rehan Hafiz, and Muhammad Shafique. 2021. An Approximate-Computing Empowered Green 6G Downlink. *Physical Communication* 49 (2021), 101444.
[64] Mohsen Imani, Ricardo Garcia, Saransh Gupta, and Tajana Rosing. 2018. RMAC: Runtime Configurable Floating Point Multiplier for Approximate Computing. In *Proc. of International Symposium on Low Power Electronics and Design (ISLPED)*. ACM, 1–6.
[65] Mohsen Imani, Abbas Rahimi, Deqian Kong, Tajana Rosing, and Jan Rabaey. 2017. Exploring Hyperdimensional Associative Memory. In *Proc. of 23rd International Symposium on High Performance Computer Architecture (HPCA)*. IEEE, 445–456.
[66] Chandan Jha and Joycee Mekie. 2019. Design of Novel CMOS based Inexact Subtractors and Dividers for Approximate Computing: An in-Depth Comparison with PTL based Designs. In *Proc. of 22nd Euromicro Conference on Digital System Design (DSD)*. IEEE, 174–181.
[67] Shuhao Jiang, Jiajun Li, Xin He, Guihai Yan, Xuan Zhang, and Xiaowei Li. 2018. RiskCap: Minimizing Effort of Error Regulation for Approximate Computing. In *Proc. of 27th Asian Test Symposium (ATS)*. IEEE, 133–138.
[68] Zhe Jiang, Xiaotian Dai, and Neil Audsley. 2021. HIART-MCS: High Resilience and Approximated Computing Architecture for Imprecise Mixed-Criticality Systems. In *Proc. of 42nd Real-Time Systems Symposium (RTSS)*. IEEE, 290–303.
[69] Junseo Jo, Jaeha Kung, and Youngjoo Lee. 2020. Approximate LSTM Computing for Energy-Efficient Speech Recognition. *Electronics* 9, 12 (2020), 2004.
[70] Hounghun Joe and Youngmin Kim. 2019. Efficient Approximate Image Processor with Low-Part Stochastic Computing. In *Proc. of Asia Pacific Conference on Postgraduate Research in Microelectronics and Electronics (PrimeAsia)*. IEEE, 29–32.
[71] Michael Jordan, Marcelo Brandalero, Guilherme Malfatti, Geraldo Oliveira, Arthur Lorenzon, Bruno da Silva, Luigi Carro, Mateus Rutzig, and Antonio Carlos Beck. 2020. Data Clustering for Efficient Approximate Computing. *Design Automation for Embedded Systems* 24, 1 (2020), 3–22.
[72] Yirong Kan, Man Wu, Renyuan Zhang, and Yasuhiko Nakashima. 2020. A Multi-grained Reconfigurable Accelerator for Approximate Computing. In *Proc. of Computer Society Annual Symposium on VLSI (ISVLSI)*. IEEE, 90–95.
[73] Pentti Kanerva. 1992. *Sparse Distributed Memory and Related Models.* Technical Report. NASA.
[74] Mingu Kang, Sujan Gonugondla, and Naresh Shanbhag. 2020. Deep in-Memory Architectures in SRAM: An Analog Approach to Approximate Computing. *Proc. IEEE* 108, 12 (2020), 2251–2275.
[75] Bapi Kar, Pradeep Kumar Gopalakrishnan, Sumon Kumar Bose, Mohendra Roy, and Arindam Basu. 2020. ADIC: Anomaly Detection Integrated Circuit in 65-nm CMOS Utilizing Approximate Computing. *IEEE Transactions on Very Large Scale Integration (VLSI) Systems* 28, 12 (2020), 2518–2529.
[76] Geethan Karunaratne, Manuel Le Gallo, Giovanni Cherubini, Luca Benini, Abbas Rahimi, and Abu Sebastian. 2020. In-Memory Hyperdimensional Computing. *Nature Electronics* 3, 6 (2020), 327–337.
[77] Taylor Kemp, Yao Yao, and Younghyun Kim. 2021. MIPAC: Dynamic Input-Aware Accuracy Control for Dynamic Auto-Tuning of Iterative Approximate Computing. In *Proc. of 26th Asia and South Pacific Design Automation Conference (ASP-DAC)*. IEEE, 248–253.
[78] Behnam Khaleghi, Sahand Salamat, Anthony Thomas, Fatemeh Asgarinejad, Yeseong Kim, and Tajana Rosing. 2020. SHEAR er: Highly-Efficient Hyperdimensional Computing by Software-Hardware Enabled Multifold Approximation. In *Proc. of International Symposium on Low Power Electronics and Design (ISLPED)*. ACM, 241–246.
[79] Daya Shanker Khudia, Babak Zamirai, Mehrzad Samadi, and Scott Mahlke. 2015. RUMBA: An Online Quality Management System for Approximate Computing. In *Proc. of 42nd International Symposium on Computer Architecture (ISCA)*. ACM, 554–566.
[80] Duckhwan Kim, Jaeha Kung, and Saibal Mukhopadhyay. 2017. A Power-Aware Digital Multilayer Perceptron Accelerator with on-Chip Training based on Approximate Computing. *IEEE Transactions on Emerging Topics in Computing* 5, 2 (2017), 164–178.
[81] Eric Kim and Naresh Shanbhag. 2014. Energy-Efficient Accelerator Architecture for Stereo Image Matching Using Approximate Computing and Statistical Error Compensation. In *Proc. of Global Conference on Signal and Information Processing (GlobalSIP)*. IEEE, 55–59.
[82] Younghoon Kim, Swagath Venkataramani, Sanchari Sen, and Anand Raghunathan. 2021. Value Similarity Extensions for Approximate Computing in General-Purpose Processors. In *Proc. of Design, Automation & Test in Europe Conference & Exhibition (DATE)*. IEEE, 481–486.
[83] Yongtae Kim, Yong Zhang, and Peng Li. 2014. Energy Efficient Approximate Arithmetic for Error Resilient Neuromorphic Computing. *IEEE Transactions on Very Large Scale Integration (VLSI) Systems* 23, 11 (2014), 2733–2737.

[84] Lucas Klemmer, Saman Froehlich, Rolf Drechsler, and Daniel Große. 2021. XbNN: Enabling CNNs on Edge Devices by Approximate On-Chip Dot Product Encoding. In *Proc. of International Symposium on Circuits and Systems (ISCAS)*. IEEE, 1–5.

[85] Bingzhe Li, Yaobin Qin, Bo Yuan, and David Lilja. 2017. Neural Network Classifiers Using Stochastic Computing with a Hardware-Oriented Approximate Activation Function. In *Proc. of 35th International Conference on Computer Design (ICCD)*. IEEE, 97–104.

[86] Bo Liu, Hao Cai, Yu Gong, Wentao Zhu, Yan Li, Wei Ge, and Zhen Wang. 2020. Binarized Weight Neural-Network Inspired Ultra-Low Power Speech Recognition Processor with Time-Domain Based Digital-Analog Mixed Approximate Computing. In *Proc. of International Symposium on Circuits and Systems (ISCAS)*. IEEE, 1–5.

[87] Bo Liu, Hao Cai, Xuan Zhang, Haige Wu, Anfeng Xue, Zilong Zhang, Zhen Wang, and Jun Yang. 2022. A Target-Separable BWN Inspired Speech Recognition Processor with Low-Power Precision-Adaptive Approximate Computing. In *Proc. of Design, Automation & Test in Europe Conference & Exhibition (DATE)*. IEEE, 196–201.

[88] Bo Liu, Xiaoling Ding, Hao Cai, Wentao Zhu, Zhen Wang, Weiqiang Liu, and Jun Yang. 2021. Precision Adaptive MFCC based on R2SDF-FFT and Approximate Computing for Low-Power Speech Keywords Recognition. *IEEE Circuits and Systems Magazine* 21, 4 (2021), 24–39.

[89] Bo Liu, Yan Li, Lepeng Huang, Hao Cai, Wentao Zhu, Shisheng Guo, Yu Gong, and Zhen Wang. 2020. A Background Noise Self-adaptive VAD Using SNR Prediction Based Precision Dynamic Reconfigurable Approximate Computing. In *Proc. of Great Lakes Symposium on VLSI (GLSVLSI)*. ACM, 271–275.

[90] Bo Liu, Hai Qin, Yu Gong, Wei Ge, Mengwen Xia, and Longxing Shi. 2018. EERA-ASR: An Energy-Efficient Reconfigurable Architecture for Automatic Speech Recognition with Hybrid DNN and Approximate Computing. *IEEE Access* 6 (2018), 52227–52237.

[91] Bo Liu, Yuhao Sun, Hao Cai, Zeyu Shen, Yu Gong, Lepeng Huang, and Zhen Wang. 2020. An Ultra-low Power Keyword-Spotting Accelerator Using Circuit-Architecture-System Co-design and Self-adaptive Approximate Computing Based BWN. In *Proc. of Great Lakes Symposium on VLSI (GLSVLSI)*. ACM, 193–198.

