IEEE Access
Multidisciplinary ⋮ Rapid Review ⋮ Open Access Journal

Received 2 September 2024, accepted 21 September 2024, date of publication 25 September 2024,
date of current version 15 October 2024.

Digital Object Identifier 10.1109/ACCESS.2024.3467375

RESEARCH ARTICLE

Approximate Computing: Concepts, Architectures, Challenges, Applications, and Future Directions

AYAD M. DALLOO[1], AMJAD JALEEL HUMAIDI[2], AMMAR K. AL MHDAWI[3], AND HAMED AL-RAWESHIDY[4], (Senior Member, IEEE)

[1]Department of Communication Engineering, University of Technology, Baghdad 10066, Iraq
[2]Department of Control and Systems Engineering, University of Technology, Baghdad 10066, Iraq
[3]School of Engineering and Sustainable Development, De Montfort University, LE1 9BH Leicester, U.K.
[4]Department of Electronic and Electrical Engineering, Brunel University London, UB8 3PH Uxbridge, U.K.

Corresponding author: Hamed Al-Raweshidy (hamed.al-raweshidy@brunel.ac.uk)

ABSTRACT The unprecedented progress in computational technologies led to a substantial proliferation of artificial intelligence applications, notably in the era of big data and IoT devices. In the face of exponential data growth and complex computations, conventional computing encounters substantial obstacles pertaining to energy efficiency, computational speed, and area. Due to the diminishing advantages of technology scaling and increased demands from computing workloads, novel design techniques are required to increase performance and decrease power consumption. Approximate computing, nowadays considered a promising paradigm, achieves considerable improvements in overhead cost reduction (i.e., energy, area, and latency) at the expense of a modest (i.e., still acceptable) deterioration in application accuracy. Therefore, approximate computing at different levels (Data, Circuit, Architecture, and Software) has been attracted by the research and industrial communities. This paper presents a comprehensive review of the major research areas of different levels of approximate computing by exploring their underlying principles, potential benefits, and associated trade-offs. This is a burgeoning field that seeks to balance computational efficiency with acceptable accuracy. The paper highlights opportunities where these techniques can be effectively applied, such as in applications where perfect accuracy is not a strict requirement. This paper presents assessments of applying approximate computing techniques in various applications, especially machine learning algorithms (ML) and IoT. Furthermore, this review underscores the challenges encountered in implementing approximate computing techniques and highlights potential future research avenues. The anticipation is that this survey will stimulate further discourse and underscore the necessity for continued research and development to fully exploit the potential of approximate computing.

INDEX TERMS Approximate computing, approximate programming language, approximate memory, circuit-level, approximate machine learning, deep learning, approximate logic synthesis, statistical and neuromorphic computing, cross layer and end-to-end approximate computing.

I. INTRODUCTION

Since 1974, Moore's law and Dennard scaling have projected that the transistor would become smaller and the transistor density would double, resulting in a 40% increase in clock rate while the power density remained constant with each generation [1]. As transistors shrink with technological advancements, it becomes more costly for designers and manufacturers to maintain transistors that behave deterministically, even under typical operating conditions. Verifying the correct operation of digital integrated circuits is becoming more and more costly as technology scales down. Both intrinsic (such as varying dopant concentrations) and extrinsic (such as temperature) factors are drastically increasing the variability of transistors and interconnects [2], [3], [4] as well as reducing energy-delay advantages via CMOS scaling. This nondeterministic phenomenon impedes the constant development of technology. According to ITRS and Intel's technical data, at the 8 nm node, the area of dark silicon exceeds 50% of the chip's area [3], [5].

The associate editor coordinating the review of this manuscript and approving it for publication was Hang Shen.

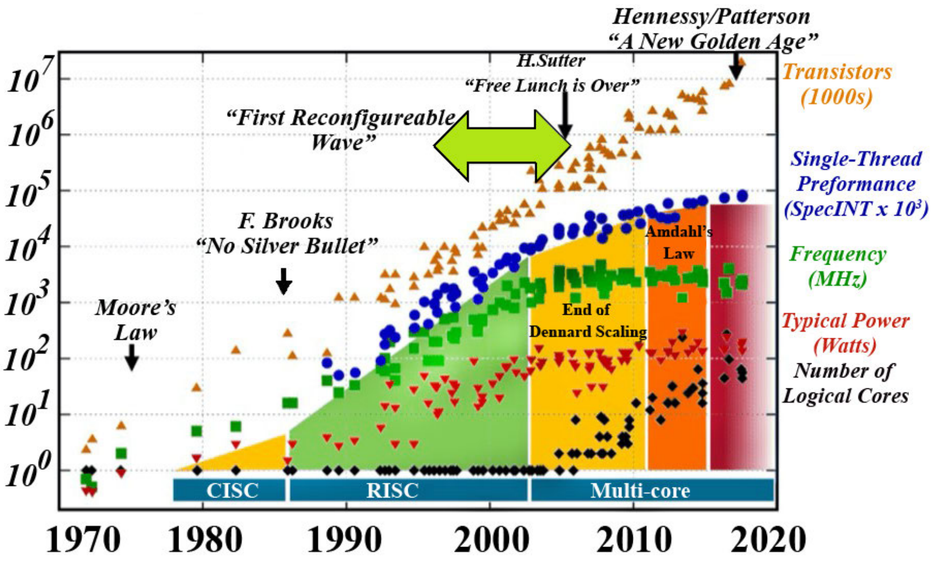


FIGURE 1. 42 years of microprocessor trend data [6], [7].

In 2007, the rate of Dennard scaling slowed dramatically, and by 2012, it had almost stopped completely [6], [7], as shown in Figure 1. Therefore, the scaling of the threshold and supply voltages slowed down due to concerns with leakage currents resulting from increasing on-chip power density. To prevent the chip from overheating, the clock frequency was gradually increased [1], [8], [9]. Due to these limitations and requirements for future applications, novel design techniques are required to handle ever-increasing amounts of data at ever-increasing performance and ever-decreasing power consumption. This is propelling us towards the multi-core era [10]. Despite multi-core system-on-chips (SoCs) aiming to augment throughput while minimizing power consumption, this objective has only been partially achieved due to the inherent challenges in parallelizing certain sequential workloads and existing power constraints [1], [9], As a consequence, the number of active cores is restricted (a phenomenon termed ''dark silicon''), resulting in a gradual scaling-up of cores in contemporary SoCs. As a result, thermal dissipation power (TDP) is a limiting factor for multicore CPUs [1], [11], as shown in Figure 2. Overheating problems were solved by reducing processor clock speeds and powering down unused cores in the ''dark silicon'' era, which followed the TDP constraint [11].

The days of Dennard scaling are over, Amdahl's Law is nearing its end, and keeping up with Moore's Law is becoming difficult and expensive, particularly when the benefits in terms of power and performance begin to diminish [6], [11]. In many computer systems, especially mobile devices, clusters, and server farms, energy efficiency has become a primary design requirement. Saving energy on a mobile phone may lengthen battery life and improve mobility [12]. At nanometer age, the circuits become more sensitive to parameter variations and faults. Reducing the feature size of CMOS technology below 7 nm can lead to deteriorating reliability [13]. This is due to the increased difficulty in controlling and preventing parameter variations and faults at such advanced nanoscales. At these smaller dimensions, physical and quantum effects become more pronounced. All these challenges have changed the dynamics for designing and producing far faster, lower-power circuits and haven't diminished the possibilities for achieving that. For instance, manufacturers implement various techniques such as strained silicon, high-k/metal gates, and FinFET structures to tackle challenges like leakage currents, variability, and other reliability concerns. Furthermore, the percentage of computations for many applications in the runtime represents 83% [14], as shown in Figure 3. These challenges compel both the industry and the academic communities to investigate feasible alternatives and strategies for sustaining the conventional scaling of performance and energy efficiency. In an era marked by the explosive growth of data and the increasing complexity of computations, the traditional methods of computing face significant challenges in terms of energy efficiency, computational speed, and resource utilization.

Approximate computing is one of the promising techniques in this trend that has attracted significant traction from both academic and industry communities [15], [16]. Major corporations such as IBM, Google, Intel, and ARM are actively engaged in pioneering research and the development of commercial offerings that incorporate approximate computing strategies. An illustrative case is Google's Tensor Processing Units (TPUs), which employ an approximate computing technique known as reduced precision to lower energy usage [17]. Parallel paradigms, such as stochastic computing, neuromorphic computing, and quantum computing, have also garnered considerable interest [18]. Table 1 shows a general comparison of these four paradigms, where approximate computing can provide a good balance between latency, accuracy, power consumption, and reliability compared to others. The table compares different computing paradigms, highlighting their trade-offs. Approximate computing is fast but less accurate, while stochastic computing is power-efficient but may not be fast and accurate. Neuromorphic computing excels in power efficiency but might lack in reliability, whereas quantum computing could offer speed and precision but is not yet fully developed. The ideal choice for a computing paradigm depends on the application's specific accuracy needs, which might lead one to choose stochastic computing, whereas power constraints might favor neuromorphic computing. This comparison is critical when selecting a suitable computing approach for a given task.

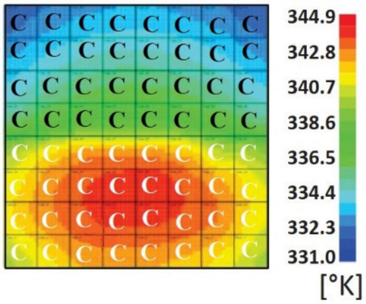


FIGURE 2. An abstract illustration of the Dark Silicon phenomenon which prevents powering-on more cores due to high power density and thermal hotspots, where the white C represents the active cores and the black C represents the idle cores [11].

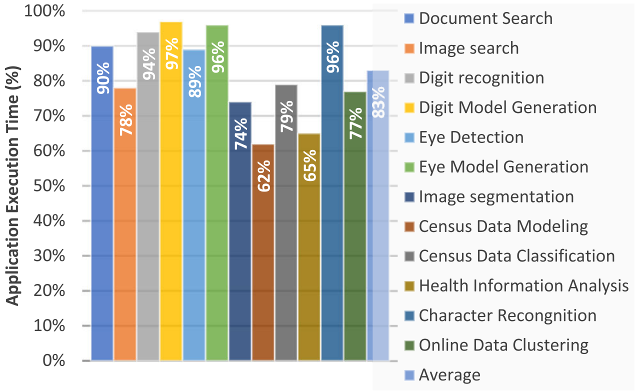


FIGURE 3. Intrinsic application resilience [9], [14].

Approximate computing offers large power and performance improvements in digital systems by relaxing the numerical equality for implementing error-tolerant applications [19]. In approximate computing, error metrics emerge as a novel design parameter that can be traded off to enhance performance or reduce power consumption. Although computational faults are never desirable, applications tolerant to errors confer additional advantages due to their inherent resistance to inaccuracies, attributable to several factors [19], [20]. Firstly, these algorithms handle real-world, noisy input and redundant data, typically output from diverse sensor types. Secondly, they exhibit a probabilistic nature, often evident in iterative algorithms. Lastly, a minor degree of imprecision in their results is generally acceptable, largely due to the limitations of human sensory capabilities.

Typical paradigms of approximate computing applications range from big data to scientific applications, such as image processing, machine learning, and data mining domains. The multifaceted nature of approximate computing results in unique trade-offs. Techniques can be implemented at various levels, from transistor design to software; each approach impacts hardware integrity and output quality in different ways. For example, leveraging acceptable error margins, as high as 10%, in a typical error-resilient image processing algorithm can significantly enhance energy efficiency and computational performance [21]. Another example is that varying memory refresh rates or adopting different data storage and representation precisions are viable strategies to achieve such improvements. However, these techniques might not be suitable for critical applications like medical and military applications [22].

At the heart of approximate computing are four different levels: data, software, architecture, and circuit (hardware). One of its main issues is that the consequences of certain approximations are far-reaching on efficiency and accuracy for different applications; thus, there is no one-size-fits-all solution. This paper will delve into the evaluation of approximate computing at these four levels.

- Data-level: The importance of these techniques cannot be underestimated in the quest for lower power consumption and improved performance. Sampling, quantization, and compression are some of the techniques that allow us to manage quality vs. efficiency issues for smaller or simpler data representations.
- Software Level: There are approximate approaches such as code optimizations like loop perforation, which handle software code to have optimized code, using approximate functions to construct approximate algorithms, or using relaxed synchronization. This aims to show valuable efficiency with a slight degradation in output quality.
- Architecture Level: The approaches at this level for increasing efficiency and saving power are more complicated because we need to rethink the design of specialized approximate processing units and memory systems. Furthermore, it is necessary to expand and enhance instruction sets and use different precisions to contribute to increasing efficiency.
- Circuit level: the approaches are the cornerstone of improved efficiency in power consumption; here we need to rethink to approximate logic gates, optimize transistor behavior, and redesign arithmetic unit circuits, but they come with different degrees of inaccuracy.

This multi-level analysis opens up further exploration of how different approximations combine in real systems. Understanding interactions between levels will guide the

TABLE 1. Comparison of features of the existing computing paradigms.