[92] Bo Liu, Zhen Wang, Shisheng Guo, Huazhen Yu, Yu Gong, Jun Yang, and Longxing Shi. 2019. An Energy-Efficient Voice Activity Detector Using Deep Neural Networks and Approximate Computing. *Microelectronics Journal* 87 (2019), 12–21.

[93] Bo Liu, Zhen Wang, Wentao Zhu, Yuhao Sun, Zeyu Shen, Lepeng Huang, Yan Li, Yu Gong, and Wei Ge. 2019. An Ultra-Low Power Always-on Keyword Spotting Accelerator Using Quantized Convolutional Neural Network and Voltage-Domain Analog Switching Network-based Approximate Computing. *IEEE Access* 7 (2019), 186456–186469.

[94] Bo Liu, Zilong Zhang, Hao Cai, Reyuan Zhang, Zhen Wang, and Jun Yang. 2022. Self-Compensation Tensor Multiplication Unit for Adaptive Approximate Computing in Low-Power CNN Processing. *Science China Information Sciences* 65, 4 (2022), 1–2.

[95] Weiqiang Liu, Chongyan Gu, Máire O'Neill, Gang Qu, Paolo Montuschi, and Fabrizio Lombardi. 2020. Security in Approximate Computing and Approximate Computing for Security: Challenges and Opportunities. *Proc. IEEE* 108, 12 (2020), 2214–2231.

[96] Weiqiang Liu, Chongyan Gu, Gang Qu, and Máire O'Neill. 2018. Approximate Computing and Its Application to Hardware Security. In *Cyber-Physical Systems Security*. Springer, 43–67.

[97] Weiqiang Liu, Liangyu Qian, Chenghua Wang, Honglan Jiang, Jie Han, and Fabrizio Lombardi. 2017. Design of Approximate RADIX-4 Booth Multipliers for Error-Tolerant Computing. *IEEE Trans. Comput.* 66, 8 (2017), 1435–1441.

[98] Tieu-Khanh Luong, Van-Tinh Nguyen, Anh-Thai Nguyen, and Emanuel Popovici. 2019. Efficient Architectures and Implementation of Arithmetic Functions Approximation based Stochastic Computing. In *Proc. of 30th International Conference on Application-specific Systems, Architectures and Processors (ASAP)*. IEEE, 281–287.

[99] Fei Lyu, Xiaoqi Xu, Yu Wang, Yuanyong Luo, Yuxuan Wang, and Hongbing Pan. 2020. Ultralow-Latency VLSI Architecture based on a Linear Approximation Method for Computing $N$th Roots of Floating-Point Numbers. *IEEE Transactions on Circuits and Systems I: Regular Papers* 68, 2 (2020), 715–727.

[100] Wolfgang Maass. 1997. Networks of Spiking Neurons: The Third Generation of Neural Network Models. *Neural Networks* 10, 9 (1997), 1659–1671.

[101] Yashaswi Mannepalli, Viraj Bharadwaj Korede, and Madhav Rao. 2021. Novel Approximate Multiplier Designs for Edge Detection Application. In *Proc. of Great Lakes Symposium on VLSI (GLSVLSI)*. ACM, 371–377.

[102] Maurizio Martina, Guido Masera, Massimo Ruo Roch, and Gianluca Piccinini. 2015. Result-Biased Distributed-Arithmetic-based Filter Architectures for Approximately Computing the DWT. *IEEE Transactions on Circuits and Systems I: Regular Papers* 62, 8 (2015), 2103–2113.

[103] Sana Mazahir, Osman Hasan, and Muhammad Shafique. 2019. Self-Compensating Accelerators for Efficient Approximate Computing. *Microelectronics Journal* 88 (2019), 9–17.

[104] Joshua San Miguel, Jorge Albericio, Andreas Moshovos, and Natalie Enright Jerger. 2015. Doppelgänger: A Cache for Approximate Computing. In *Proc. of 48th International Symposium on Microarchitecture (MICRO)*. ACM, 50–61.

[105] Sparsh Mittal. 2016. A Survey of Techniques for Approximate Computing. *ACM Computing Surveys (CSUR)* 48, 4 (2016), 1–33.

[106] Mohammad Hossein Moaiyeri, Farnaz Sabetzadeh, and Shaahin Angizi. 2018. An Efficient Majority-based Compressor for Approximate Computing in the Nano Era. *Microsystem Technologies* 24, 3 (2018), 1589–1601.

[107] Masoomeh Momeni and Hadi Shahriar Shahhoseini. 2022. Energy Efficient 3D Network-on-Chip based on Approximate Communication. *Computer Networks* 203 (2022), 108652.
[108] Marcio Monteiro, Ismael Seidel, Mateus Grellert, José Luis Güntzel, Leonardo Soares, and Cristina Meinhardt. 2022. Exploring the Impacts of Multiple Kernel Sizes of Gaussian Filters Combined to Approximate Computing in Canny Edge Detection. In *Proc. of 13th Latin America Symposium on Circuits and System (LASCAS)*. IEEE, 1–4.
[109] Bert Moons and Marian Verhelst. 2015. DVAS: Dynamic Voltage Accuracy Scaling for Increased Energy-Efficiency in Approximate Computing. In *Proc. of International Symposium on Low Power Electronics and Design (ISLPED)*. IEEE/ACM, 237–242.
[110] Thierry Moreau, Mark Wyse, Jacob Nelson, Adrian Sampson, Hadi Esmaeilzadeh, Luis Ceze, and Mark Oskin. 2015. SNNAP: Approximate Computing on Programmable SoCs via Neural Acceleration. In *Proc. of 21st International Symposium on High Performance Computer Architecture (HPCA)*. IEEE, 603–614.
[111] Amir Najafi, Lennart Bamberg, Ardalan Najafi, and Alberto Garcia-Ortiz. 2019. Integer-Value Encoding for Approximate on-Chip Communication. *IEEE Access* 7 (2019), 179220–179234.
[112] Gabriel Luca Nazar, Pedro Kopper, Marcos Leipnitz, and Ben Juurlink. 2021. Lightweight Dual Modular Redundancy through Approximate Computing. In *Proc. of XI Brazilian Symposium on Computing Systems Engineering (SBESC)*. IEEE, 1–8.
[113] Geneviève Ndour, Tiago Trevisan Jost, Anca Molnos, Yves Durand, and Arnaud Tisserand. 2019. Evaluation of Variable Bit-Width Units in a RISC-V Processor for Approximate Computing. In *Proc. of 16th International Conference on Computing Frontiers (CF)*. ACM, 344–349.
[114] Kumud Nepal, Soheil Hashemi, Hokchhay Tann, Ruth Iris Bahar, and Sherief Reda. 2016. Automated High-Level Generation of Low-Power Approximate Computing Circuits. *IEEE Transactions on Emerging Topics in Computing* 7, 1 (2016), 18–30.
[115] William Stafford Noble. 2006. What is a Support Vector Machine? *Nature Biotechnology* 24, 12 (2006), 1565–1567.
[116] Tuaha Nomani, Mujahid Mohsin, Zahid Pervaiz, and Muhammad Shafique. 2020. xUAVs: Towards Efficient Approximate Computing for UAVs—Low Power Approximate Adders with Single LUT Delay for FPGA-based Aerial Imaging Optimization. *IEEE Access* 8 (2020), 102982–102996.
[117] Bernard Nongpoh, Rajarshi Ray, and Ansuman Banerjee. 2019. Approximate Computing for Multithreaded Programs in Shared Memory Architectures. In *Proc. of 17th International Conference on Formal Methods and Models for System Design (MEMOCODE)*. ACM, 1–9.
[118] Bernard Nongpoh, Rajarshi Ray, Moumita Das, and Ansuman Banerjee. 2019. Enhancing Speculative Execution With Selective Approximate Computing. *ACM Transactions on Design Automation of Electronic Systems (TODAES)* 24, 2 (2019), 1–29.
[119] Aleksandr Ometov, Oliver Liombe Molua, Mikhail Komarov, and Jari Nurmi. 2022. A Survey of Security in Cloud, Edge, and Fog Computing. *Sensors* 22, 3 (2022), 927.
[120] Aleksandr Ometov and Jari Nurmi. 2022. Towards Approximate Computing for Achieving Energy vs. Accuracy Trade-offs. In *Proc. of Design, Automation & Test in Europe Conference & Exhibition (DATE)*. IEEE, 632–635.
[121] Aleksandr Ometov, Viktoriia Shubina, Lucie Klus, Justyna Skibińska, Salwa Saafi, Pavel Pascacio, Laura Flueratoru, Darwin Quezada Gaibor, Nadezhda Chukhno, Olga Chukhno, Asad Ali, Asma Channa, Ekaterina Svertoka, Waleed Bin Qaim, Raúl Casanova-Marqués, Sylvia Holcer, Joaquín Torres-Sospedra, Sven Casteleyn, Giuseppe Ruggeri, Giuseppe Araniti, Radim Burget, Jiri Hosek, and Elena Simona Lohan. 2021. A Survey on Wearable Technology: History, State-of-the-Art and Current Challenges. *Computer Networks* 193 (2021), 108074.
[122] Roberto Osorio and Gabriel Rodriguez. 2019. Truncated SIMD Multiplier Architecture for Approximate Computing in Low-Power Programmable Processors. *IEEE Access* 7 (2019), 56353–56366.
[123] Matthew Page, Joanne McKenzie, Patrick Bossuyt, Isabelle Boutron, Tammy Hoffmann, Cynthia Mulrow, Larissa Shamseer, Jennifer Tetzlaff, Elie Akl, Sue Brennan, et al. 2021. The PRISMA 2020 Statement: An Updated Guideline for Reporting Systematic Reviews. *BMJ* 372 (2021), 11.
[124] Krishna Palem and Avinash Lingamneni. 2013. Ten Years of Building Broken Chips: The Physics and Engineering of Inexact Computing. *ACM Transactions on Embedded Computing Systems (TECS)* 12, 2s (2013), 1–23.
[125] Alberto Paltrinieri, Riccardo Peloso, Guido Masera, Muhammad Shafique, and Maurizio Martina. 2019. On the Effect of Approximate-Computing in Motion Estimation. *Journal of Low Power Electronics* 15, 1 (2019), 40–50.
[126] Francesca Palumbo and Carlo Sau. 2021. Reconfigurable and Approximate Computing for Video Coding. *arXiv preprint: 2103.03712* (2021), 33.
[127] Keerthana Pamidimukkala, Kyung Ki Kim, Yong-Bin Kim, and Minsu Choi. 2018. Generalized Adaptive Variable Bit Truncation Method for Approximate Stochastic Computing. In *Proc. of 15th International SoC Design Conference (ISOCC)*. IEEE, 218–219.
[128] Behrooz Parhami. 2018. A Case for Table-Based Approximate Computing. In *Proc. of 9th Information Technology, Electronics and Mobile Communication Conference (IEMCON)*. IEEE, 650–653.
[129] Jongse Park, Emmanuel Amaro, Divya Mahajan, Bradley Thwaites, and Hadi Esmaeilzadeh. 2016. AxGames: Towards Crowdsourcing Quality Target Determination in Approximate Computing. In *Proc. of 21st International Conference on Architectural Support for Programming Languages and Operating Systems (ASPLOS)*. ACM, 623–636.

[130] Zhenghao Peng, Xuyang Chen, Chengwen Xu, Naifeng Jing, Xiaoyao Liang, Cewu Lu, and Li Jiang. 2018. AXNet: ApproXimate Computing Using an End-to-End Trainable Neural Network. In *Proc. of 37th International Conference on Computer-Aided Design (ICCAD)*. ACM, 1–8.

[131] Michael Pfeiffer and Thomas Pfeil. 2018. Deep Learning with Spiking Neurons: Opportunities and Challenges. *Frontiers in Neuroscience* 12 (2018), 774.

[132] Ali Piri, Sepide Saeedi, Mario Barbareschi, Bastien Deveautour, Stefano Di Carlo, Ian O'Connor, Alessandro Savino, Marcello Traiola, and Alberto Bosio. 2022. Input-Aware Approximate Computing. In *Proc. of International Conference on Automation, Quality and Testing, Robotics (AQTR)*. IEEE, 1–6.

[133] Stefania Preatto, Andrea Giannini, Luca Valente, Guido Masera, and Maurizio Martina. 2020. Optimized VLSI Architecture of HEVC Fractional Pixel Interpolators with Approximate Computing. *Journal of Low Power Electronics and Applications* 10, 3 (2020), 24.

[134] Waleed Bin Qaim, Aleksandr Ometov, Claudia Campolo, Antonella Molinaro, Elena Simona Lohan, and Jari Nurmi. 2021. Understanding the Performance of Task Offloading for Wearables in a Two-Tier Edge Architecture. In *Proc. of 13th International Congress on Ultra Modern Telecommunications and Control Systems and Workshops (ICUMT)*. IEEE, 1–9.

[135] Fei Qiao, Ni Zhou, Yuanchang Chen, and Huazhong Yang. 2015. Approximate Computing in Chrominance Cache for Image/Video Processing. In *Proc. of International Conference on Multimedia Big Data (BigMM)*. IEEE, 180–183.

[136] Karri Manikantta Reddy, Moodabettu Harishchandra Vasantha, Yernad Balachandra Nithin Kumar, and Devesh Dwivedi. 2020. Design of Approximate Booth Squarer for Error-Tolerant Computing. *IEEE Transactions on Very Large Scale Integration (VLSI) Systems* 28, 5 (2020), 1230–1241.

[137] Karri Manikantta Reddy, Moodabettu Harishchandra Vasantha, Yernad Balachandra Nithin Kumar, Ch Keshava Gopal, and Devesh Dwivedi. 2021. Quantization Aware Approximate Multiplier and Hardware Accelerator for Edge Computing of Deep Learning Applications. *Integration* 81 (2021), 268–279.

[138] Francesco Regazzoni and Ilia Polian. 2020. Side Channel Attacks vs Approximate Computing. In *Proc. of Great Lakes Symposium on VLSI (GLSVLSI)*. ACM, 321–326.

[139] Md Farhadur Reza and Paul Ampadu. 2019. Approximate Communication Strategies for Energy-Efficient and High Performance NoC: Opportunities and Challenges. In *Proc. of Great Lakes Symposium on VLSI (GLSVLSI)*. ACM, 399–404.

[140] Gennaro Severino Rodrigues, Juan Fonseca, Fabio Benevenuti, Fernanda Kastensmidt, and Alberto Bosio. 2019. Exploiting Approximate Computing for Low-Cost Fault Tolerant Architectures. In *Proc. of 32nd Symposium on Integrated Circuits and Systems Design (SBCCI)*. IEEE, 1–6.

[141] Kaushik Roy, Akhilesh Jaiswal, and Priyadarshini Panda. 2019. Towards Spike-Based Machine Intelligence with Neuromorphic Computing. *Nature* 575, 7784 (2019), 607–617.

[142] Jochen Rust, Nils Heidmann, and Steffen Paul. 2017. Approximate Computing of Two-Variable Numeric Functions Using Multiplier-Less Gradients. *Microprocessors and Microsystems* 48 (2017), 48–55.

[143] Christos Sakalis, Carl Leonardsson, Stefanos Kaxiras, and Alberto Ros. 2016. Splash-3: A Properly Synchronized Benchmark Suite for Contemporary Research. In *Proc. of International Symposium on Performance Analysis of Systems and Software (ISPASS)*. IEEE, 101–111.

[144] Ferdos Salmanpour, Mohammad Hossein Moaiyeri, and Farnaz Sabetzadeh. 2021. Ultra-Compact Imprecise 4:2 Compressor and Multiplier Circuits for Approximate Computing in Deep Nanoscale. *Circuits, Systems, and Signal Processing* 40, 9 (2021), 4633–4650.

[145] Syed Shakib Sarwar, Gopalakrishnan Srinivasan, Bing Han, Parami Wijesinghe, Akhilesh Jaiswal, Priyadarshini Panda, Anand Raghunathan, and Kaushik Roy. 2018. Energy Efficient Neural Computing: A Study of Cross-Layer Approximations. *IEEE Journal on Emerging and Selected Topics in Circuits and Systems* 8, 4 (2018), 796–809.

[146] Yuuki Sato, Takanori Tsumura, Tomoaki Tsumura, and Yasuhiko Nakashima. 2015. An Approximate Computing Stack based on Computation Reuse. In *Proc. of 3rd International Symposium on Computing and Networking (CANDAR)*. IEEE, 378–384.

[147] Carlo Sau, Francesca Palumbo, Maxime Pelcat, Julien Heulot, Erwan Nogues, Daniel Menard, Paolo Meloni, and Luigi Raffo. 2017. Challenging the Best HEVC Fractional Pixel FPGA Interpolators with Reconfigurable and Multifrequency Approximate Computing. *IEEE Embedded Systems Letters* 9, 3 (2017), 65–68.

[148] Ilaria Scarabottolo, Giovanni Ansaloni, George Anthony Constantinides, Laura Pozzi, and Sherief Reda. 2020. Approximate Logic Synthesis: A Survey. *Proc. IEEE* 108, 12 (2020), 2195–2213.

[149] Michael Schaffner, Frank Kagan Gürkaynak, Aljosa Smolic, Hubert Kaeslin, and Luca Benini. 2014. An Approximate Computing Technique for Reducing the Complexity of a Direct-Solver for Sparse Linear Systems in Real-Time Video Processing. In *Proc. of 51st Design Automation Conference (DAC)*. ACM, 1–6.

[150] Lukas Sekanina. 2021. Evolutionary Algorithms in Approximate Computing: A Survey. *arXiv preprint arXiv:2108.07000* (2021), 12.