Computing Approach	Exact	Approximate	Stochastic	Neuromorphic	Quantum
Latency	Low	Low to Medium	High	Low	Low (compared to the complexity of problem)
Accuracy	High	Medium to High	Low to Medium	Medium to High	High (theoretically)
Power Consumption	High	Low to Medium	Low	Low	High (due to cooling requirements)
Reliability	High	Medium to High	Medium	High	Low to Medium (due to qubit instability)
Memory Usage	High	Low to Medium	Low	Medium to High	Low (due to qubit superposition)

development of robust, highly optimized hardware and software designs for approximate computing. However, cross-layer approaches to approximate computing have emerged as a powerful tool for intelligently combining approximation techniques across hardware, architecture, software, and data levels. Through strategic coordination across these layers, researchers aim to maximize efficiency gains while adhering to user-specified quality constraints. These are critical elements for the field's progress. Despite its relative youth, this field demonstrates highly promising results [23], [24], [25]. These early successes underscore the potential of cross-layer techniques to push the boundaries of resource efficiency without sacrificing the functionality of computing systems.

This review tackles the fragmented nature of approximate computing with a comprehensive approach, covering techniques from circuit to architecture levels. It explores how strategies like voltage scaling and selective precision optimize energy efficiency and speed while carefully balancing accuracy, making it ideal for domains like machine learning where slight imprecision is acceptable. We also aim to address the future directions of this promising field, highlighting the potential research avenues and emerging trends. This paper is intended to serve as a primer for researchers and practitioners interested in exploring approximate computing at different levels, providing insights into its potential and limitations.

The subsequent sections of this manuscript unfold in the following manner: Initially, Section II delves into prior surveys to identify the main gaps to be filled by this present survey. Section III presents the scope of this survey and the review methodology. Section IV offers an overview of the general framework for approximate computing. Section V elaborates on the techniques of approximate computing at the data level. Section VI delves into the methodologies employed in approximate computing within the software domain, focusing specifically on the nuances of programming languages designed for approximation. Section VII furnishes an in-depth examination of approximate computing at the architectural stratum, with a particular emphasis on approximate memory systems. Following this, Section VIII elucidates the methodologies of approximate computing at the circuit level, providing detailed insights into their implementation and applications. Section IX provides an overview of frameworks and approaches in approximate logic synthesis. Section X explores three emerging computing frameworks of: cross-layer and comprehensive end-to-end methodologies and statistical and Neuromorphic Computing. Section XI explores the impact of applying approximate computing strategies across diverse applications. Section XII provides the benchmarks, tools, and libraries. Section XIII discusses our perspectives of Future Directions. Section XIV presents the remaining challenges in approximate computing at the different levels, open research questions, and future research directions. Finally, Section XV concludes this review paper.

II. EXITING AND CURRENT SURVEYS

In this section, we delve into an examination of extant literature specifically oriented towards the realm of approximate computing at different levels. As of the writing of this paper, a limited number of surveys probing the domain of approximate computing have been identified. Hence, we have compiled the most significant surveys on approximate computing up to the end of 2023, arranging them in chronological order according to their publication years in Table 2. Additionally, we directed attention to comprehensive surveys that delve into and concentrate on specific subjects within each broader topic, aiding readers in their exploration. Table 2 provides a comparative analysis of various surveys with regards to the topics addressed, namely approximate techniques, applications, hardware and software, and challenges. Furthermore, the table provides an overview of the extent to which each topic was addressed, indicating whether it was fully covered, only partially covered, or not covered at all. The review paper encompasses the number of pages and references cited, as well as the range of years covered. Typically, a review paper should concentrate on the various general aspects pertaining to the implementation of approximate computing.

In the scholarly landscape, there are a modest number of surveys that have embarked on the exploration of approximate computing [19], [26], [27], [28], [29], [30], [31], [32], [33], [34], [35]. These studies, while valuable, primarily offer a cursory overview of the subject, often focusing on specific facets and, consequently, leaving certain aspects underexplored. The granularity of detail and comprehensive understanding of the topic that these surveys provide is, therefore, somewhat limited. Recognizing these gaps in the existing literature, the present survey endeavors to redress these shortcomings. For example, Zervakis et al. [29] concentrated on a limited range of approximate computing techniques. They presented a survey that covers approximate multipliers and approximate high-level synthesis for implementing CNN. Furthermore, they focused on the reconfigurable approximation for neural network inference. Damsgaard and colleagues [33] presented a review paper that touched upon various AxC techniques at the architecture and circuit levels, albeit with brief explanations. Their work distinctively highlights the exploration of approximate wired and wireless network-of-chips, an area frequently neglected in other reviews. Leon and their colleagues [34], [35] presented a two-part review that offers valuable insights and a comprehensive overview of the field.

Recent literature on approximate computing (AxC) has enriched the field with key insights but often lacks the scope and depth our research intends to cover, particularly in exploring the nuances of AxC techniques. This observation underscores the necessity for continued research and discussion to fill these gaps and provide a more detailed exploration of AxC methodologies and applications. However, our review delves deeply into the most approximate techniques and applications, ranging from mobile to cloud computing. Our review expands upon Leon's work by offering a more nuanced analysis, breaking down topics into detailed subtopics, and updating the discourse with research from the last eight years. We also explore areas not covered by Leon, such as approximate elementary and activation functions, the impact on communication and security. Moreover, we providing a broader and more updated perspective in this area by presenting a comprehensive list of influential review papers in the field of approximate computing. We aim to enrich our understanding of this sophisticated domain and lay a robust groundwork for future research by filling the knowledge gaps left by previous surveys to ensure more inclusive coverage of the topic. This review paper covers the important key points as follows: 1) the benefits; 2) techniques; 3) the cases used for each technique; 4) frameworks; 5) hardware circuits and accelerators; 5) programming languages 6) tools, including compilers and logic synthesis; 7) security; and 8) challenges and a future roadmap.

This survey underlines the transformative potential of approximate computing in a variety of domains, particularly machine learning and IoT, and aims to enrich the research community by offering a valuable reference for researchers.

III. SURVEY METHODOLOGY

This literature review provides a solid foundation for understanding the evolution of this field. By tracing developments from early foundational works to the most influential recent publications (2017-2024), we identify key trends and breakthroughs. Publications from established publishers (IEEE, ACM, Elsevier, Nature, Springer, MDPI, etc.) were carefully considered, supplemented by insights from select ArXiv preprints. Five hundred and two studies encompassing various techniques in approximate computing have been examined. The review includes 20 articles from 2024, 93 from 2023, 77 from 2022, 59 from 2021, 57 from 2020, 57 from 2019, 52 from 2018, 36 from 2017 and 61 from the preceding years. The focus of this research was primarily on contemporary literature within the field of Approximate Computing (AxC). An analytical review of selected papers was conducted with several objectives in mind: firstly, to catalog and elucidate the various AxC methodologies; secondly, to enumerate and describe notable AxC architectures that have been documented; thirdly, to discuss the hurdles associated with AxC while proposing feasible solutions; and lastly, to assess how AxC is applied in practice. Figure 4 presents a visual representation of the distribution of selected papers, categorized by their publication years and the publishers involved.

IV. GENERAL FRAMEWORK OF APPROXIMATE COMPUTING

Approximate computing represents a paradigm in computational methodology that willingly sacrifices a degree of precision in exchange for enhanced performance and improved energy efficiency. This strategy proves particularly advantageous for applications that can accommodate a certain measure of inaccuracy without a significant impact on the overall outcome. As delineated in Figure 5, the framework for approximate computing encompasses a multitude of stages and components.

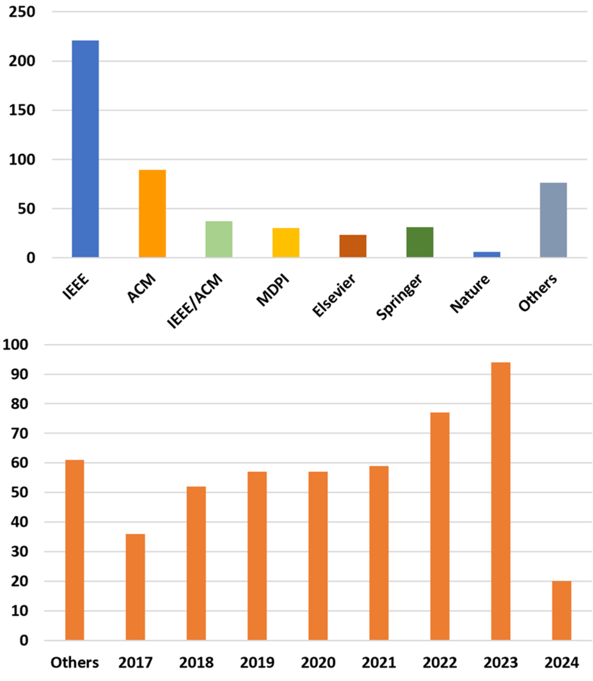


FIGURE 4. The distribution of selected papers, categorized by their publication years and the publishers involved.

The overarching structure of the approximate computing framework is primarily composed of three integral components: the selection of error-tolerant applications, the implementation of approximate-aware design at compile-time (offline), and the execution of approximate tuning at run-time (online). The framework is segmented into finer

TABLE 2. Comparative analysis of approximate computing surveys across the full computing stack.

References	Ours	[19]	[26]	[27]	[28]	[29]	[30]	[32]	[33]	[34], [35]
Year of publication	2024	2016	2016	2017	2018	2021	2022	2023	2023	2023
Pages	70	10	33	34	32	8	10	22	49	69
References	513	59	82	70	183	38	83	62	186	717
Range of Years	2017-2024	-	-	-	-	-	-	-	-	-
Data-level Approximation	✓	×	~	×	~	~	~	~	~	~
Approx. Data	✓	×	✓	×	✓	~	✓	~	×	✓
Approx. structure	✓	×	×	×	×	×	×	×	×	×
Software level	✓	✓	✓	×	✓	×	~	~	~	✓
Code Optimization	✓	✓	✓	×	✓	×	×	~	×	✓
Approximate Compiler	✓	✓	✓	×	×	×	×	×	×	×
Algorithm Approx.	✓	✓	✓	×	×	×	✓	×	✓	✓
Approx. Parallelism	✓	×	×	×	×	×	×	×	×	✓
Relaxed Synchronization	✓	×	×	×	×	×	✓	×	×	✓
Programming Frameworks and Tools	✓	✓	✓	×	×	×	×	×	×	✓
Architectural level	✓	✓	✓	×	~	×	~	~	✓	~
Approximate Memory	✓	✓	✓	×	~	×	~	✓	~	✓
Approximate processors	✓	✓	✓	×	✓	×	×	×	✓	✓
Energy-Memory Management	✓	×	✓	×	~	×	×	×	~	~
Circuit Level	✓	✓	✓	✓	~	~	×	✓	✓	✓
Approximate Adders	✓	~	~	✓	~	×	×	✓	~	✓
Transistor Level	✓	~	~	✓	×	×	×	~	×	×
Gate Level	✓	~	~	✓	×	×	×	~	~	✓
Approximate Multiplier	✓	×	~	✓	~	✓	×	~	~	✓
Approximate divider	✓	×	×	✓	×	×	×	~	×	✓
Elementary and Activation Functions	✓	×	×	×	×	×	×	×	~	×
Approximate Logic Synthesis	✓	✓	×	×	×	~	×	×	×	✓
Application Level	✓	×	✓	✓	~	~	~	~	✓	✓
AI/ML	✓	✓	✓	×	×	✓	✓	✓	✓	✓
IOT	✓	×	×	×	×	×	×	×	×	×
Data mining	✓	×	✓	×	×	×	×	×	✓	✓
Security	✓	×	×	×	✓	×	×	×	×	×
Cross Layer and End-to-End Approx.	✓	×	×	×	×	×	×	×	×	✓
Statistical and Neuromorphic Computing	✓	×	×	×	×	×	×	×	×	×
Benchmarks, Tools and Libraries	✓	×	×	×	×	×	×	×	×	✓
Challenges and Future directions	✓	✓	✓	×	✓	✓	✓	✓	✓	✓

✓ discussed, × not discussed, ~ shortly discussed, - undefined years

elements, each vital for the effective deployment of approximate computing. These subdivisions collectively contribute to enhancing the computational performance and energy efficiency of the overall system. The process begins by choosing one or more approximate levels (employing a cross-layer approach) to implement an application. To successfully leverage approximate computing, a critical first step is to identify the non-critical computation units of the application, which allow for relaxation in accuracy without degrading the overall output quality. This step is called ''non-critical unit identification,'' which requires thoughtful analysis. Once these units are identified, the next step is ''Approximate Design'', which is performed both at compile-time (offline) and runtime (online). Compile-time Approximate design transforms these application units by strategically introducing approximate computations. This process can require specialized tools and compilers to optimize the trade-off between accuracy and efficiency. To maximize efficiency while maintaining accuracy, systems must reconfigure themselves at runtime to change the degree of approximations. This process involves the following steps: monitor conditions, readjust approximation levels, and continuously assess adherence to system goals. Intelligent runtime management is crucial for realizing the full potential of approximate computing. Developing systems that can autonomously and rapidly select the optimal degree of approximation in response to fluctuating requirements and conditions remains a complex and active area of research.

To achieve approximate computation, researchers and practitioners employ a toolbox of diverse techniques. These span hardware components (approximate adders), software frameworks, system-level strategies (sampling), and programming language and logic synthesis features. However, the error analysis and quality evaluation help us in the selection and dynamic adjustment of these techniques. The approximate computing framework reflects a comprehensive strategy. The process begins to help us define and identify the candidate parts or units of the application to apply a suitable approximate technique. After integrating approximated units or parts into the application's design, the compilation and error analysis phases begin. Another critical aspect of this framework is the runtime management of the application through an ongoing assessment of the quality of the results. This methodology facilitates more efficient computing, especially in scenarios where some inaccuracy is acceptable. The next subsequent sections will delve into each component in detail.