[151] Sanchari Sen, Swagath Venkataramani, and Anand Raghunathan. 2017. Approximate Computing for Spiking Neural Networks. In *Proc. of Design, Automation & Test in Europe Conference & Exhibition (DATE)*. IEEE, 193–198.

[152] Sayandeep Sen, Tan Zhang, Syed Gilani, Shreesha Srinath, Suman Banerjee, and Sateesh Addepalli. 2012. Design and Implementation of an "Approximate" Communication System for Wireless Media Applications. *IEEE/ACM Transactions on Networking* 21, 4 (2012), 1035–1048.

[153] Ramu Seva, Prashanthi Metku, Kyung Ki Kim, Yong-Bin Kim, and Minsu Choi. 2016. Approximate Stochastic Computing (ASC) for Image Processing Applications. In *Proc. of 13th International SoC Design Conference (ISOCC)*. IEEE, 31–32.

[154] Botang Shao and Peng Li. 2015. Array-based Approximate Arithmetic Computing: A General Model and Applications to Multiplier and Squarer Design. *IEEE Transactions on Circuits and Systems I: Regular Papers* 62, 4 (2015), 1081–1090.

[155] Weisong Shi, Jie Cao, Quan Zhang, Youhuizi Li, and Lanyu Xu. 2016. Edge Computing: Vision and Challenges. *IEEE Internet of Things Journal* 3, 5 (2016), 637–646.

[156] Majid Shoushtari, Abbas BanaiyanMofrad, and Nikil Dutt. 2015. Exploiting Partially-Forgetful Memories for Approximate Computing. *IEEE Embedded Systems Letters* 7, 1 (2015), 19–22.

[157] Sharad Sinha and Wei Zhang. 2016. Low-Power FPGA Design Using Memoization-based Approximate Computing. *IEEE Transactions on Very Large Scale Integration (VLSI) Systems* 24, 8 (2016), 2665–2678.

[158] Midde Venkata Siva and EP Jayakumar. 2020. Approximated Algorithm and Low Cost VLSI Architecture for Edge Enhanced Image Scaling. In *Proc. of International Conference on Industry 4.0, Artificial Intelligence, and Communications Technology (IAICT)*. IEEE, 125–130.

[159] Farhana Sharmin Snigdha, Deepashree Sengupta, Jiang Hu, and Sachin Sapatnekar. 2016. Optimal Design of JPEG Hardware under the Approximate Computing Paradigm. In *Proc. of 53rd Design Automation Conference (DAC)*. ACM, 1–6.

[160] Leonardo Bandeira Soares, Julio Oliveira, Eduardo Antonio César da Costa, and Sergio Bampi. 2020. An Energy-Efficient and Approximate Accelerator Design for Real-Time Canny Edge Detection. *Circuits, Systems, and Signal Processing* 39 (2020), 6098–6120.

[161] Haiyue Song, Chengwen Xu, Qiang Xu, Zhuoran Song, Naifeng Jing, Xiaoyao Liang, and Li Jiang. 2018. Invocation-Driven Neural Approximate Computing with a Multiclass-Classifier and Multiple Approximators. In *Proc. of 37th International Conference on Computer-Aided Design (ICCAD)*. ACM, 1–8.

[162] Giuseppe Tagliavini, Andrea Marongiu, Davide Rossi, and Luca Benini. 2016. Always-on Motion Detection with Application-Level Error Control on a Near-Threshold Approximate Computing Platform. In *Proc. of 23rd International Conference on Electronics, Circuits and Systems (ICECS)*. IEEE, 552–555.

[163] Giuseppe Tagliavini, Davide Rossi, Andrea Marongiu, and Luca Benini. 2016. Synergistic HW/SW Approximation Techniques for Ultralow-Power Parallel Computing. *IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems* 37, 5 (2016), 982–995.

[164] Tomoaki Tsumura, Ikuma Suzuki, Yasuki Ikeuchi, Hiroshi Matsuo, Hiroshi Nakashima, and Yasuhiko Nakashima. 2007. Design and Evaluation of an Auto-Memoization Processor. In *Proc. of 25th International Multi-Conference: Parallel and Distributed Computing and Networks (IASTED)*. ACM, 245–250.

[165] Fengbin Tu, Shouyi Yin, Peng Ouyang, Leibo Liu, and Shaojun Wei. 2018. Reconfigurable Architecture for Neural Approximation in Multimedia Computing. *IEEE Transactions on Circuits and Systems for Video Technology* 29, 3 (2018), 892–906.

[166] Georgios Tziantzioulis, Ali Murat Gok, SM Faisal, Nikolaos Hardavellas, Seda Ogrenci-Memik, and Srinivasan Parthsarathy. 2016. Lazy Pipelines: Enhancing Quality in Approximate Computing. In *Proc. of Design, Automation & Test in Europe Conference & Exhibition (DATE)*. IEEE, 1381–1386.

[167] Kimiyoshi Usami, Hajime Ochi, and Yoshinori Ono. 2020. Approximate Computing based on Latest-result Reuse for Image Edge Detection. In *Proc. of 35th International Technical Conference on Circuits/Systems, Computers and Communications (ITC-CSCC)*. IEEE, 234–239.

[168] Martin Van Leussen, Jos Huisken, Lei Wang, Hailong Jiao, and Jose Pineda De Gyvez. 2017. Reconfigurable Support Vector Machine Classifier with Approximate Computing. In *Proc. of Computer Society Annual Symposium on VLSI (ISVLSI)*. IEEE, 13–18.

[169] Swagath Venkataramani, Srimat Tirumala Chakradhar, Kaushik Roy, and Anand Raghunathan. 2015. Approximate Computing and the Quest for Computing Efficiency. In *Proc. of 52nd Design Automation Conference (DAC)*. ACM, 1–6.

[170] Swagath Venkataramani, Vinay Kumar Chippa, Srimat Tirumala Chakradhar, Kaushik Roy, and Anand Raghunathan. 2013. Quality Programmable Vector Processors for Approximate Computing. In *Proc. of 46th International Symposium on Microarchitecture (MICRO)*. IEEE, 1–12.

[171] Swagath Venkataramani, Xiao Sun, Naigang Wang, Chia-Yu Chen, Jungwook Choi, Mingu Kang, Ankur Agarwal, Jinwook Oh, Shubham Jain, Tina Babinsky, et al. 2020. Efficient AI System Design with Cross-Layer Approximate Computing. *Proc. IEEE* 108, 12 (2020), 2232–2250.

[172] Yan Verdeja Herms and Yanjing Li. 2019. Crash Skipping: A Minimal-Cost Framework for Efficient Error Recovery in Approximate Computing Environments. In *Proc. of Great Lakes Symposium on VLSI (GLSVLSI)*. ACM, 129–134.

[173] Pradnya Vikhar. 2016. Evolutionary Algorithms: A Critical Review and Its Future Prospects. In *Proc. of International Conference on Global Trends in Signal Processing, Information Computing and Communication (ICGTSPICC)*. IEEE, 261–265.

[174] Qian Wang, Youjie Li, and Peng Li. 2016. Liquid State Machine based Pattern Recognition on FPGA with Firing-Activity Dependent Power Gating and Approximate Computing. In *Proc. of International Symposium on Circuits and Systems (ISCAS)*. IEEE, 361–364.

[175] Ting Wang, Qian Zhang, Nam Sung Kim, and Qiang Xu. 2016. On Effective and Efficient Quality Management for Approximate Computing. In *Proc. of International Symposium on Low Power Electronics and Design (ISLPED)*. ACM, 156–161.

[176] Ye Wang, Jian Dong, Qian Xu, Zhaojun Lu, and Gang Qu. 2020. Is It Approximate Computing or Malicious Computing?. In *Proc. of Great Lakes Symposium on VLSI (GLSVLSI)*. ACM, 333–338.
[177] Ye Wang, Jian Dong, Qian Xu, and Gang Qu. 2021. FTApprox: A Fault-Tolerant Approximate Arithmetic Computing Data Format. In *Proc. of Design, Automation & Test in Europe Conference & Exhibition (DATE)*. IEEE, 1548–1551.
[178] Ying Wang, Huawei Li, and Xiaowei Li. 2017. Real-Time Meets Approximate Computing: An Elastic CNN Inference Accelerator with Adaptive Trade-Off between QoS and QoR. In *Proc. of 54th Design Automation Conference (DAC)*. ACM, 1–6.
[179] Yang Wang, Yubin Qin, Dazheng Deng, Jingchuan Wei, Yang Zhou, Yuanqi Fan, Tianbao Chen, Hao Sun, Leibo Liu, Shaojun Wei, et al. 2022. A 28nm 27.5 TOPS/W Approximate-Computing-Based Transformer Processor with Asymptotic Sparsity Speculating and Out-of-Order Computing. In *Proc. of International Solid-State Circuits Conference (ISSCC)*. IEEE, 1–3.
[180] Zhihui Wang, Shouyi Yin, Fengbin Tu, Leibo Liu, and Shaojun Wei. 2018. An Energy Efficient JPEG Encoder with Neural Network Based Approximation and Near-Threshold Computing. In *Proc. of International Symposium on Circuits and Systems (ISCAS)*. IEEE, 1–5.
[181] Steven Cameron Woo, Moriyoshi Ohara, Evan Torrie, Jaswinder Pal Singh, and Anoop Gupta. 1995. The SPLASH-2 Programs: Characterization and Methodological Considerations. *ACM SIGARCH Computer Architecture News* 23, 2 (1995), 24–36.
[182] Di Wu and Joshua San Miguel. 2021. Special Session: When Dataflows Converge: Reconfigurable and Approximate Computing for Emerging Neural Networks. In *Proc. of 39th International Conference on Computer Design (ICCD)*. IEEE, 9–12.
[183] Hang Xiao, Haobo Xu, Xiaoming Chen, Yujie Wang, and Yinhe Han. 2021. Fast and High-Accuracy Approximate MAC Unit Design for CNN Computing. *IEEE Embedded Systems Letters* 14, 3 (2021), 155–158.
[184] Jie Xiao, Jianhao Hu, and Kaining Han. 2019. Low Complexity Expectation Propagation Detection for SCMA Using Approximate Computing. In *Proc. of Global Communications Conference (GLOBECOM)*. IEEE, 1–6.
[185] Siyuan Xiao, Xiaohang Wang, Maurizio Palesi, Amit Kumar Singh, Liang Wang, and Terrence Mak. 2020. On Performance Optimization and Quality Control for Approximate-Communication-Enabled Networks-on-Chip. *IEEE Trans. Comput.* 70, 11 (2020), 1817–1830.
[186] Yan Xing, Ziji Zhang, Yiduan Qian, Qiang Li, and Yajuan He. 2018. An Energy-Efficient Approximate DCT for Wireless Capsule Endoscopy Application. In *Proc. of International Symposium on Circuits and Systems (ISCAS)*. IEEE, 1–4.
[187] Qiang Xu, Todd Mytkowicz, and Nam Sung Kim. 2015. Approximate Computing: A Survey. *IEEE Design & Test* 33, 1 (2015), 8–22.
[188] Siyuan Xu and Benjamin Carrion Schafer. 2018. Toward Self-Tunable Approximate Computing. *IEEE Transactions on Very Large Scale Integration (VLSI) Systems* 27, 4 (2018), 778–789.
[189] Tongxin Yang, Tomoaki Ukezono, and Toshinori Sato. 2018. A Low-Power yet High-Speed Configurable Adder for Approximate Computing. In *Proc. of International Symposium on Circuits and Systems (ISCAS)*. IEEE, 1–5.
[190] Tongxin Yang, Tomoaki Ukezono, and Toshinori Sato. 2022. Reducing Power Consumption using Approximate Encoding for CNN Accelerators at the Edge. In *Proc. of Great Lakes Symposium on VLSI (GLSVLSI)*. ACM, 229–235.
[191] Wu Yang and Himanshu Thapliyal. 2020. Low-Power and Energy-Efficient Full Adders with Approximate Adiabatic Logic for Edge Computing. In *Proc. of Computer Society Annual Symposium on VLSI (ISVLSI)*. IEEE, 312–315.
[192] Wu Yang and Himanshu Thapliyal. 2021. Approximate Adiabatic Logic for Low-Power and Secure Edge Computing. *IEEE Consumer Electronics Magazine* 11, 1 (2021), 88–94.
[193] Zhixi Yang, Jie Han, and Fabrizio Lombardi. 2015. Transmission Gate-based Approximate Adders for Inexact Computing. In *Proc. of International Symposium on Nanoscale Architectures (NANOARCH)*. IEEE, 145–150.
[194] Zhixi Yang, Ajaypat Jain, Jinghang Liang, Jie Han, and Fabrizio Lombardi. 2013. Approximate XOR/XNOR-based Adders for Inexact Computing. In *Proc. of 13th International Conference on Nanotechnology (IEEE-NANO)*. IEEE, 690–693.
[195] Ruoheng Yao, Lei Chen, Pingcheng Dong, Zhuoyu Chen, and Fengwei An. 2022. A Compact Hardware Architecture for Bilateral Filter with the Combination of Approximate Computing and Look-up Table. *IEEE Transactions on Circuits and Systems II: Express Briefs* 69, 7 (2022), 3324–3328.
[196] Pruthvy Yellu, Landon Buell, Miguel Mark, Michel A Kinsy, Dongpeng Xu, and Qiaoyan Yu. 2021. Security Threat Analyses and Attack Models for Approximate Computing Systems: From Hardware and Micro-architecture Perspectives. *ACM Transactions on Design Automation of Electronic Systems (TODAES)* 26, 4 (2021), 1–31.
[197] Ashkan Yousefpour, Caleb Fung, Tam Nguyen, Krishna Kadiyala, Fatemeh Jalali, Amirreza Niakanlahiji, Jian Kong, and Jason Jue. 2019. All One Needs to Know about Fog Computing and Related Edge Computing Paradigms: A Complete Survey. *Journal of Systems Architecture* 98 (2019), 289–330.
[198] Shuyuan Yu, Yibo Liu, and Sheldon Tan. 2021. Approximate Divider Design Based on Counting-Based Stochastic Computing Division. In *Proc. of 3rd Workshop on Machine Learning for CAD (MLCAD)*. IEEE, 1–6.
[199] Vinícius Zanandrea, Douglas Borges, Vagner Santos da Rosa, and Cristina Meinhardt. 2021. Exploring Approximate Computing and Near-Threshold Operation to Design Energy-Efficient Multipliers. In *Proc. of 34th Symposium on Integrated Circuits and Systems Design (SBCCI)*. IEEE, 1–6.
[200] Georgios Zervakis, Hassaan Saadat, Hussam Amrouch, Andreas Gerstlauer, Sri Parameswaran, and Jörg Henkel. 2021. Approximate Computing for ML: State-of-the-art, Challenges and Visions. In *Proc. of 26th Asia and South Pacific Design Automation Conference (ASP-DAC)*. IEEE, 189–196.

[201] Xianwei Zhang, Youtao Zhang, Bruce Childers, and Jun Yang. 2017. DrMP: Mixed Precision-Aware DRAM for High Performance Approximate and Precise Computing. In *Proc. of 26th International Conference on Parallel Architectures and Compilation Techniques (PACT)*. IEEE, 53–63.

[202] Yangcan Zhou, Zhiyu Chen, Jun Lin, and Zhongfeng Wang. 2018. A High-Speed Successive-Cancellation Decoder for Polar Codes Using Approximate Computing. *IEEE Transactions on Circuits and Systems II: Express Briefs* 66, 2 (2018), 227–231.

[203] Yangcan Zhou, Jun Lin, and Zhongfeng Wang. 2017. Energy Efficient SVM Classifier Using Approximate Computing. In *Proc. of 12th International Conference on ASIC (ASICON)*. IEEE, 1045–1048.

[204] Feiyu Zhu, Shaowei Zhen, Xilin Yi, Haoran Pei, Bowen Hou, and Yajuan He. 2022. Design of Approximate Radix-256 Booth Encoding for Error-Tolerant Computing. *IEEE Transactions on Circuits and Systems II: Express Briefs* 69, 4 (2022), 2286–2290.

## A METHODOLOGY

For this systematic literature review, we followed the PRISMA guidelines [123]. As such, we initially set out to identify a set of appropriate search entry formed by keywords and their synonyms, allowing us to form a comprehensive search expression. A scan of the most frequently cited, relevant survey papers led to the following expression:

```
( approximat* OR inexact OR inaccurate OR ``good enough'' ) AND
( computing OR edge OR cloud OR fog OR communic* OR wireless )
```

A search was performed using Scopus [45]. We applied filters limiting results to English publications from 2013 till 2022, whose main topic is in computing or engineering. This resulted in a total of 1309 potentially relevant publications (as of August 1, 2022). We filtered these results in three rounds: **1)** a coarse filtering based primarily on paper titles; **2)** a filtering based on abstracts and conclusions; and **3)** a fine filtering of the remaining papers based on a brief overview of their full contents. We applied the following general exclusion criteria: **C1** works not related to AxC or Edge computing; **C2** works on software-only or emerging transistor technology-based AxC techniques, including, e.g., memristive devices; **C3** works on AxC *frameworks* whose main contributions are not hardware but rather related to synthesis, compilation, significance analysis, or tooling; **C4** invited works with no technical content; and **C5** full text not available. Criteria **C2** and **C3** ensure the review's relevancy to most digital hardware design researchers but imply the exclusion of articles such as [11, 52, 74, 130]. We further exclude the majority of survey and review works, including only those considered fundamental to the AxC field and those covering related topics not comprised by other publications. We also filter clearly iterative works, excluding all but the most recent related publication. Lastly, to avoid unreasonably broadening the scope of the review, we limit inclusions to direct search results and, thus, do not include works cited by the selected publications. All three authors have participated in the filtering to ensure its fairness.