V. DATA-LEVEL APPROXIMATIONS

A. APPROXIMATE DATA TYPES AND STRUCTURES

One straightforward way to incorporate approximation into hardware and software is to use approximate data types and structures. To save computing resources, data types and structures allow for certain imprecision in storage and manipulation. For example, we know that precision scaling (e.g., using fixed-point) can accelerate computations and reduce storage. Likewise, approximate data structures (such as Bloom filters or Count-Min sketches) are also useful to save more resources and time by providing probabilistic functionality. Furthermore, approximate data representation focuses on approximating the input data to allow for more efficient computation. Unfortunately, the applications in image processing and neural networks offer a certain inherent level of error tolerance and this provide us the opportunities for concrete enhancements in both performance and energy efficiency. This section delves into the specifics of approximate data types and structures and discusses their implementation, benefits, and potential drawbacks.

1) APPROXIMATE DATA REPRESENTATION

Approximate data representation involves strategically employing techniques like sampling or simplified representations to reduce the complexity or volume of datasets. These methods see wide adoption in domains such as data analysis, machine learning, and other computationally demanding fields. Approximate data types prove advantageous in three key scenarios:

- Resource Constraints: When hardware limitations (memory, storage) are present, data-level approximations enable operation on datasets that would be otherwise infeasible, trading some precision for efficiency gains.
- Real-Time Processing: In streaming or sensor data scenarios, approximate techniques allow for rapid insight extraction and decision-making, prioritizing responsiveness over exhaustive analysis.
- Inherent Imprecision: Many real-world datasets (e.g., weather data, image data) contain natural variability. In these cases, absolute accuracy may be less critical, justifying the benefits of approximate representations.

This makes approximate methodologies suitable for effective data handling, as the natural variability and uncertainty in data sources make exact precision less critical.

a: DATA SAMPLING

One common technique for approximate data representation is data sampling [36], [37], [38], [39], [40], [41], [42], [43], [44], [45]. Instead of analyzing the entire dataset, a representative subset of the data with error bounds is selected for applications such as database search, stream analysis, and model training. This can reduce the computational complexity of the analysis and speed up the processing time. The selection of data can be done based on various criteria such as random, systematic, adaptive, stratified, multistage (clustering), reservoir Sampling, Sampling-Over-Joins, Bucketing Strategy, Coreset etc. The inclusion of sampling operators in leading database products (e.g., Oracle, Microsoft SQL Server, IBM Db2) highlights their importance in extracting insights from large datasets. This capability proves crucial in many areas, including exploratory analysis, predictive modeling, and hypothesis testing. The most basic approach to random sampling is known as uniform random sampling, in which every item in the full data set (also referred to as the ‘‘population’’) has an equal chance of being selected. Despite its simplicity, uniform random sampling can potentially result in significant variability in the resulting estimates.

One strategy for overcoming this obstacle is to provide the developer with abstractions to identify, reduce, and reshape resilient and best-effort computations to be more parallelizable or run on unstable hardware components [9], [46]. There are many frameworks that provide the developer with these abstractions and the capabilities of distributed computing and data processing, such as Hadoop MapReduce [47], ApproxHadoop [48], Apache Spark, Apache Flink, Apache Storm, Apache Tez, Apache Beam, etc. For example, Apache Beam, an open-source framework, simplifies batch and stream processing with its high-level API, compatible with various execution engines like Apache Flink, Spark, and Google Cloud Dataflow. Initiated by Google and developed with partners such as Cloudera and PayPal, it transitioned from Google Cloud Dataflow in 2014 to Apache Beam in 2016 under the Apache Software Foundation.

Data sampling is a technique used in various frameworks to improve the efficiency and speed of processing large datasets, especially in decision-making and analytical applications. In this paper, we provide an overview of how some frameworks utilize data sampling. Laptev [47] proposed enhancing Hadoop with statistics-based uniform sampling for efficient analysis of massive datasets, addressing time and resource limits. This extension, EARL on Hadoop, accelerates processing when preliminary results suffice, maintaining high accuracy with small samples and using bootstrapping for accuracy estimates. Goiri et al. [48] introduced an approximate Hadoop version using strategies like data sampling and task dropping for large datasets. This approach, allowing for both precise and approximate MapReduce operations, can significantly cut runtimes by up to 32 times with a tolerable error margin of 1% at 95% confidence. Hu et al. [49] explored sampling as a way to speed up decision-making queries on large data sets by introducing a sampling framework in Spark that allows for approximate computing with error estimates. ApproxSpark supports various sampling methods, such as partition versus data item sampling and

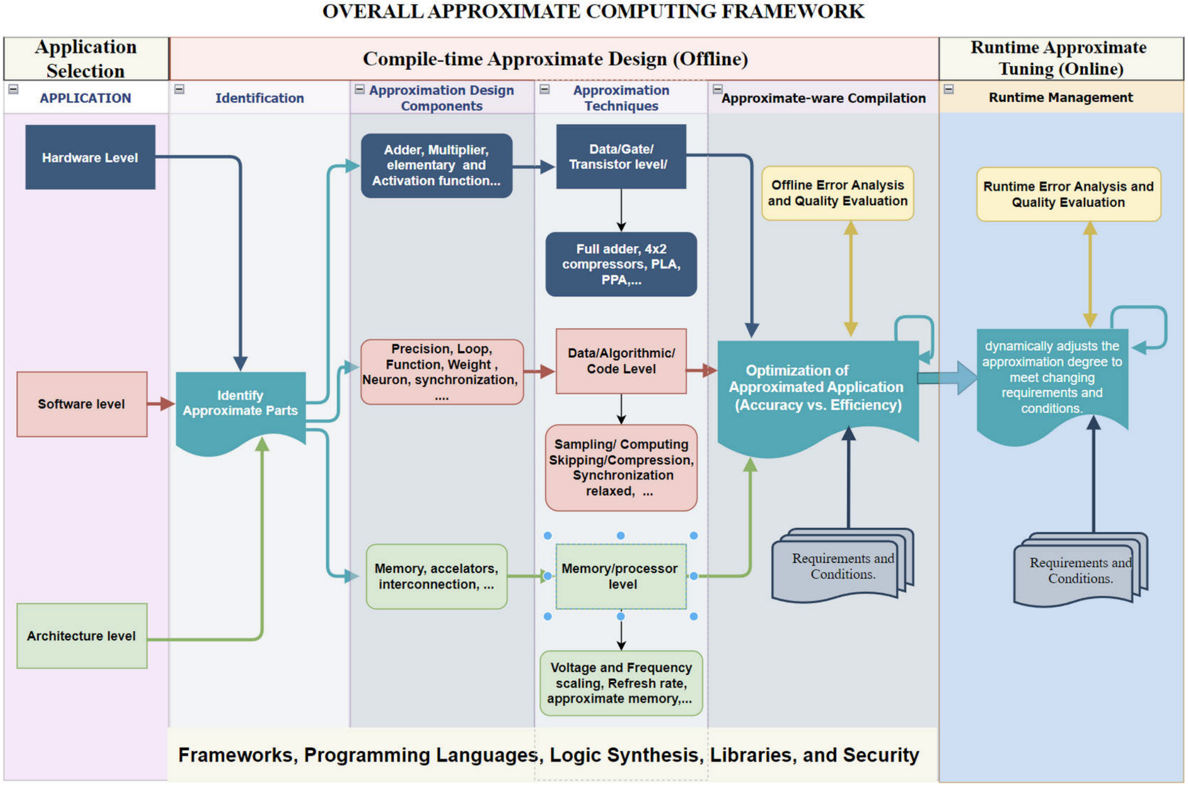

OVERALL APPROXIMATE COMPUTING FRAMEWORK
Application Selection
Compile-time Approximate Design (Offline)
Runtime Approximate Tuning (Online)
APPLICATION
Identification
Approximation Design Components
Approximation Techniques
Approximate-ware Compilation
Runtime Management
Hardware Level
Software level
Architecture level
Identify Approximate Parts
Adder, Multiplier, elementary and Activation function...
Precision, Loop, Function, Weight , Neuron, synchronization, ...
Memory, accelators, interconnection, ...
Data/Gate/ Transistor level/
Full adder, 4x2 compressors, PLA, PPA,...
Data/Algorithmic/ Code Level
Sampling/ Computing Skipping/Compression, Synchronization relaxed, ...
Memory/processor level
Voltage and Frequency scaling, Refresh rate, approximate memory,...
Offline Error Analysis and Quality Evaluation
Optimization of Approximated Application (Accuracy vs. Efficiency)
Requirements and Conditions.
Runtime Error Analysis and Quality Evaluation
dynamically adjusts the approximation degree to meet changing requirements and conditions.
Requirements and Conditions.
Frameworks, Programming Languages, Logic Synthesis, Libraries, and Security

FIGURE 5. Overall framework of approximate computing.

stratified sampling, to provide fast results with estimated error bounds. The findings indicate that ApproxSpark can notably enhance speed while retaining accuracy to optimize for different applications.

Sampling techniques play a crucial role in addressing the challenges of stream analytics. Quoc et al. [50] developed StreamApprox, an approximate computing system for stream analytics that provides significant speedups and throughput gains (1.15x−3x) over native Spark Streaming and Flink. This is achieved through selective sampling, while still maintaining high accuracy levels. StreamApprox outperforms a competing Spark-based sampling system with comparable accuracy. Wen et al. [51] proposed a system called ApproxIoT that employed approximate analytics for high throughput edge computing. The authors used online hierarchical stratified reservoir sampling to gather data in a decentralized manner, but the aforementioned systems [50] are designed to handle the task of processing input data streams in a centralized datacenter. The authors also employed an extended stratified reservoir sampling to select data from multiple sub-streams, ensuring no individual sub-stream is ignored. It generates approximate output with defined error bounds, making effective use of edge computing resources. ApproxIoT surpassed traditional sampling with 1.3x to 9.9x faster processing across 10%-80% sampling rates, showing slight accuracy decreases (0.07% at 10% sampling). In tests with NYC taxi data, it offered improved data throughput, balancing efficiency and quality. However, it's limited to linear queries and needs manual sampling adjustments. Nguyen et al. [37] introduced S-VOILA, a stratified random sampling algorithm designed for efficient and representative data stream handling. The algorithm was evaluated using real-world datasets, including the OpenAQ dataset, and compared with other methods such as Reservoir, ASRS, and Senate sampling. It achieves a lower variance than ASRS and approximates VOILA allocation. Empirical results on real-world data demonstrate its superiority over Neyman allocation. This makes S-VOILA valuable for reducing computational overhead in machine learning model training. Park et al. [52] developed BlinkML, a system that enables error-sample size trade-offs for machine learning training, efficiently estimating the needed sample size for desired accuracies. BlinkML outshone traditional methods by training 961 models in 30 minutes and finding the best model in 6 minutes, but the traditional methods failed within an hour. It achieved up to 95% accuracy in various models, using only 0.16% to 15.96% of the usual training time, and employed uniform random sampling for large datasets with a memory-efficient approach. Anderson and Cafarella [53] focused on optimizing the time-consuming process of feature engineering in machine learning. They proposed a system, ZOMBIE, that treats feature evaluation as a query optimization problem, thus accelerating the feature evaluation loop. They employed a variation of active learning for data sampling. The system was tested using different learning tasks and index group creation methods, and the results showed that ZOMBIE significantly outperformed conventional methods, reaching the accuracy plateau for a task nearly eight times faster. The authors conclude that ZOMBIE can reduce engineer wait times from 8 to 5 hours in some settings.

Sampling frameworks offer compelling advantages when dealing with massive datasets. By intelligently reducing the volume of processed data, they lead to faster execution times and improved scalability. Current research is investigating improvements in sample techniques to reduce errors and customize them for certain analytical purposes.

Researchers commonly approximate data at the software and hardware levels using three approaches: precision scaling, quantization, and relaxed precision. These techniques can reduce the complexity of computational applications.

b: RELAXING PRECISION

The design methodology of approximate computing involves sacrificing computational precision in exchange for enhanced power efficiency and performance. A prevalent approach is the relaxation of precision, which involves reducing the bit count employed in representing data or performing computations. However, the compromise lies in the potential occurrence of errors or imprecisions in calculations. Error-tolerant algorithms, error compensation techniques, and error-aware design can be used to alleviate the deleterious effects of precision relaxation. The appropriateness of relaxing precision is contingent upon the application's capacity to accommodate errors, and a judicious evaluation is necessary to achieve equilibrium between the advantages of minimizing precision and the requisite degree of exactness for a particular application. This involves reducing the precision of numerical calculations, such as using single-precision floating-point numbers instead of double-precision. The floating-point data type is a common target for approximation. By reducing the precision of floating-point numbers, computations can be performed more quickly and with less energy. This can significantly reduce the computational cost of the algorithm at the expense of reduced accuracy. Carmichael et al. [54] explore low-precision numeric formats (fixed-point, floating-point, and posit) at ≤8-bit precision for use in DNN accelerators. Static analysis tools [55] play a key role in enabling such precision reduction techniques.

c: QUANTIZATION

This technique refers to the process of reducing the precision of numerical data in a program by mapping the values to a smaller set of discrete values. This is typically done in machine learning models to reduce the memory requirements and computation costs of the model, which is especially important for deployment on edge devices with limited resources. As a result, the majority of recent studies on quantization have concentrated on inference [56]. One common method of quantization is fixed-point quantization, where each numerical value is represented as an integer or fixed-point number with a limited number of bits. Quantization can be done during the training or inference of a machine learning model. In post-training quantization, the weights and activations of a pre-trained model are quantized to a lower precision [57], while in quantization-aware training [56], [57], [58], [59], [60], the model is trained with the quantization process in mind, often with the use of special quantization-aware algorithms and techniques. The choice of quantization method and the level of precision to use depend on the specific requirements of the application and the trade-off between accuracy and resource usage. This can save memory and computational resources, but it can also introduce some errors [61]. In the depicted training workflow as shown in Figure 6, Novac et al. [59] employed floating-point quantization to strike a balance between computational efficiency and precision. Prior to performing computations within each neural network layer, the inputs, weights, and biases are quantized to lower precision while retaining their floating-point nature. Post-computation, the outputs are similarly quantized before they proceed to the subsequent layer. This approach ensures a consistent precision level throughout the network's forward pass. The precise methodology for quantization is outlined in [59]. Notably, during Training, certain processes, such as the system dynamically reassess the value range and updates the scale factor before performing layer computations. However, during inference, the scale factor remains fixed. Also, Dai and Fan [62] tackled the challenge of deploying accurate crop disease recognition models onto resource-constrained hardware. Their multi-pronged approach combines pruning, knowledge distillation, and ActNN compression with INT8 quantization. Remarkably, this significantly reduced model size (by 88%) and inference time (by 72%) while achieving an impressive 94.24% accuracy. Their contribution demonstrates the feasibility of accurate real-world image analysis on smaller devices. Real-time ECG analysis at the edge is challenging due to device limitations. Mohammed's work [63] addresses this with a lightweight model that uses quantization and pruning to achieve up to 99.1% and a 95% F1-score for edge-based deployment.

Due to hardware improvements and privacy considerations, machine learning (ML) is moving towards edge devices. Federated learning (FL) shines here, improving privacy and network efficiency. To support this trend, Costa et al. [64] proposed L-SGD, a lightweight version of SGD optimized for microcontrollers (MCUs). Their implementation is 4.2x faster than standard SGD while consuming significantly less memory (2.8%). It boasts both a floating-point and a quantized version for fine-tuning, showing promise for quick model updates and fairness fixes in FL scenarios.

d: PRECISION TUNING OR SCALING

This technique involves adjusting the numerical precision of calculations to improve both accuracy and efficiency.

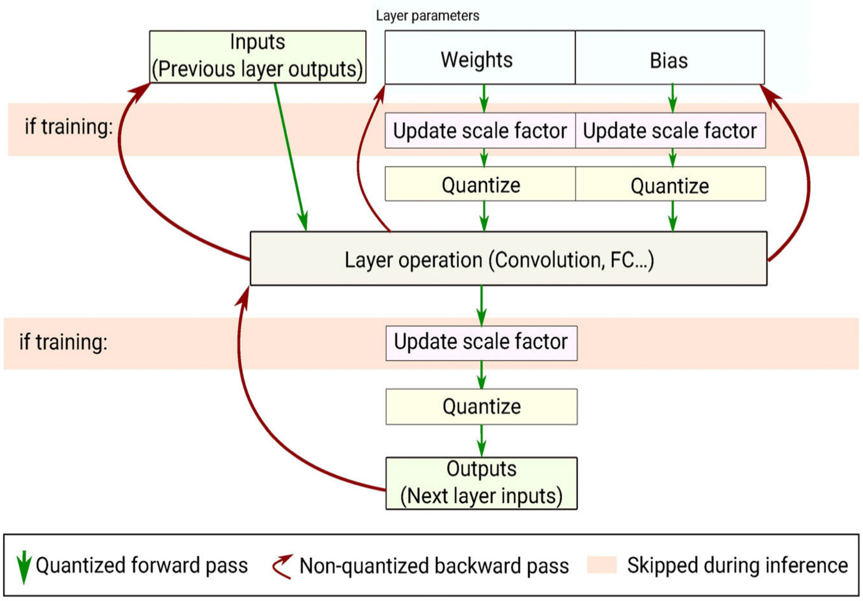


FIGURE 6. **Quantization-Aware training architecture [59].**

It entails fine-tuning data and computations to maximize efficiency and accuracy while using as few resources as possible. Precision scaling or feature scaling approaches (e.g., half-precision (16-bit) and mixed-precision training) are both techniques used in deep learning that aim to improve training efficiency and reduce computational resources while maintaining or even improving model performance [54], [65], [66]. These breakthroughs have become particularly relevant with the advent of powerful hardware accelerators such as GPUs and TPUs, which can effectively leverage the benefits of reduced precision arithmetic. Nevertheless, achieving precision below half-precision has presented a considerable challenge that requires extensive fine-tuning. Numerous cutting-edge software-level approaches [67], [68] have been developed to tackle various challenges associated with precision scaling, including scaling degree, scaling automation, mixed precision, and dynamic scaling.

To handle the complexity and non-intuitive nature of round-off errors in floating-point, Wei-Fan et al. [69] addressed this issue using formal analysis with FPTUNER, an automated tool that optimizes precision through symbolic expansions. FPTUNER efficiently manages precision modifications and was tested on various benchmarks, showing significant energy savings with mixed-precision code despite some compiler-related challenges. For a detailed study on these quantization techniques, the review paper [56] offers extensive insights.

The utilization of graphics processing units (GPUs) has become widespread in accelerating various emerging applications, including but not limited to big data processing and machine learning. Although GPUs have demonstrated their effectiveness, one prevalent approach to enhancing performance is approximate computing, which involves sacrificing accuracy in exchange for improved performance. The technique of approximating high-precision values into lower-precision values with precision scaling has become increasingly popular on GPUs, with support for half-precision at the hardware level. The issue with GPU-side kernel-level scaling is that the overall improvement in program performance is often limited due to the combination of data transfer, type conversion, and kernel execution. To address this issue, several solutions can be employed: optimizing data transfer, kernel fusion [67], adaptive precision techniques [70], memory hierarchy optimization, compiler and runtime support, and advanced code analysis and optimizations. By implementing these solutions, the performance of GPU-side kernel-level scaling can be significantly improved. Kotipalli et al. [70] addressed the limitations of precision selection for applications with strict accuracy requirements, neglect of performance concerns in GPGPU accelerators, and insufficient optimization techniques in existing approaches. It provides a comprehensive solution, AMPT-GA, that optimizes performance while satisfying accuracy requirements in high-performance computing applications. To face the scalability limitations of precision tuning techniques due to the wide search space, Guo and Rubio-González [71] presented a scalable hierarchical search algorithm for precision tuning, which was implemented in the tool HiFPTuner. The results showed the proposed algorithm reduce the search time by 59.6%. compared to the state-of-the-art.

The concept of "dynamic precision scaling" pertains to the modification of numerical precision in real-time, which is contingent upon the particular demands of a given computation or system [68], [72]. Deep neural networks demand extensive linear operations, impacting speed. Giamougiannis et al. [73] introduce a dynamic-mixed-precision inference scheme to address this problem. Their results show a significant execution time reduction (55%) for linear operations while maintaining model accuracy. Effective mixed-precision tuning demands tailored hardware and software. Fornaciari et al. [74] presented a roadmap for this co-design. Their roadmap, informed by recent advances, strives to maximize mixed-precision benefits (performance and energy efficiency) for diverse applications.

e: COMPRESSION

This technique involves reducing the size of data or files through various compression techniques. The goal is to store or transmit data in a more efficient way, thus reducing storage or bandwidth requirements and potentially improving performance and energy efficiency [53], [54]. There are two main categories of data compression: lossless and lossy. **Lossless compression** is a type of compression that uniquely guarantees the ability to recover the exact original data from its compressed form. Examples of lossless compression techniques include Huffman coding, Run Length Encoding, LZ77, ZIP, GZIP, and RAR, which are used to compress text, images, and other types of data [75]. There are also **lossy compression** algorithms such as JPEG, MP3, and MPEG, which eliminate unnecessary or less important information where a certain amount of data loss will not be detected by most users. These types of compression are used to compress multimedia files like images, audio, and video. For instance, JPEG, MP3, and MPEG-4 are used for images, audio, and video, respectively [75], [76], [77], [78], [79], [80]. Lossy compression formats, such as MP3 or MPEG4, achieve smaller file sizes in comparison to lossless formats, albeit with a trade-off of reduced output fidelity. Data compression plays a dual role in machine learning and big data contexts. Lossy techniques (MP3, MPEG4) aren't the only way to reduce the size of a file. Dimensionality reduction techniques like Principal Component Analysis (PCA) and t-Distributed Stochastic Neighbor Embedding (t-SNE) hold particular significance. These techniques streamline processing by extracting high-level features from vast datasets while potentially mitigating issues like the curse of dimensionality.

Classical dimensionality reduction (PCA, t-SNE) excels at finding linear structure in data but can have difficulties capturing complex, nonlinear relationships that often exist in high-dimensional datasets. Beyond Linear Compression, Autoencoders [81] and generative models (including Variational Autoencoders (VAEs) [82] and Generative Adversarial Networks (GANs) [83]) use deep neural networks inherently adept at nonlinear patterns. These can encode richer, more expressive representations of data. uto-Encoder (AE) [84] is a neural network architecture that specializes in encoding and decoding data. The encoder component compresses the input data x into a condensed representation known as the latent variable z, following the function $q_\phi(z|x)$, as shown in Figure 7 (a). The decoder then attempts to reconstruct the original input from this latent variable, outputting x as per the function $p_\theta(x|z)$. The AE is generally trained without supervision to minimize the reconstruction error between x and $\hat{x}$. Variations of AEs, including Variational Auto-Encoders (VAEs) and their derivatives, extend this basic framework to serve more complex purposes like data generation and denoising, adapting the architecture to a range of applications, as shown in Figure 7(b).

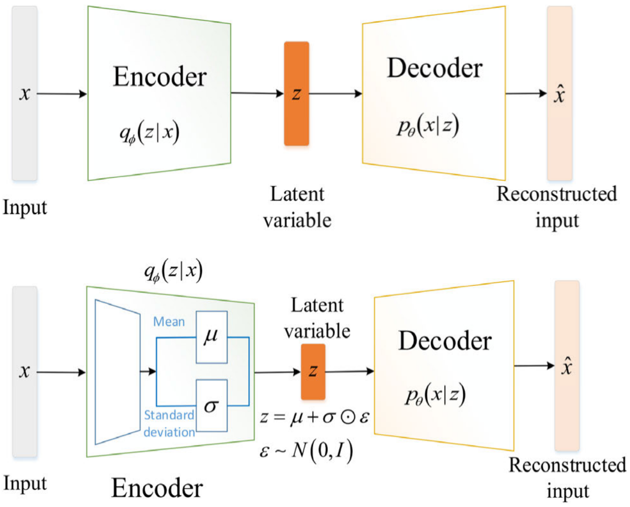


FIGURE 7. Architectures of (a) Autoencoder, and (b) Variational Autoencoders (VAEs) [84].

For example, Duan et al. [82] introduced a Quantization-aware ResNet VAE (QARV) for lossy image compression, combining hierarchical VAEs design with quantization optimizations for efficient entropy coding and fast decoding. QARV is characterized by using variable compression rates, which outperforms existing methods in rate-distortion metrics. However, choices like PCA's number of components or an autoencoder's bottleneck size directly influence information loss.

The choice of data compression technique depends on the specific requirements of the application, such as the need for lossless reconstruction, the acceptable level of data loss, and the computational resources available. Wiedemann et al. [85] introduced DeepCABAC, a novel neural network compression method based on Context-based Adaptive Binary Arithmetic Coder (CABAC), achieving high compression rates without compromising accuracy. They demonstrate that DeepCABAC can compress the VGG16 ImageNet model by a factor of 63.6, reducing the network's memory footprint to a mere 9 MB without compromising its accuracy.

In conclusion, although compression methods provide notable advantages in minimizing data volume and enhancing storage and communication efficiency, they present a set of challenges that need to be addressed. The low performance and high complexity of compression and decompression algorithms can offset the benefits, especially for low-power devices and real-time scenarios (video streaming). The data types and compression ratio also specify the type of compression algorithm to be used. Therefore, being careful when making the decision to select and implement compression techniques is crucial.

2) APPROXIMATE DATA STRUCTURES

Data structures offer a strategic approach to data storage and retrieval, incorporating mechanisms for approximation or lossy compression to curtail memory and computational demands. This efficiency extends to supporting decrement operations and managing negative counts, further enhancing system performance. For example, in data analytics, approximate data structures such as Bloom filters and HyperLogLog can be used to estimate the cardinality of a set without storing all the elements of the set [86], [87]. There are some examples of approximate data structures:

a: BLOOM FILTER

The Bloom filter's core strength lies in its space efficiency and fast membership queries. However, its probabilistic nature introduces the possibility of false positives (indicating an element is present when it isn't actually in the set) [88]. Despite this limitation, Bloom filters find wide adoption in scenarios where some inaccuracy is tolerable and space is a major constraint [89]. They are widely used in various domains such as IOT, networking, databases, and bioinformatics. Burton [90] introduced Bloom filters in the 1970s. There are many categories of Bloom filters based on practical measurements, namely, Standard, Counting, Dynamic, Hierarchical, Loglog, Spectral, Multidimensional, Fingerprint-based, Shifting, Compressed Bloom Filters, etc. In general, designing Bloom filters presents several key challenges: a trade-off between false positive rate and space, no false negative control, optimal hash function choice, predefining size, and scalability. The predefined size of the Bloom Filter, which cannot be changed later, poses challenges for large or growing datasets. The rate of false positives can be reduced by increasing the size of the Bloom filter or using more hash functions. There are many proposed approaches to reduce the rate of false positives. However, both solutions require more computational resources. For applications where false positives are absolutely unacceptable within known data size constraints, EGH filters provide a valuable solution, as demonstrated by Sándor et al. [91]. This has potential implications for areas like network security and data validation. For providing control over false negatives, Bloom filters can handle the deletion of elements, thus providing control over false negatives.