The above resulted in a set of 167 relevant publications. The publications are distributed over the included years as shown in Fig. 11. The number of publications is on an upward trend with, so far, most publications in 2019, indicating an increasing interest in the field.

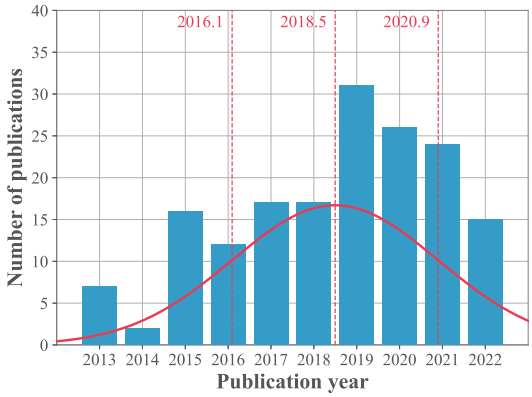


Fig. 11. Distribution of selected publications per year with lines for mean +/- one standard deviation.

We have classified 140 of the selected publications into the following main categories (as previously highlighted in Fig. 2): **fundamental AxC techniques** general approximation methods that can be applied in architectures for different applications; **AxC-enabled hardware architectures** larger-scale architectures incorporating one or more fundamental AxC techniques; and **applications of AxC** examples of AxC techniques applied to one or more specific applications. The remaining 27 works are surveys or survey-style papers that motivate the use of AxC. Thus, the unclassified works span several of the listed categories.

# B OVERVIEW TABLES

Table 6. A summary of reviewed fundamental AxC techniques by class and publication year.

| Class | Ref. | Main content | Merits | Demerits |
|---|---|---|---|---|
| Circuit-level | [53] 2015 | Exploration of availability and power gains from reducing refresh rate in embedded DRAMs. | Availability gains are inversely proportional to transistor size. | **N/A** |
| | [109] 2015 | A scheme for dynamic voltage-accuracy scaling in pipelined designs. | Energy savings with run-time reconfigurability. | Some area and leakage power overheads. |
| | [156] 2015 | Using caches with faulty ways due to manufacturing imperfections in approximate systems. | Improved yield and leakage energy savings at negligible errors. | Relies on developer annotation. |
| | [163] 2016 | An ultra-low power processing system for NTV operation utilizing a hybrid SRAM/standard cell memory architecture. | Reduced energy consumption with compiler support. | Relies on developer annotation. |
| | [166] 2016 | Lazy write-back of results from voltage over-scaled FUs allowing for their results to converge closer to exact ones. | Reduced bit error rate at low FU utilization. | Hardware overheads for extra issue and write-back logic. |
| | [192] 2021 | Adiabatic logic as an alternative to CMOS, specifically for approximate adders (like [191]). | Improved power consumption and resilience to side-channel attacks. | Costly clock net distribution required for adiabatic logic. |
| Arithmetic | [194] 2013 | Three approximate full adders based on XOR/XNOR logic. | Reduced area and power consumption. | Missing error analysis when applied to $n$-bit adders. |
| | [7] 2014 | Four approximate full adders based on CMOS logic. | Reduced area and power consumption. | **N/A** |
| | [83] 2014 | Approximate segmented adder and comparator with reduced-length carry propagation. | Improved area, power, delay, and error metrics. | **N/A** |
| | [29] 2015 | Three approximate subtractors and an approximate divider utilizing them. | Reduced power consumption at negligible quality degradation. | **N/A** |
| | [154] 2015 | A technique for designing approximate array-based multipliers and squarers with signature-based error compensation. | Reduced error metrics. | Area and power overheads from compensation logic. |
| | [193] 2015 | Two approximate full adders based on transmission gates. | Reduced delay and power consumption. | **N/A** |
| | [42] 2017 | Four approximate full adders based on CMOS logic and a segmented adder with error compensation. | Reduced delay and error metrics. | Area and power overheads from compensation logic. |
| | [97] 2017 | Two approximate Booth encodings for multiplier design. | Reduced area, power, delay, and error metrics. | **N/A** |
| | [17] 2018 | A technique for approximate partial product reduction in array-based multipliers with maskable carries. | Reduced area and power consumption. | Worsened error metrics. |
| | [35] 2018 | Approximate segmented adder with feedback-based error compensation. | Reduced area, power, delay, and error metrics. | **N/A** |
| | [64] 2018 | Approximate floating-point MAC unit with OR-based mantissa addition, error compensation, and exact re-computation. | Improved energy efficiency and system speedup. | Area overhead from error compensation. |
| | [106] 2018 | An approximate compressor based on majority gates for partial product reduction. | Reduced area, power, delay, and error metrics. | **N/A** |
| | [189] 2018 | Approximate segmented adder with maskable carry propagation. | Reduced power consumption with run-time reconfigurability. | Area overhead from carry masking logic. |
| | [66] 2019 | Four approximate subtractors and an approximate divider utlizing them. | Reduced, area, delay, and power at negligible quality degradation. | Missing error analysis when applied to $n$-bit dividers. |
| | [103] 2019 | Design of concatenated arithmetic circuit-based accelerators which cancel out errors through use of complementary circuits with opposite error polarity. | Reduced error with no effect on power consumption. | Requires deterministic error distribution. |
| | [122] 2019 | Design of approximate SIMD-style multiplier with adapted Booth encoding and truncated partial product array. | Reduced area and energy consumption. | **N/A** |
| | [62] 2020 | A technique for designing fixed-width multipliers with Booth-encoded partial products with low errors based on probability analysis. | Reduced area, power, delay, and error metrics. | **N/A** |
| | [99] 2020 | Piece-wise linear approximation of $N$th roots of floating-point numbers with corresponding hardware architecture. | Reduced area, power, and latency at no quality degradation. | **N/A** |
| | [116] 2020 | Approximate adder designs with limited carry propagation targeting FPGAs. | Reduced power consumption and error probability. | High mean error distance from approximate carry propagation. |
| | [136] 2020 | Design of squarers with approximate Booth encoding and partial product reduction. | Reduced area, power, delay, and error metrics. | **N/A** |
| | [191] 2020 | Two approximate full adders based on adiabatic logic. | Reduced area and power consumption. | Costly clock net distribution required for adiabatic logic. |
| | [101] 2021 | Sequential multiplier with approximate partial product accumulation. | Reduced area, power, delay, and error metrics. | Unclear implementation details. |
| | [144] 2021 | An approximate compressor based on CMOS logic for partial product reduction. | Reduced area, power, and delay. | Substantially worsened output quality. |
| | [183] 2021 | Approximate MAC unit with inexact partial product reduction and accumulation. | Reduced area, power, and delay. | Slightly worsened output quality. |

Continued on the next page

| | | | | |
|---|---|---|---|---|
| *cont.* | [199] 2021 | Exploration of using inexact adders in different multipliers in NTV operation. | Different multipliers show different benefits of approximation. | Unclear power consumption benefits. |
| | [132] 2022 | A technique for designing approximate multipliers for particular ranges of input values. | **N/A** | Mismatching conclusions and reported results. |
| | [204] 2022 | Approximate high-radix Booth encoding for multiplier design. | Improved error metrics at low area. | **N/A** |
| **Stochastic** | [153] 2016 | Truncated seeds for bitstream generation compensated for by increased run-time. | Reduced overall bitstream generation run-time. | Lacks run-time reconfigurability. |
| | [127] 2018 | Generalization of [153] to dynamically truncate operands based on most significant asserted bit. | Improved output quality. | Only works for unsigned operands. |
| | [70] 2019 | Approximate adder design with the least-significant bits computing stochastically. | Reduced area and run-time. | Uncertain output quality results. |
| | [20] 2020 | Extended stochastic number representation supporting subtraction and division operations. | Reduced area, power, delay, and error metrics. | **N/A** |
| | [48] 2021 | Approximate unary constant coefficient multiplier design. | Reduced area. | Complex implementation of unary-binary conversions. |
| | [198] 2021 | Counter-based approximate divider with deterministic bitstream generation and early termination. | Reduced area, delay, and energy consumption. | **N/A** |
| **Others** | [38] 2015 | Piece-wise Taylor expansion-based approximation of non-linear functions and corresponding co-processor architecture. | Low-overhead with arbitrary accuracy. | Limited applicability given popular alternatives, e.g., ReLU. |
| | [157] 2016 | HLS post-processing step for generating and inserting static and dynamic memoization wrappers on specific logic blocks. | Significant power savings at low resource consumption overheads. | No reconfigurability at run-time. |
| | [142] 2017 | Segmented linear approximation of complex two-variable functions implemented in multiplier-free binary trees. | Low hardware complexity and efficient generation flow. | Potentially high memory area. |
| | [43] 2018 | Evaluation of approximate adders and memoization on FPGAs. | Clear benefits of memoization on FPGAs. | Limited evaluation. |
| | [128] 2018 | Table-based approximate arithmetic circuits. | High power efficiency and easy error analysis. | Potentially high memory area. |
| | [98] 2019 | Stochastic computing-based function approximation architectures. | Low area, power, and error metrics. | Long latency per function evaluation. |
| | [71] 2020 | Table-based technique based on pre-computed K-means clusterings and run-time nearest-centroid classification. | Low hardware complexity. | No reconfigurability at run-time. |
| | [72] 2020 | CGRA-like accelerator for bi-section NNs used for neural approximation of arbitrary input functions. | Low-complexity architecture with run-time reconfigurability. | Bi-section NNs are (not yet) popular. |