Bloom filter is a little more memory-intensive hashing method. BF's compute cost comes from hash function computation and query judgment. MD5, SHA-1, and other computation-intensive hash algorithms are needed for BF. Perfect and locality-sensitive hashes are considerably harder to compute. Determining the ideal number of hash functions in a Bloom filter depends on several factors: the filter's size, the expected dataset size, and the most importantly, the relative cost of the hash function itself. Modern optimization balances these factors. Bloom filters exhibit either the capability to delete data while incurring supplementary memory usage, or the ability to expand data while incurring a higher rate of false positives and a reduction in query speed. Therefore, Yuhan W. [92] addressed and solved the two shortcomings: no deletion and no expansion, by proposing a new Bloom Filter, called Elastic Bloom Filter.

The classic Bloom filter, while remarkably space-efficient, faces inherent trade-offs between accuracy, query speed, and memory usage. Recent work by Gebretsadik et al. [93] presented the enhanced Bloom filter (eBF), a novel design specifically tailored to the challenges of intrusion detection in IoT networks. Their experimental evaluation reveals the eBF as a significant step forward, demonstrating considerable memory savings (15.6x, 13x, 8x) over standard Bloom filters, Cuckoo filters, and robust BFs, while maintaining fast and accurate performance. Seymen and Yalçın [94] proposed a lightweight Bloom filter for IOT applications and implemented it using the Murmur3 hash on a Nexys A7 FPGA board.

In summary, Bloom filters are celebrated for their compactness and proficiency in membership determination, despite their computational and memory demands [95], [96]. Future efforts will aim at refining these structures to lower false positives, enhance scalability, and conserve computational resources, thereby bolstering their effectiveness and efficiency for expansive datasets.

b: SKETCHING DATA STRUCTURES

This structure is a family of data structures used to summarize large data sets in a small amount of space. They can be used for approximate query answering and data compression [97]. Particularly, sketch-based data structures, such as the traditional Count sketch (CS), Count-Min Sketch (CMS) [7], Count-Mean-Min Sketch (CMMS) [98], and many more, are a frequent technique for frequency estimation. The efficiency and reasonable accuracy make sketch-based approaches compelling for network measurement. However, the ongoing need to balance accuracy with memory constraints creates an active research area. Developing more versatile sketches or techniques for dynamically adjusting sketch parameters is crucial. New sketch data structures, including the Count Min Log Sketch (CMLS) [99], Switch Sketch [100], Elastic Sketch [101], HBL (Heavy-Buffer-Light)-Sketch [102], or Diamond Sketch [103], have emerged in recent years [104]. For example, faced with the challenge of counting item frequencies in huge datasets where exact storage is impossible, the Count-Min Sketch emerges as a powerful solution. With a probabilistic approach, it intelligently trades some accuracy for a significantly reduced memory footprint [105]. It is used in various applications like compressed sensing, networking, databases, NLP, security, machine learning, etc.

One of the challenges of the Count-min sketch algorithm is overestimation of the frequency of events due to hash collisions [106], and to mitigate the issue of overestimation, one could use a variant of the Count-Min Sketch known as the Count-Mean-Min Sketch [98]. The accuracy of the CMS depends on the quality of the hash functions used [107]. The quality of the hash functions can be improved by using independently universal hash families [108]. Khan A. et al. [107] introduced an enhanced approach to sketch-based hashing, diverging from direct full-key hashing. Their methodology involves the use of multiple independent hash functions, each targeting different segments and combinations of a key, thereby establishing a composite hashing framework for improved accuracy. The fact that the accuracy of CMS improves with more space (i.e., more hash functions and larger arrays) is another challenge leading to a trade-off between the two. In addition, it should be noted that CMS lacks support for decrement operations and negative counts. Count-Sketch is a viable alternative to Count-Min-Sketch for accommodating negative counts. CMS can provide frequency estimates, and a combination of data structures could be used to support exact queries; for example, one could use a hash map for exact queries [109] and Count-Min Sketch for frequency estimation. Another challenge is that the CMS data structure cannot be resized once it's created. This issue was addressed by Zhu et al. [110] by proposing a dynamic variant of Count-Min Sketch that allows for resizing, called Dynamic Count-Min Sketch. The Count-Min Sketch is widely used in data stream analysis, network monitoring, database size estimation, and other areas where processing massive amounts of data is required.

In 2018, a team from Tsinghua University and Microsoft Research [91] proposed Elastic Sketch algorithm, which would consume less memory and provide a more precise estimation of item frequencies. It is considered a solution for network-wide measurements, which is a critical function for network management and security. It is designed to adapt to different traffic distributions and measurement tasks. Elastic Sketch outperforms contemporary benchmarks with a speed increase of 44.6 to 45.2-fold and a reduction in error rates ranging from 2.0 to 273.7 times. This algorithm was enhanced by Keyan [102], known as Heavy-Buffer-Light (HBL) sketch. By comparing it to its predecessor, such as the elastic sketch, and other conventional methods, HBL manages to decrease the average relative error rate by 55% to 93% under identical memory constraints.

c: HYPERLOGLOG (HLL)

HLL is a probabilistic data structure that is a very powerful approximate algorithm used for estimating the cardinality of a set. It's particularly useful when dealing with large datasets because it provides acceptable accurate estimation with significantly less memory [80], [87]. It is used in various applications like network monitoring, web analytics, data analysis, and databases. HLL is a probabilistic algorithm that provides approximate estimation, and one can improve the accuracy of HLL [112] by increasing the number of registers used or using high-quality hash functions. There are also improved versions of the algorithm, such as HyperLogLog++ [113], HyperLogLogLog [114], or HLL-Tailcut [115], that offer better accuracy and less memory usage. Unfortunately, HyperLogLog doesn't support the deletion of elements; therefore, the sliding HyperLogLog algorithm [116] was proposed to support deletions. HyperLogLog sketches [117] are proposed to extend the HyperLogLog algorithm to support estimating the cardinalities of union, intersection, or relative complements of two sets. Another issue is that understanding privacy-related attributes of datasets, such as re-identifiability and joinability, is crucial for data governance. However, large datasets and organizations require more efficient strategies, as brute force methods are inefficient due to their massive systems and data volume. Pern et al. [87] introduced an extension of the HyperLogLog algorithm, KHyperLogLog (KHLL), an algorithm based on approximate counting techniques for estimating re-identifiability and joinability risks in large databases. KHLL's joinability analysis helps distinguish between pseudonymous and identified datasets. This leads to reduce reliance on expert judgment and manual reviews. It uses less memory and linear runtime.

d: MINHASH

MinHash is a probabilistic data structure used to estimate the similarity between two sets. It can return approximate answers with high probability [88]. The utilization of MinHash and HyperLogLog sketching algorithms has become an essential practice in the realm of big data applications for the purpose of set summarization. HyperLogLog is a technique that enables the counting of distinct elements using small fraction of storage space. On the other hand, MinHash is a method that is well-suited for rapid set comparison, as it permits the estimation of Jaccard similarity and other related measures. Therefore, Ertl [119] introduced a novel data structure named SetSketch, which effectively bridges the gap between the two aforementioned use cases. In numerous instances, it exhibits superior performance compared to the corresponding state-of-the-art estimators. Also, Yu and Weber [120] introduced a novel compressed sketch known as HyperMinHash, which is based on the HyperLogLog framework and can serve as a seamless substitute for MinHash. The HyperMinHash algorithm preserves the fundamental characteristics of MinHash, including the ability to perform streaming updates, unions, and estimate cardinality.

e: T-DIGEST

T-digest is an algorithm that is used for real-time operations and constructing concise representations of data that are capable of approximating rank-based statistics with a high degree of accuracy, especially in the vicinity of the distribution's extremities [121]. It was introduced by Ted Dunning in 2013. This novel form of sketch exhibits resilience in the face of non-normal distributions, multiple iterations of sampling, and arranged data sets. The integration of independently computed sketches can be achieved with minimal or negligible compromise in precision. The t-digest algorithm is extensively utilized within prominent corporations and is additionally incorporated into commonly used software applications such as Postgres, ElasticSearch, Apache Kylin, and Apache Druid. The t-Digest has the property that the error is smaller around the median and larger at the extremes, which makes it particularly useful for applications that require accurate estimates of quantiles for skewed data [89].

Overall, approximate data structures can be a useful tool for handling large amounts of data efficiently while sacrificing some level of accuracy. Each of those techniques offers unique advantages and can be applied in different scenarios depending on the specific requirements of the application. However, they also have their own challenges and limitations, such as ensuring that the introduced approximations do not significantly degrade output quality or lead to unacceptable errors. The choice among these techniques, therefore, requires a careful understanding of both the application's characteristics and the capabilities of the approximation technique.

VI. SOFTWARE-LEVEL APPROXIMATIONS

Approximate computing is a technique used in computer engineering to reduce the computational complexity and energy consumption of computing systems while relaxing the accuracy of the computations. This approach can be particularly useful for applications where accuracy is not critical or where the computations are too complex or time-consuming to be performed exactly. The complexity of these applications is ever-increasing since they must constantly adapt to provide new services and process a large amount of data. The growing cost of developing such systems, including the target cost, power consumption, execution time, and memory space for software development, is directly proportional to the increasing complexity of systems. The idea behind approximation computing at the software level is to minimize processing complexity, which is represented by the number of processing operations and memory accesses, in order to reduce implementation costs. Therefore, there are many approximate techniques at the software level proposed in the literature in order to reduce the computation and the time-execution of a program by introducing inaccuracies or approximations in certain parts of the computation while producing an acceptable accuracy of results. The task of identifying and selecting computations for approximation that have less influence on the quality of the results is one of the most difficult aspects of approximate computing. Software-level approximation techniques refer to the methods used to simplify the design and analysis of software platforms. These techniques aim to reduce the complexity of software systems while maintaining acceptable levels of performance and functionality. They can be applied at various stages of the software development process, including design, implementation, and testing.

A. CODE OPTIMIZATION-BASED APPROXIMATE METHODS

These methods focus on modifying the code to optimize for approximate computation while maintaining an acceptable level of accuracy [122]. These methods can be applied manually by the programmer or automatically by a compiler or another tool. Approximation-enabled compilers are another important avenue for software-level approximate computing. These compilers introduce approximations into programs automatically or semi-automatically. They analyze the source code to identify parts of the program where approximations can be introduced without significantly affecting the overall output quality. Techniques employed by these compilers include loop perforation (skipping some iterations of a loop), operator approximation (replacing exact operators with approximate ones), and task skipping (skipping some non-critical computations). These techniques modify the compiler to generate approximate code that trades off accuracy and performance. Examples of such techniques include AutoTuning and Knowledge distillation, matrix approximation, numerical optimization, rounding, truncation, statistical sampling, Taylor series approximation, linearization, neural networks, and piecewise linear approximation. There are several different techniques for optimizing code using approximate methods.

1) COMPUTATION SKIPPING

Computation skipping is a technique used in computer programming to improve the performance and efficiency of code by reducing the number of computations that need to be performed [123]. This technique involves the exclusion of code blocks based on predetermined criteria such as acceptable levels of Quality-of-Service degradation, constraints established by the programmer, and/or predictions made regarding the accuracy of the output at runtime. It involves skipping unnecessary computations that would not change the outcome of the program. Skipping computations in Convolutional Neural Networks (CNNs) has been the subject of numerous studies. CNNs excel in many recognition tasks, but their computational complexity limits their use on power-constrained platforms. Therefore, Lin et al. [124] introduced PredictiveNet, a method for reducing the computational complexity of CNN without significant accuracy loss. It predicts sparse outputs from non-linear layers, bypassing most computations. It skips many CNN convolutions during runtime without changing the CNN structure or needing additional branch networks. When tested, PredictiveNet reduced computational cost by a factor of 2.9 compared to a standard CNN, with minimal accuracy degradation. There are several different techniques for computation skipping, including:

a: LOOP PERFORATION (SKIPPING)

This technique involves selectively skipping iterations of a loop that are not critical to the output in a software program to provide performance and energy gains in exchange for QoS loss. There are several skipping approaches for a different set of iterations based on different criteria, such as skipping every other iteration, skipping based on a condition, or skipping until a certain threshold is met. Loop tiling involves breaking a loop into smaller sub-loops to reduce the memory access pattern [125], [126], [127], [128]. Figure 8 shows a loop that iterates over a set of data. For each iteration, the loop checks a perforation condition. If the condition is true, the iteration is skipped. Otherwise, the iteration is executed. Loop perforation is a powerful technique that can be used to improve the performance and accuracy of loops. However, traditional loop perforation, which only considers the number of instructions to skip, overlooks the significant influence of differences between instructions and loop iterations on performance and accuracy. To address this issue, Li et al. [126] advanced loop perforation with their Sculptor system, introducing selective dynamic loop perforation to enhance performance and accuracy by skipping specific instructions within loop iterations. Despite challenges in instruction analysis and strategy optimization, they proposed compiler improvements for selective and adaptive perforation. Testing across eight applications showed this method outperforms traditional loop perforation, achieving speedups of 2.89x and 4.07x with 5% and 10% error tolerances, proving its effectiveness in boosting both speed and accuracy.