Table 7. A summary of reviewed AxC-enabled architectures by class and publication year.

| Class | Ref. | Main content | Merits | Demerits |
|---|---|---|---|---|
| **GPPs** | [170] 2013 | Quality-programmable ISA and corresponding vector processor micro-architecture supporting dynamic precision scaling. | Fine-grained quality control and monitoring at run-time. | Missing compilation tool flow. |
| | [57] 2015 | Heterogeneous computing platform with approximate/exact neural accelerators and dynamic error control through LWC-based multi-stage acceleration. | Improved power consumption, quality control, and reconfigurability. | Limited NN flexibility, and invocation of multiple approximators. |
| | [104] 2015 | Cache architecture exploiting approximate similarity between cache lines and extra indirection to reduce data storage area. | Significant on-chip area and power reductions. | Complex hashing hardware, and no run-time quality control. |
| | [110] 2015 | Systolic array-based NN accelerator targeting FPGA-equipped SoCs. | Significant power savings and better performance than HLS. | Limited flexibility for NN topologies, and no run-time quality control. |
| | [146] 2015 | Developer-controlled approximate auto-memoization capable GPP architecture with compiler support. | High degree of automation and fine-grained error control. | Great initial adaptation effort and hardware overheads. |
| | [61] 2016 | Significance-aware memoization scheme based on regressions and dynamically truncated ternary CAM lookups. | Significant speedup with fine-grained quality control. | Great hardware complexity of ternary CAM. |
| | [27] 2017 | Approximate memoization-extended Floating-Point Unit of a RISC-style GPP with custom instruction-based control. | Significant speedup and dynamic approximation control. | Missing information on insertion of `SXL` instructions. |
| | [201] 2017 | Three schemes for fine-grained data management in DRAM for trading off performance and energy-efficiency for errors. | Significant speedup and energy savings. | Requires both OS and hardware support. |
| | [161] 2018 | Multi-class classifier and multiple approximator architecture for NN-based approximation with corresponding training algorithm and hardware accelerator. | Increased number of invocations, speedup and energy reduction. | No quality guarantee. |
| | [113] 2019 | Energy-error trade-off exploration with truncated integer arithmetic/logic and load/store units in a RISC-V core. | Easy evaluation of AxC in GPPs. | Limited potential energy savings from integer arithmetic. |
| | [117] 2019 | Relaxed coherence requirements on shared approximable data with an approximate store instruction in many-core GPPs. | Improved application run-time and energy consumption. | High hardware overhead from extra cache. |
| | [118] 2019 | Selectively disabling roll-back on branch and load value mispredictions in speculative GPPs based on sensitivity analysis. | Extensive sensitivity analysis without developer interference. | No quality guarantee from Bayesian analysis. |
| | [82] 2021 | ISA and micro-architecture extension to detect value similarity and subsequently skip entire instruction sequences in GPPs. | Significant speedup and dynamic approximation control. | Difficult to adapt to out-of-order cores. |

Continued on the next page

| | | | | |
|---|---|---|---|---|
| *cont.* CGRAs | [33] 2013 | Dynamic effort scaling-based processor for RMS applications with error resiliency estimation flow. | Reduced energy consumption, quality control and reconfigurability. | Limited applicability beyond RMS kernels. |
| | [2] 2018 | Quality-scalable CGRA enabling run-time adaptation of hardware approximations to application quality constraints. | Improved power consumption and adjustable quality constraints. | Limited quality controllability due to static AxC implementation. |
| | [41] 2020 | CGRA with run-time adaptation of hardware approximations to quality constraints from predetermined operating points. | Reduced power consumption with dynamic adaptation. | Limited applicability due to *simple* PEs. |
| NoCs | [15] 2018 | Developer-controlled, variable transmission power-based approximate wireless NoC enabling unreliable loads and stores to approximable data structures in multicores. | Significant power savings with little impact on application output quality. | Manual annotation of approximable data structures. |
| | [49] 2019 | Approximate wired/wireless NoC architecture with software-exposed broadcast memories enabling fast synchronization and data sharing. | Significant speedup, reduced energy, and low-effort adaptation. | Hardware overhead (particularly in broadcast memories). |
| | [111] 2019 | Integer-value coding schemes based on swap and inversion techniques and extended by crosstalk-avoidance coding for approximate wired on-chip interconnects. | Reduced mean squared error of received data. | Hardware overheads and brute-force crosstalk-avoidance coding search. |
| | [139] 2019 | Truncation-based approximate network interface for wired NoCs. | Simple hardware extension. | No cumulative error control. |
| | [185] 2020 | Dynamic quality control in approximate NoCs supporting packet dropping and compression/decompression. | Reduced energy consumption and guaranteed error constraints. | Relies on developer annotation and compile-time analysis. |
| | [107] 2022 | Truncation- and VOS-based network interface and router for wired NoCs. | Improved latency and energy efficiency. | No quantitative error evaluation. |

Table 8. A summary of reviewed applications of AxC by class and publication year.

| Class | Ref. | Main content | Merits | Demerits |
|---|---|---|---|---|
| ML | [81] 2014 | Stereo image matching accelerator with ANT mitigating approximate additions and voltage over-scaling. | Reduced voltage and power consumption. | Significant hardware overhead. |
| | [174] 2016 | Liquid state machine hardware accelerator for FPGA with activity-based power gating and quality-configurable adders. | Reduced power consumption. | No SNN topology flexibility. |
| | [65] 2017 | Exploration of efficient digital, memristive, and analog memory architectures for hyperdimensional computing. | Analog memory revealed as the most efficient architecture. | **N/A** |
| | [80] 2017 | Approximate hardware accelerator for MLPs with on-chip training, synapse skipping, and dynamic precision control. | Reduced power consumption with on-chip training. | No run-time inference details. |
| | [85] 2017 | Approximate hardware accelerator for NN with stochastic neurons. | Greatly reduced area at negligible accuracy degradation. | Inflexible hardware implementation with physical neurons. |
| | [151] 2017 | SNN framework and hardware accelerator with activity-based synapse skipping and early termination. | Reduced energy consumption with run-time quality control. | **N/A** |
| | [168] 2017 | Hardware accelerator for linear and polynomial kernel SVM models with approximate multipliers. | Reduced area and power consumption. | No run-time reconfigurability and quality control. |
| | [178] 2017 | Design flow and reconfigurable hardware architecture for quality-trade-off-aware CNN acceleration. | Dynamic quality (or performance/energy) trade-off. | Software-level approximations only. |
| | [203] 2017 | Hardware accelerator for Gaussian kernel SVM models with approximate adders and multipliers. | Reduced area and power consumption | No run-time reconfigurability and quality control. |
| | [90] 2018 | Mixed-signal approximate hardware accelerator for hybrid BWN and LSTM networks for speech recognition. | Improved power efficiency. | Limited benefits beyond processing scaling. |
| | [145] 2018 | Comprehensive study of effects arising from applying cross-layer approximation techniques to NN inference. | Significant power savings with little impact on NN accuracy. | No reconfigurability at run-time. |
| | [165] 2018 | NN accelerator with scheduler-supported dataflow reconfiguration at run-time and corresponding performance model. | Significant speedup and increased power efficiency. | No quality control. |
| | [50] 2019 | Algoritmic and hardware approximations for low-power SVM acceleration. | Vastly reduced area and energy consumption. | Unclear implementation details. |
| | [56] 2019 | Hardware accelerator for CNNs with approximate adders and multipliers. | Improved throughput and energy efficiency. | Complex design space exploration and mapping flows. |
| | [92] 2019 | Mixed-signal hardware accelerator for DNN-based VAD with dynamic alphabet-set multipliers. | Improved power efficiency with run-time quality reconfigurability. | Low throughput. |
| | [93] 2019 | Hardware accelerator for KWS based on quantized CNNs and voltage-based approximate multiplication. | Reduced word error rate. | **N/A** |
| | [24] 2020 | Approximate `im2col` architecture for IoT edge CNN acceleration. | Speedup with little quality degradation. | Significant area overheads. |
| | [89] 2020 | Hardware accelerator for BWN-based VAD with dynamic SNR-based arithmetic precision scaling. | Improved classification accuracy. | **N/A** |
| | [86] 2020 | Hardware accelerator for BWN-based KWS with delay-based approximate addition. | Improved resilience toward noise. | **N/A** |
| | [91] 2020 | Hardware accelerator using CNNs quantized into BWNs and approximate self-adaptive adders. | Reduced power per operation. | **N/A** |
| | [69] 2020 | Hardware accelerator for LSTMs using similarity-based cell skipping/approximation and sparsity-aware memory accesses. | Number of MACs and memory accesses halved. | Works only for bi-directional LSTMs. |
| | [75] 2020 | Hardware accelerator for ensemble-based anomaly detection with on-chip training and quantized approximate evaluation. | Improved energy efficiency with run-time adaptability. | No throughput comparisons with related works. |