The graph algorithms are widely used in high-performance and mobile computing. The performance of these algorithms can vary due to input dependence, i.e., changes in the input graph. Omar et al. [128] proposed an input-aware loop perforation predictive model called GraphTuner, which allows graph algorithms to systematically trade off accuracy for performance and power benefits. In this approximate computing circumstance, they examine the consequences of input dependence on graph algorithms. This helps to identify the requirement for adaptation of inner and outer loop perforations depending on input graph features such as graph density or size. The outcomes indicate an average performance improvement of approximately 30% and a power utilization improvement of about 19% at a program accuracy loss limit of 10% for NVidia® GPU.

The loop perforation technique has also been used in approximation frameworks for optimizing embedded GPU kernels. Maier and Juurlink [127] proposed a new memory-aware perforation approach for GPU kernels, optimized for embedded GPUs, and a framework for automatic loop nest approximation based on polyhedral compilation. This framework introduces new multidimensional perforation schemes and generalizes existing ones. To enhance result accuracy, a reconstruction technique is incorporated, and a pruning method is proposed to eliminate low-quality transformations in the large transformation space.

b: MEMORY ACCESS SKIPPING (MAS)

MAS is a new approach that tries to optimize storage and memory access by skipping unnecessary for uncritical data. To achieve this technique, we need to statistically analyze and profile the code in offline and real-time to figure out unnecessary memory access. The main goals of this technique are to save energy on memory access, ineffective utilization of bandwidth, and the overall performance of the system. MAS boosts performance mainly for memory-bound applications. However, the implementation of MAS faces two challenges: the complexity and managing the overheads of accurate skip detection. However, this area has significant potential for improving performance and power consumption.

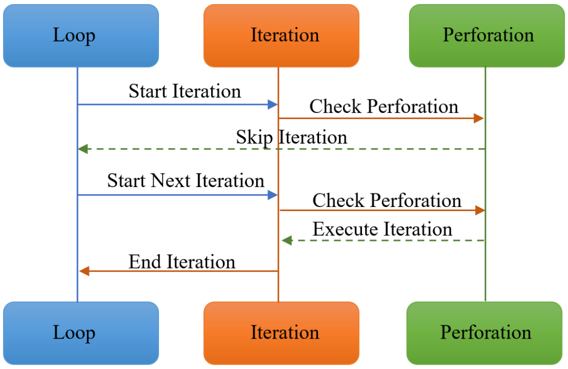


FIGURE 8. Flowchart illustrating the concept of loop perforation.

Due to the growth of dataset sizes and multi-level cache hierarchies, memory performance in data mining applications is a significant problem being addressed by the current research. The important methods in data mining applications are recursive partitioning methods such as decision trees and random forest learning. To address this issue, Kislal and Kandemir [129] introduced a framework to optimize performance in recursive partitioning applications while managing accuracy loss. Their key components include a data access skipping module (DASM) guided by user-defined strategies and a heuristic to predict the impact of skipping data accesses for accuracy preservation. This proposed framework leverages the inherent flexibility in these applications to enhance performance with minimal accuracy losses. Experimental evaluations show that this method can enhance performance by up to 25% with minor accuracy losses of up to 8%. The authors also prove the framework's scalability under different accuracy needs and its potential for memory performance improvement in NoC/SNUCA systems. Also, Raparti and Pasricha [130] introduced two innovative solutions for memory bottlenecks in many-core GPGPU (NoC) architectures. They introduced an approximate memory controller (AMC) to lower DRAM latency and optimize scheduling, and a low-power NoC (Dapper) to enhance communication efficiency. Experiments show the architectures boost NoC throughput by 21% and cut latency and power use by 45.5% and 38.3%, respectively.

Certain researchers have directed their attention towards the deliberate skipping of costly data accesses. Researchers must be aware of three critical questions. What is the upper limit of skipping data accesses while maintaining a specified level of inaccuracy? The significance of architectural awareness in discerning which data accesses to eliminate is a pertinent inquiry. Is it always the case that two executions, which both skip the same number of data accesses, will yield identical output quality? Karakoy et al. [131] attempt to answer these critical questions through proposing a program slicing-based approach that identifies the set of data accesses to skip.

2) ITERATIVE REFINEMENT

Iterative refinement is a method used when we are dealing with ill-conditioned systems, where small changes in the input can lead to large changes in the output. The technique involves starting with an initial estimate of the solution and then iteratively refining the estimate until a desired level of accuracy is achieved. Iterative refinement can be used in various fields, including computer graphics, machine learning, and scientific computing. Recent research has shown that iterative refinement can be particularly effective in certain domains, such as optimization and machine learning [132], [133], [134], [135], [136]. Recently, Yang et al. [133] highlighted the effects of applying iterative refinement in machine learning. They introduced a compact deep neural network and applied learned gating criteria during the training phase to figure out if the weight-sharing cycle would work. This mechanism gives adaptive behavior to the model. However, iterative refinement may not always be best. It may converge slowly or not at all for ill-conditioned systems.

3) EARLY STOPPING

Early Stopping is a technique used to improve performance and prevent poor generalization in machine learning models by stopping the training process before the maximum number of iterations or epochs is reached. The primary goal of early stopping is to prevent overfitting, reduce computational costs, and enhance the efficiency of the training process. Early stopping can be achieved by monitoring various metrics during the training process when certain criteria are met. Datasets are divided into three subsets: training, validation, and test subsets. The training dataset is utilized for modeling and assessing accuracy, whereas the validation subset measures model generalization. A threshold is set so as to decide the early stopping condition and the ideal number of epochs for training when the error on the validation subset drifts from that on the training subset. As seen in Figure 9, during the early training of a model showing high bias and low complexity, both training and validation errors tend to drop. This is evident in the underfitting area, where bias and generalization are included. In this area, the behavior is that of a model that has not been trained enough to recognize the patterns in the data. In the overfitting area, the variance or error increases as a result of the model being trained for too long. This is evident in the growing divergence between training and validation errors and the loss of generalization. There are a couple of techniques for mitigating overfitting, such as early stopping, regularization techniques like L1/L2 regularization, data augmentation, and ensembling. However, the early stopping approach is a simpler and easier way than the others.

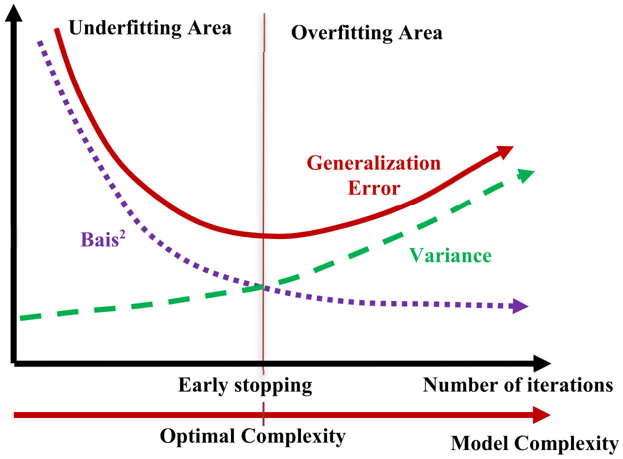


FIGURE 9. The importance of early stopping approach in the machine Learning.

Recent research has shown that early stopping can be particularly effective in deep learning models, which are often computationally expensive and require large amounts of data for training [137], [138]. Early stopping can be applied to various types of algorithms, including search algorithms, optimization algorithms, and machine learning algorithms. Together, early stopping and the validation set help to find that ideal balance to establish the optimal capacity for a model's training, as shown in Figure 9. There are several types of early stopping techniques that can be used in machine learning to stop the training process before it reaches the maximum number of iterations or epochs. There are some common types, such as Fixed early stopping [139], [140], Adaptive early stopping [141], Noisy early exit [140], [142], Early stopping with patience [143], and Gradual unfreezing [144], [145], [146].

Another recent study that uses early stopping is "Early Stopping without a Validation Set" by Mahsereci et al. [138]. The authors proposed a validation-free early stopping approach that depends on the statistics of locally accessible computed gradients. This method increases a little in computation complexity, delay, and memory. The method achieved comparable or better performance compared to traditional methods that use a validation set.

4) FUNCTION APPROXIMATION

Function approximation is a technique used in mathematics and computer science to estimate an unknown function using a set of input-output pairs or data points. The purpose of this technique is to figure out a function that approximates the true underlying function as closely as possible. There are many methods for function approximation, including polynomial interpolation, the CORDIC algorithm, regression analysis, spline interpolation, and neural networks [147], [148], [149]. The implementation of approximate functions within complex systems is facilitated by the utilization of neural networks in software-hardware co-design. This approach involves converting traditional approximable codes into equivalent neural networks, resulting in improved execution time performance at the expense of reduced output accuracy [150], [151]. These techniques are also used at the circuit (hardware) level, and we will discuss them in detail later.

5) PRUNING

Pruning is a technique used a lot in deep learning and machine learning models to make models smaller and simpler. The goal is to remove redundant or unnecessary parameters from the model, which can lead to better generalization performance and faster inference times. There are several types of pruning techniques that can be used depending on the specific application and model architecture [152], [153], [154], [155], [156]. The goal of pruning is to generate a more compact and efficient model that can be implemented on resource-constrained devices or used in real-time applications without reducing accuracy or performance. This can be done through a variety of methods, including **magnitude-based pruning** [157], where weights with small absolute values are removed, and **iterative pruning** [158], where weights are gradually removed over multiple iterations of training. Pruning methods can be broadly categorized into **unstructured pruning** [157], [158], [159], and **structured pruning** [79], [160], [161], [162], [163]. For example, in deep learning, **Neuron or Weight pruning** can be used to remove neurons or connections that do not contribute significantly to the final output. This can reduce the computational complexity of the model and speed up the training process [164]. Another example is that in a convolutional neural network, **filter pruning** can be used to remove filters that have low activation values or are redundant, which can help to decrease the computational cost and memory requirements of the model [163], [152], [165], [166], [167]. For example, Luo et al. [168] proposed a filter pruning algorithm called ThiNet, which considers the interdependence of filters in a layer and prunes them in a way that preserves accuracy. The findings indicate that ThiNet achieves a significant reduction in computational resources for VGG-16, including over 3 times fewer FLOPs and over 16 times compression, with a minimal accuracy loss of 0.52%. Additionally, ThiNet cuts parameters and FLOPs by over half, with a slight accuracy decrease of about 1%.

6) SPARSITY

Sparsity is a crucial concept in modern data processing and machine learning to optimize energy, memory, and computation in algorithms, all without significant loss of accuracy. Sparsity is a technique employed to ensure that a large proportion of the elements in a dataset or matrix are zero or have values that will not significantly impact a calculation. There are many techniques to achieve sparsity: pruning, regularization, dimensionality reduction techniques, and matrix factorization methods. We discussed pruning techniques in the previous subsection. We can use regularization techniques like L1 regularization to handle sparsity in neural network model parameters. To reduce the dimensionality of datasets and extract features, we can use dimensionality reduction techniques like Principal Component Analysis (PCA) and t-Distributed Stochastic Neighbor Embedding (t-SNE), and Matrix Factorization methods like Singular Value Decomposition (SVD) and Non-negative Matrix Factorization (NMF). We discussed dimensionality reduction techniques in the compression subsection. These techniques and methods are parts of low-rank matrix factorization techniques [169], which are unsupervised learning methods used for data analysis tasks such as dimension reduction, feature extraction, blind source separation, data compression, and knowledge discovery.

Recent years have seen the success of artificial neural networks in solving real-world problems and the rapid increase in their complexity and parameters. Larger networks are more computationally and memory-intensive, making them difficult to use on embedded devices [170]. To address this, there is growing interest in sparsifying neural networks. Sparse neural networks can match the performance of fully connected networks while using less energy and memory, making them ideal for resource-limited devices [171]. NVIDIA [172] has developed a straightforward and widely applicable technique for generating sparse deep neural networks through inference by utilizing a particular form of sparsity structure known as 2:4 pattern. For example, the NVIDIA Ampere architecture's third-generation Tensor Cores in A100 GPUs utilize fine-grained sparsity in their neural network weights, enhancing matrix multiplication speed in deep learning without losing accuracy. Another example, Lu et al. [173] aims to develop an FPGA accelerator for sparse CNNs, addressing inefficiencies in existing FPGA architectures designed for dense models. The proposed solution includes a weight-oriented dataflow for handling irregular connections in sparse convolutional layers, a tile look-up table to eliminate runtime indexing matches, and a weight layout with a channel multiplexer to prevent data access conflicts. Experiments show the accelerator achieves 223.4-309.0 GOP/s on Xilinx ZCU102, offering a 3.6x-12.9x speedup over previous dense CNN FPGA accelerators. Also, Tragoudaras et al. [174] used a state-of-the-art HLS tool to implement a MobileNetV2 model by integrating design methodologies with sparsification techniques, including sparse matrix methods and two weight pruning approaches. The objective is to develop hardware accelerators that maintain error metrics comparable to state-of-the-art systems while significantly reducing inference latency and resource utilization.

In sum, Sparsification techniques, such as sparse matrix methods and weight pruning, are essential for enhancing the efficiency of deep neural networks. By reducing the number of non-zero elements, these techniques lower memory usage and computational demands, enabling faster and more resource-efficient inference. Additionally, they help maintain model accuracy while optimizing hardware performance. The challenges of implementing sparse algorithms can be more complex compared to their dense counterparts. However, sparsification is a crucial strategy for advancing the practicality of real-time AI applications.

7) APPROXIMATE MEMOIZATION

Memoization is a technique employed to store the outcomes of computationally expensive operations for subsequent utilization in cases where identical operations and input data are encountered [175]. Different levels of accuracy can be used to compute many algorithms. Approximate computing exploits this to decrease execution time by determining the tradeoff between performance and accuracy. Approximate memoization extends this concept by providing approximate results for new input data that correlate with previously computed and stored data. This approach, which relies on software frameworks, compilers, and programmer's decisions, is particularly useful in optimizing computational efficiency. Real programs often contain redundant computations due to factors like repetitive inputs, pattern repetitions, repeated function calls, and poor programming practices [176]. There are many works that achieve the functions or tasks memoization either at compile-time or at runtime. Although, Large Language Models (LLMs) train on extensive datasets, they can potentially expose sensitive information. Data preprocessing and differential privacy techniques are designed to prevent data memorization and face the challenge of reliance on data structure assumptions that might lead to false privacy concerns.

Performance enhancement is a critical requirement in high-performance and embedded computing applications, often relying on the expertise of performance engineers to optimize their efficiency by leveraging both manual work and numerous analysis and optimization tools. Pinto and Cardoso, [175] introduced a methodology that automates code analysis and memoization to simplify the application of memoization. It aims to assist developers without optimization expertise and provide customizable analysis for performance engineers. This approach caches the results of computations for efficiency and is tailored for both novice developers and expert performance engineers. Also, Suresh et al. [177] introduced a compile-time technique for function memoization, extended its scope to user-defined functions, and enabled transparent application to dynamically linked functions.

High-performance and energy-efficient memoization approaches face drawbacks like high runtime overheads and limited applicability, while conventional hardware techniques use specialized caches that consume excessive area and energy. Zhang and Sanchez [178] introduced MCACHE, a hardware technique that utilizes data caches for memoization while sharing cache memory with regular program data. This method boosts performance by 21x and outperforms software memoization by 2.2x in runtime efficiency. The need to improve computing efficiency by reducing function call overhead through approximate function memoization [179]. Therefore, Arundhati et al. [180] introduced a software approach to function memoization that bypasses the execution of functions implemented using approximate computing techniques. A decision-making rule utilizing the Bloom filter and Cantor's pairing function is proposed to determine whether to search the look-up table (LUT) or perform the actual computation. Additionally, a simple approximation technique is proposed to search in the LUT to find an approximate one. Evaluation conducted using benchmarks from the AxBench suite demonstrates the effectiveness of the proposed technique. To memoize a block of code, Liu [181] proposed a hardware-compiler Codesign framework, AxMemo. The goal of AxMemo is to memoize code blocks with many inputs. In other words, AxMemo tries to replace long instruction sequences with a few hash and lookup operations. Brumar et al. [176] introduced Approximate Task Memoization (ATM), a novel approach to memoizing functions or tasks at runtime. Memoization of previously executed tasks enables predictions of future results without actual execution, preserving accuracy. The runtime system also incorporates task similarity measurement and correctness assessment to automatically determine the feasibility of task approximation. The method results in a 1.4x speed increase with memoization alone and a 2.5x boost when adding task approximation, with a negligible average accuracy drop of 0.7% (up to 3.2%).

Contrary to the aforementioned techniques, researchers from Microsoft, in collaboration with researchers from the Weizmann Institute [182], introduced a new training procedure for ReLU networks that utilizes complex recombination of neurons to achieve approximate memorization. This approach aims to address the shortcomings of previous constructions and achieve efficient memorization with an almost ideal number of neurons and weight magnitudes.

In relation to Large Language Models (LLMs), LLMs train on massive amounts of text, including sensitive information. LLM can potentially expose this sensitive information, including personal information. Previous research concentrated on literally preventing data memorization using data preprocessing and differential privacy techniques. This process faces the challenge of reliance on data structure assumptions that might lead to false privacy concerns and impact the model's overall quality. Current research treats this issue of approximate memorization in LLMs by using Reinforcement Learning. For example, Kassem [183] proposed a novel framework that employs a reinforcement learning approach, specifically Proximal Policy Optimization (PPO). This framework uses a negative similarity score, such as BERTScore or SacreBLEU, to measure how close the LLM's output is to the memorized data. If it's too similar, that's a negative reward.

8) ARCHITECTURE SEARCH

This technique is a process in machine learning where a computer algorithm searches for the optimal architecture, or configuration, of a neural network (NAS) for a specific task [184]. There are several approaches to architecture search, including reinforcement learning, evolutionary algorithms, and Bayesian optimization. These methods can be used to explore the vast space of possible network architectures and identify those that are most likely to perform well on a given task.

9) KNOWLEDGE DISTILLATION

Knowledge distillation is an approach using machine learning to transfer knowledge from a large, sophisticated teacher model to a simpler, faster, and smaller student model. The student model mimics the teacher's behavior efficiently, using limited resources. It makes this model adequate for implementing on resource-constrained devices and using in real-time applications. For example, in natural language processing, a large language model can be distilled into a smaller and faster model that can be deployed on mobile devices [185], [186]. The soft targets produced by the teacher model can be seen as a compressed representation of the knowledge learned by the teacher model, and by incorporating them into the student model training process, the student model can effectively learn from the teacher's knowledge. Knowledge distillation has been applied to a variety of tasks and has been shown to be effective in minimizing the scale and computational intricacy of deep neural networks without compromising their effectiveness [187], [188], [189], [190], [191], [192], [193], [194]. For example, Hongxu et al. [195] proposed a new method called DeepInversion. This technique reverses a trained network to create class-specific images from random noise, refining the input and using batch normalization data for regularization. Adaptive DeepInversion enhances image variety by leveraging differences between teacher and student network outputs. The method has been applied to network pruning, knowledge transfer, and continual learning without needing original data. Existing knowledge distillation techniques used to train student networks typically rely on task-specific data. However, the availability of such data may be limited due to privacy or confidentiality considerations. Several techniques involve generating training samples from the teacher network. Nevertheless, the generated images often exhibit discrepancies when compared to authentic ones, thereby imposing limitations on the performance of the student network. Therefore, Tang et al. [188] proposed an approach for building training datasets based on proposed web crawling (ICCD). They proposed a pseudo-classification strategy and frequency-domain supervision (PCFS) to enhance performance by reducing the divergence between the generated ICCD and target dataset. The findings show the proposed PCFS surpasses the existing data-free methods. The code is available online.

B. APPROXIMATE PARALLELISM AND RELAXED SYNCHRONIZATION

The relaxed synchronization technique removes synchronization points that represent one of the major bottlenecks in parallel applications, as synchronization points can cause threads or processes to spend a lot of time waiting [196]. The efficient execution of concurrent applications on multicore systems necessitates the implementation of synchronization mechanisms that consume significant amounts of time, either to enable access to shared data or to fulfill data dependencies. In general, developers often use synchronization to prevent undesirable interactions like data races when multiple parallel threads access shared data. However, there are several drawbacks associated with standard synchronization mechanisms: synchronization overhead (time and space costs), parallelism reduction (threads waiting), and failure propagation (unperformed synchronization operations can cause threads to hang indefinitely). Every synchronization point, acting as a serialization point, can potentially impede parallel scalability [197]. Therefore, researchers are indeed exploring the concepts of relaxed synchronization and approximate parallelism to mitigate these issues with trading minor computational errors for enhanced performance and efficiency. The synchronization error has higher performance compared with mixed precision but produces more errors. In particular, synchronization errors introduce non-deterministic errors that are complex to handle. Loading data into local memory requires a synchronization point to ensure that all threads in a block have the same view of the local memory. To decrease the time lost during synchronization, SYprox was proposed by [198], which provides a synchronization elimination mechanism that defines a way to handle the number of synchronization points. Lee et al. [199] introduced a novel algorithm for solving large-scale quadratic programming problems in parallel computing systems. They proposed "lazy synchronization," which reduces the synchronization rate while improving processor utilization and convergence speed. Tested on Amazon's 40-node distributed system, the algorithm achieved speedup by 160x and reduced communication overhead by 99.65% using the relaxed synchronization technique compared to conventional methods. To convert inherently sequential code to parallel approximations, Stitt and Campbell [200] introduced an automatic parallelizing approximation-discovery framework, PANDORA, based on symbolic regression machine learning. The findings show the code accelerated by 2.3x to 81x with maintaining acceptable accuracy. The framework's capabilities are further demonstrated through FPGA experiments and by eliminating loops from the code. The authors defined some limitations of PANDORA framework and suggested some solutions. For example, PANDORA faces difficulty handling complex problems due to its reliance on symbolic regression, and consumes a lot of time for discovering approximations, etc.

The majority of these works were noticed by Bueno et al. [201], who applied the aggregate elimination of all synchronization points without accounting for output quality variations due to varying input data. Therefore, Bueno et al. [201] proposed a novel strategy by using supervised learning methods to relax synchronization in parallel applications that allow trade-offs between quality and execution time. Also, the authors proposed the relax factors to be applied to the input, application, and execution environments together. The results show this proposed technique enhanced the K-means algorithm by a gain factor of 3.5x for video processing while maintaining an acceptable quality rate.

Overall, applications like image processing and neural networks can tolerate some errors, offering potential for significant improvements in execution time and energy use. Key software approximation techniques include mixed precision, which uses lower precision data representation; perforation, which skips instruction blocks, loop iterations, or data assuming nearby values are similar; and relaxed synchronization, which removes synchronization points, a major bottleneck in parallel applications. These approaches vary in performance and error. Typically, perforation and synchronization elimination offer higher performance but produce more errors than mixed precision. Synchronization elimination also introduces complex, non-deterministic errors.

C. PROGRAMMING FRAMEWORKS AND TOOLS

1) PROGRAMMING FRAMEWORKS FOR AXC

Approximate programming frameworks are considered tools and mechanisms that help developers integrate approximations into their programs in a controlled way to manage the trade-off between accuracy and resource usage. **Approximate programming languages** are particularly advantageous in scenarios where computational efficiency is of paramount importance and minor inaccuracies in the final output do not significantly impact the overall result. Such scenarios are commonplace in domains such as machine learning, signal processing, and big data analytics, where computations can be computationally intensive. Programming languages with approximate features offer novel constructs and abstractions that empower developers to clearly define specific portions of a program where approximations are deemed acceptable. The compiler and runtime system utilize these specifications to enhance the program's performance, energy efficiency, or other measurable factors, while also guaranteeing that the ultimate outcome falls within acceptable margins of error. Approximation-enabled compilers provide a powerful means of exploiting the error resilience of applications. By automatically introducing approximations, they can significantly improve performance and energy efficiency without requiring extensive manual intervention. However, they also face several challenges. One key challenge is ensuring that the approximations do not significantly degrade the quality of the program's output. This requires careful analysis of the program's behavior and the impact of different approximation techniques. Another challenge is managing the trade-off between accuracy and performance, which can require sophisticated heuristics and tuning mechanisms. For example, ACCEPT compiler was developed by Sampson et al. [202], which uses a combination of static and dynamic program analysis to automatically determine the approximable regions of a program. It then applies a variety of approximation techniques to these regions, such as loop perforation and task skipping. Approximate programming languages can be classified based on their approach to approximation:

a: LANGUAGE EXTENSIONS

These are conventional programming languages augmented with new syntax and semantics to support approximation [203]. Examples of this category include EnerJ, Rely [204], and Chisel. EnerJ [205] is a Java extension with a design applicable to languages where data types are explicitly declared by programmers. FlexJava [206] streamlines approximate programming by automating annotations, making energy-efficient coding simpler and safer. FlexJava matches EnerJ's energy savings by reducing the number of annotations by 2x to 17x and annotation time by up to 12x in user studies. Typically, the foundational elements of language extensions manifest in three primary stages:

- Introduction of Data Types: Extensions such as EnerJ in Java incorporate novel data types like approx int or approx float, which, though less precise in calculations, yield benefits in performance and energy efficiency.
- Modification of Overloaded Operators: Arithmetic operations such as addition, subtraction, multiplication, and division may undergo alterations for approximate data types, facilitating the management of error propagation or enabling more relaxed calculations.
- Implementation of Annotations: These serve as directives for the compiler, delineating the contexts in which approximations are viable and specifying the acceptable threshold for errors, (e.g., @approx_tolerance (0.05) for a function).