Continued on the next page

| | | | | |
|---|---|---|---|---|
| *cont.* | [78] 2020 | Hardware accelerator for hyper-dimensional classification tasks with approximate encoding and accumulation. | Significant speedup at reduced resource and energy consumption. | No run-time reconfigurability. |
| | [171] 2020 | Cross-layer approximation-aware DNN design flow, hardware accelerator, and model-to-hardware mapping. | Run-time reconfigurability and near-optimal resource utilization. | No system-wide evaluation results. |
| | [30] 2021 | Technique combining contrast reduction and selective truncation of pixels in image classification applications on NoCs. | Overall speedup, reduced NoC latency and power. | Narrow scope of application. |
| | [58] 2021 | Approximate multipliers for mitigating the effectiveness of adversarial attacks on CNNs. | Reduced transferability, resource and power consumption. | No guaranteed protection. |
| | [84] 2021 | Custom encoding and hardware implementation of binary-operand dot products for CNN acceleration. | Increased operating frequency and throughput. | No run-time reconfigurability. |
| | [137] 2021 | Approximate radix-4 multiplier and GEMM-style accelerator architecture targeting FPGAs. | High throughput, reduced resource and power consumptions. | Needs weight re-ordering. |
| | [182] 2021 | Accelerator architecture for NNs combining matrix-matrix and linear approximation functionalities. | Reduced area and power consumption. | Large quality degradation in some configurations. |
| | [94] 2022 | Approximate SIMD-like multiplier for CNN acceleration with approximate partial product generation and reduction. | Reduced area, power, and delay. | **N/A** |
| | [87] 2022 | Hardware accelerator using CNNs quantized into BWNs and approximate dual-voltage capable adders (extends [91]). | Reduced power per operation. | **N/A** |
| | [179] 2022 | Accelerator architecture for transformer NNs with approximate arithmetic, sparsity speculation, and out-of-order scheduling. | Greatly improved energy efficiency. | **N/A** |
| | [190] 2022 | Approximate maskable input encoding for CNN accelerators. | Low-overhead and non-intrusive technique. | Limited benefits without significant accuracy degradation. |
| **Image processing** | [37] 2016 | Canny edge detection accelerator with approximate Gaussian and gradient filters. | Reduced area and energy per image. | No run-time reconfigurability. |
| | [159] 2016 | ILP-based optimization of a Loeffler-style DCT implementation with approximate adders and multipliers. | Reduced power consumption and area with bounded error variance. | Only arithmetic is approximated. |
| | [9] 2017 | Framework for error analysis of multiplier-less approximate adder-based DCT implementations. | Shows clear approximation potential. | All additions are subject to the same approximation. |
| | [180] 2018 | CGRA-like accelerator for MLP-based approximation of the DCT algorithm. | Significantly reduced latency and energy-delay product metrics. | Increased area and energy per pixel. |
| | [186] 2018 | DCT implementation with approximate weights, thresholding of intermediate values, and inexact adders. | Improved output quality at low energy consumption. | **N/A** |
| | [158] 2020 | Image scaling accelerator based on bi-linear interpolation and approximate edge detection and sharpening filters. | **N/A** | No run-time reconfigurability. |
| | [160] 2020 | Canny edge detection accelerator with approximate Gaussian filters, gradient filters, and gradient direction and magnitude (like [37]). | Reduced area and energy per image. | No run-time reconfigurability. |
| | [167] 2020 | Memory-free, most-recent-result reuse technique applied to Kirsch edge detection. | **N/A** | Limited benefits due to hardware overheads. |
| | [108] 2022 | Canny edge detection accelerator with reduced-size Gaussian and gradient filters. | Large area and power savings. | Substantially worsened output quality. |
| | [195] 2022 | Bilateral filter accelerator for image denoising with approximated filter weights and inexact division. | Large area savings with negligible output quality degradation. | **N/A** |
| **Video processing** | [149] 2014 | Approximate hardware accelerator for Cholesky decomposition of sparse linear systems, useful in video coding. | Great throughput improvements. | **N/A** |
| | [135] 2015 | Extended cache hierarchy for approximately reusing chrominance data for image and video coding. | Reduced number of off-chip memory accesses. | Intrusive technique with unclear hardware overhead. |
| | [44] 2017 | Heterogeneous SAD accelerator with different approximation tiles and power-gating. | Power savings from run-time reconfigurability. | All additions are subject to the same approximation. |
| | [147] 2017 | Reconfigurable fractional pixel interpolation circuit with approximate filters and clock-gating targeting FPGAs. | Reduced power and energy with run-time reconfigurability. | Unclear approximation error impact. |
| | [125] 2019 | Application of approximate additions/subtractions in an optimized SAD accelerator. | **N/A** | No reconfigurability, insignificant power and area savings. |
| | [133] 2020 | Reconfigurable approximate adder-based accelerator architecture for fractional pixel interpolation targeting ASICs. | Reduced power consumption with run-time reconfigurability. | Area overhead. |
| | [126] 2021 | Exploration of various AxC techniques applicable to HEVC decoding. | Potentially great savings. | Some techniques require purpose-built hardware. |
| **Reliability** | [28] 2017 | Two approximate voting schemes for DMR and TMR systems. | Reduced power consumption and increased fault tolerance. | **N/A** |
| | [140] 2019 | Approximate TMR scheme based on precision-scaling of converted fixed-point modules. | Reduced area and increased reliability. | Only considers precision-scaling. |
| | [172] 2019 | Skipping crashes in non-critical code regions instead of restarting. | Great run-time benefits with limited hardware overhead. | Requires both OS and hardware support. |
| | [12] 2021 | Approximate TMR scheme using probability analysis-based over- and under-approximated modules. | Reduced detection energy and algorithmic complexity. | Single fault detection not guaranteed. |
| | [39] 2021 | An approximate QMR scheme based on output subsetting and majority voting. | Potentially improved multiple fault resilience. | Potentially increased area overhead. |

Continued on the next page

| | | | | |
|---|---|---|---|---|
| *cont.* | [112] 2021 | Exploiting area savings from approximating DMR systems to implement extra hardware, improving throughput. | Enables trading off accuracy for throughput. | **N/A** |
| | [177] 2021 | A truncated integer data format with built-in fault tolerance. | Significance-relative errors and reduced power consumption. | Unclear encoding of negative numbers. |
| **Other applications** | [152] 2013 | Approximate wireless communication from importance-based data interleaving and non-trivial symbol encoding. | Increased application signal-to-noise ratio with no retransmissions. | Hardware and communication overheads. |
| | [102] 2015 | Distributed arithmetic-based approximate DWT accelerator. | Improved performance at reduced hardware complexity. | **N/A** |
| | [23] 2016 | Approximate algorithm and systolic array-based accelerator for data detection in MIMO-based wireless communication. | Reduced latency and improved SNR. | Convergence not guaranteed. |
| | [162] 2016 | Ultra-low power architecture for always-on motion detection with approximate hybrid memory architecture at NTV. | Improved energy efficiency. | Large hardware overheads from standard cell memory. |
| | [19] 2018 | Parallel and CGRA-equipped computing system for DSP in IoT edge devices. | Run-time quality control and reconfigurability. | **N/A** |
| | [202] 2018 | Optimizations, limitations, and approximations in a successive cancellation decoder for polar codes. | Increased operating frequency and throughput. | **N/A** |
| | [60] 2019 | Evaluation of approximate adders for FFT and IFFT in wireless communication. | Clear approximation potential. | No power analysis. |
| | [184] 2019 | Applying mathematical approximations to reduce hardware complexity of expectation propagation in SCMA systems. | Near-optimal performance with reduced hardware complexity. | **N/A** |
| | [55] 2021 | Optimized approximate architecture for localization and mapping accelerated on FPGA. | Improved throughput and reduced energy per frame. | Increased resource utilization. |
| | [63] 2021 | Evaluation of approximate MAC units in FIR filters used for wireless communication. | Significantly reduced dynamic power consumption. | Degrading performance in multi-channel systems. |
| | [68] 2021 | Improving mixed-criticality system throughput by approximating low-to-medium tasks. | Increased low-criticality task survivability. | Requires OS and hardware support. |
| | [88] 2021 | Low-power Mel-frequency cepstral component accelerator with dual voltage-capable inexact adders and multipliers. | Greatly reduced power consumption. | **N/A** |
| | [40] 2022 | Algorithmic approximations of a fuel estimation algorithm for cars with corresponding hardware architecture. | Significant area savings. | Incorrectly pipelined architecture. |