Introducing language extensions for approximate computing faces key challenges: rigorous error tracking to control compounded inaccuracies, ensuring type safety to avoid mixing data types, and overcoming user resistance by offering clear benefits and easy integration to encourage widespread adoption.

b: PROBABILISTIC PROGRAMMING LANGUAGES

These languages incorporate uncertainty directly into the language and employ statistical methods to compute approximate results. PPLs are designed to express probabilistic models and perform inferences over them [207]. They provide constructs to define random variables, specify dependencies between variables, and encode probabilistic algorithms. PPLs often incorporate advanced inference techniques like Markov chain Monte Carlo (MCMC) and variational inference. FACTORIE [208], FlexJava [206], Venture [209], BiiP [210], Stan [211], SlicStan [212], Gen [213], Hakaru10 [214], HackPPL [215] Anglican [216], Infergo [217], Aloe [218], PyMC3 (Python), and Pyro [219] are representative examples of this category. In addition, there are many studies [220], [221] developing operational semantics as a basis for probabilistic programming languages such as Anglican, Venture, and Church. For example, Dylus et al. [222] introduced a library for probabilistic programming in the functional logic programming language Curry. Another example, Gen is a probabilistic programming language embedded in Julia, designed by Marco [213], which offers sufficient expressiveness and performance for general-purpose use. Gen automatically optimizes custom inference strategies for specific probabilistic models using static analysis. The findings indicate that Gen's prototype matches Stan's speed [211], is only about 1.4 times slower than a custom Julia sampler, and is roughly 7,500 times quicker than Venture, another probabilistic language allowing custom inference. FACTORIE [208] is a Scala toolkit for probabilistic models, providing tools for building factor graphs, parameter estimation, and inference, developed by McCallum and his colleagues. FACTORIE offers learning and optimization tools for classification and prediction, plus NLP features like segmentation and tokenization. UMass Amherst offers tutorials and downloads for more information [223].

c: STOCHASTIC PROGRAMMING LANGUAGES

These languages incorporate randomness directly into their computations. Statistical programming languages focus on expressing statistical models and performing data analysis. They provide a wide range of statistical functions and libraries for tasks such as data manipulation, regression analysis, hypothesis testing, and visualization. While they may not explicitly deal with uncertainty, they often support probability distributions and statistical techniques for uncertainty estimation. They are often used in simulations, optimization, and machine learning [224]. Examples include AMPL, GAMS, SimJulia, StochasticPrograms.jl and SimPy.

d: BAYESIAN PROGRAMMING LANGUAGES

These languages combine probabilistic modeling with Bayesian inference to generate a library of functional programming languages for Bayesian modeling and inference [225], [226]. Bayesian programming aims to substitute traditional languages with a probabilistic approach that accounts for uncertainty and incompleteness. One popular example of a Bayesian programming language is "JS" (Just Another Gibbs Sampler), which is specifically designed for Bayesian analysis of complex statistical models. JAGS provides a high-level syntax for creating and manipulating probabilistic graphical models and supports a wide range of built-in probability distributions and statistical functions. Stan [211] is a probabilistic language optimized for Bayesian inference with Hamiltonian Monte Carlo methods, automating model specification and inference. Such languages are valuable in fields like machine learning and data analysis, where probabilistic reasoning is crucial. By providing a dedicated framework for Bayesian modeling, these languages make it easier for developers to build applications that incorporate sophisticated probability [227].

2) APPROXIMATION COMPUTING FRAMEWORKS

a: APPROXIMATE COMPUTING FRAMEWORKS

There are several frameworks and libraries that have been developed to support software-level approximate computing, such as TensorRT, TVM, and FlexFlow. These frameworks provide tools and APIs for optimizing and deploying approximate computations on different hardware platforms, such as CPUs, GPUs, and FPGAs. They can also support different levels of approximation and error metrics and can be used to automate the process of tuning and optimizing the approximate computation [228], [229], [230], [231], [232]. The challenge of optimizing applications requires intensive resources and flexibility in precision. For example, ApproxTuner [233] is an automatic framework to address this issue. ApproxTuner optimizes tensor-based applications for accuracy-awareness, requiring just broad quality goals. It integrates approximations across algorithmic, software, and hardware levels through a unique three-phase tuning method encompassing development, installation, and operation stages, ensuring adaptability across devices. The framework introduces predictive approximation-tuning for faster autotuning by estimating the accuracy effects of approximations analytically. Tested on 10 CNNs and a CNN-image processing mix, it achieved up to 2.7x speedup on GPUs and 1.9x on CPUs with minimal accuracy loss. ApproxTuner's novel tuning method outpaced traditional tuning, offering similar advantages more efficiently. Liu et al. [234] introduced an adaptive program graph that allows for customizable quality at the user level, based on criteria set by developers. Approxilyzer framework [235] used both static and dynamic analysis methods to help find opportunities for approximation in software applications at the binary level. This makes sure that certain computations can be approximated without losing accuracy.

The rising power needs of DNN accelerators have led to the use of approximate multipliers in modern solutions. However, the accuracy assessment of these approximate DNNs presents a challenge due to the insufficiency of approximate arithmetic support in existing DNN frameworks. To mitigate this, Danopoulos et al. [236] proposed AdaPT, a rapid emulation framework that augments PyTorch, enabling it to support both approximate inference and retraining aware of approximation. AdaPT, designed for seamless deployment, is compatible with a majority of DNNs. AdaPT notably enhanced error recovery and reduced inference time by up to 53.9x across different DNN models and applications compared to conventional approximations.

b: APPLICATION-AWARE FRAMEWORK

This framework involves identifying the computations that are critical to the functionality of the application and ensuring that these computations are not approximated. This involves analyzing the application and identifying the critical computations that must be executed with high accuracy to ensure the overall functionality of the application [229], [237]. It is also called approximate-aware design framework [238], [239]. ApproxHadoop is a framework for implementing approximate computing in big data applications. It provides a way to automatically identify opportunities for approximation and selectively apply them to reduce the computational cost of data processing [48]. Hanif et al. [240] introduced a framework to systematically analyze the error resilience of deep CNN and identify parameters for applying approximate computing techniques.

c: DYNAMIC APPROXIMATION FRAMEWORK

This framework involves dynamically identifying the computations that can be approximated based on the input data and the current state of the application. This involves monitoring the application and identifying the computations that can be approximated based on the current state of the application [241], [242], [243], [244]. For example, Wang et al. [245] introduced the Runtime Machine Learning-based Identification Model (RMLIM) to highlight noncritical segments within a software program's data flow graph. Trained offline with a designated dataset, RMLIM is subsequently applied at runtime for individual inputs. This simplifies the identification process and enhances its applicability to real-time scenarios. Preliminary results indicate that RMLIM retains comparable energy efficiency and accuracy to prevailing runtime AC techniques. It notably reduces the execution time by 40 to 61 percent.

Recently, Soni et al. [246] introduced "As-Is," an innovative Anytime Speculative Interruptible System, designed to enhance the adoption of approximate computing by addressing the lack of hardware support and real-time accuracy guarantees. They proposed this system to leverage approximate computing to deliver early outputs that improve over time, ensuring eventual full accuracy. It merges approximate and speculative computing to repurpose existing architectures for efficient approximation, offering a solution that adapts to real-time needs and allows users to choose between immediate results and waiting for complete accuracy.

d: DATA/INPUT-WARE APPROXIMATE FRAMEWORK

The framework aims to identify the data that can be reasonably approximated without causing substantial disruption to the system's output. This is achieved through the introduction of intentional faults into the variables, followed by an analysis of the resulting impact on the output quality [247], [248].

The approximation-based programming approach is well-suited for error-tolerant applications on constrained-resource devices, as it allows for efficient computation and storage of program data. This is especially important for devices like smartphones and tablets, where battery life is crucial. However, implementing this paradigm requires source code annotations and type qualifiers, which can be problematic for large, real-world applications with limited access to source code. Pooja et al. [248] and Bernard et al. [247] present an innovative sensitivity analysis framework which facilitates the generation of annotations for programs designed for approximate computing. The framework facilitates the extraction of information pertaining to the sensitivity of output, enabling the identification of a crucial subset of data that requires precise computation and storage, while the remaining data can be approximated.

e: PROFILING FRAMEWORK FOR APPROXIMATE COMPUTING

It is a tool designed to analyze and measure the performance and accuracy of algorithms that use approximation techniques. These algorithms are often used in modern applications that require rapid processing of large data sets. This tool trades off result accuracy with faster execution or less memory use. The profiling framework, such as AXPROF [249], provides developers with the necessary support to implement these algorithms effectively and automatically. Based on the desired accuracy specified by developers, this framework begins to generate code for statistical analysis and models for analyzing accuracy, memory use, and timing. For verification and assessment, this framework conducts suitable statistical tests for implementation to be sure the implementation meets the specification. This type of framework is crucial in identifying bugs and performance optimizations in the implementation of approximate algorithms. AXPROF profiled 15 applications across data analytics, numerical linear algebra, and approximate computing, effectively detecting bugs and providing various performance optimizations. The tutorials and examples for this framework are available online.

VII. ARCHITECTURAL-LEVEL APPROXIMATE COMPUTING TECHNIQUES

A. APPROXIMATE MEMORY TECHNIQUES

The constant communication between processors and off-chip memory causes memory subsystems to be the largest consumers of time and energy in modern computer architectures, from servers to mobile devices. The escalating disparity in speed between the CPU and the external memory, known as the "Memory Wall". This problem is a significant bottleneck in computer system performance. In order to overcome the memory wall and narrow the gap between processors and memories, designers have experimented with a wide variety of circuit and architectural advances, including 3D integration [250], bigger on-chip caches, memory-level parallelism [251], [252], faster off-chip interconnects [253], new memory hierarchies, near-memory processing or in-memory computing [254], [255], and more [256]. Figure 10 shows the classification of computing systems based on where they process data [257]. They are still not satisfied with reducing memory energy consumption for many developing algorithms, such as machine learning, that pose increasing demands on the memory chip. Consequently, there is a need for the development of novel methodologies to enhance both energy efficiency and performance.

A common trait among the majority of emerging applications that heighten memory consumption involves the ability to endure approximations within the foundational computations or data. Despite this, these applications continue to generate outputs that meet an acceptable level of quality. Approximate computing is one of the techniques that improves energy and performance by leveraging the inherent resilience of many developing applications using techniques on memories. A key element of this methodology is the application of approximate memories. These are intentionally designed memory circuits known to demonstrate imperfect data retention, a characteristic that may be attributed to either the inherent tendency of these circuits to slowly lose data over time or to errors that transpire during read/write operations [9], [19]. Typical approximate techniques that have been proposed for developing circuits for approximation memory include, for example, voltage scaling in the case of SRAMs [258], lowering the refresh rate below the nominal value in DRAMs [259], and compressing or encoding the data [256], [260]. To lay the groundwork for understanding these concepts, the rest of this section will offer a brief overview of dynamic and static random-access memories. These two essential types of memory hold significant relevance in the field of approximation memories.

Dynamic random-access memories (DRAMs) have long been the cornerstone of memory storage in embedded systems. Due to its high-capacity, durability, and affordability, DRAM remains the major choice for primary memory in numerous embedded systems. A DRAM is organized into channels, modules, ranks, chips, banks, subarrays, rows, and columns, as shown in Figure 11(a) [261]. Manufactured in various capacities and featuring data bus widths ranging from 4 to 16 pins, DRAM chips exhibit a degree of diversity [262]. For the creation of a wider data bus, numerous DRAM chips are typically amalgamated into a single module, forming what is referred to as a rank. A closer examination of each DRAM chip reveals a composition of numerous banks. Each of these banks contains a series of two-dimensional arrays, or subarrays, composed of individual DRAM cells. DRAM operations can concurrently retrieve data from multiple chips within the same rank. The chips in DRAM direct requests towards a specific bank, row, and column location [262]. There are four commands to achieve the DRAM access operations: the read (RD) command, the write (WD) command, the activation (ACT) command, and the precharging (PRE) command. The Activation (ACT) command opens a row, transferring its contents to the row buffer for read (RD) or write operations. This is followed by cell charging through the Precharging (PRE) command. Figure 11(b) presents an illustration of the instructions associated with DRAM, namely ACT, RD or WR, and PRE. Moreover, it delineates the associated timing parameters, namely the delay from row address to column address (tRCD), the active time of the row (tRAS), and the precharge time of the row (tRP) [262], [263], [264].

While traditional DRAMs have been instrumental in memory storage, there has been a growing interest in optimizing power consumption without significantly compromising performance. This leads us to the concept of Approximate Dynamic Random-Access Memories (AxDRAMs). Approximate DRAMs refer to the subset of DRAM systems in which power conservation methodologies have been instituted at the expense of an increased bit-cell error rate. These entities hold a critical position as fundamental components within the broader domain of approximation computing. By embracing a trade-off between power efficiency and accuracy, approximate DRAMs open new avenues for energy-conscious design in embedded systems and beyond [265].

Memory occupies a disproportionate amount of real estate on an on-chip computer's integrated circuit and system layout. SRAM cell architecture is the most popular kind of memory architecture due to its speed and reliability [254]. While the popularity of SRAM is well-established, optimizing its performance is an ongoing challenge. Various methodologies have been explored to enhance the efficiency of SRAM cells, including supply voltage scaling. The method of supply voltage scaling aims to reduce the power consumption of SRAM cells. However, when a substantial number of cells are in standby mode, this can increase the leakage power across the entire semiconductor chip [255], [266]. Nevertheless, a significant degradation in stability is observed when the supply voltage decreases, which causes an increase in the occurrence of read, write, and hold errors [256], [260]. To minimize the occurrence of failures, we need to design a circuit considering the necessary device capabilities.

In sum, the pursuit of innovative techniques to design approximate memories propels a new wave of research in hardware optimization. and steps forward in developing energy-saving memory technologies to trade off power consumption and a tolerable error. For example, Enrico et al. [267] applied the approximate computing (AxC) methods to analyze hardware accelerator components for deep neural networks, focusing on computation, communication, and memory subsystems. It examines performance enhancement aspects, including approximate multipliers, link voltage swing reduction, voltage over-scaling, and lossy compression methods for internal SRAM memory. The investigation aims to improve computing systems' efficiency and effectiveness. Numerous methodologies and techniques have been explored and developed in the scientific community for the design and implementation of approximate memories, reflecting the complexity and multifaceted nature of this field of study. In the next subsections, these approaches will be explained in detail, and some of their key considerations as well as benefits will be highlighted.

1) APPROXIMATE MEMORY BASED ON REFRESH RATE

Periodic refreshes of DRAM are required, and these procedures may use up to half of the memory's entire power [259]. During the refresh mode, the memory cannot serve any memory access, and this increases the memory access latency, consequently reducing the throughput of total memory.

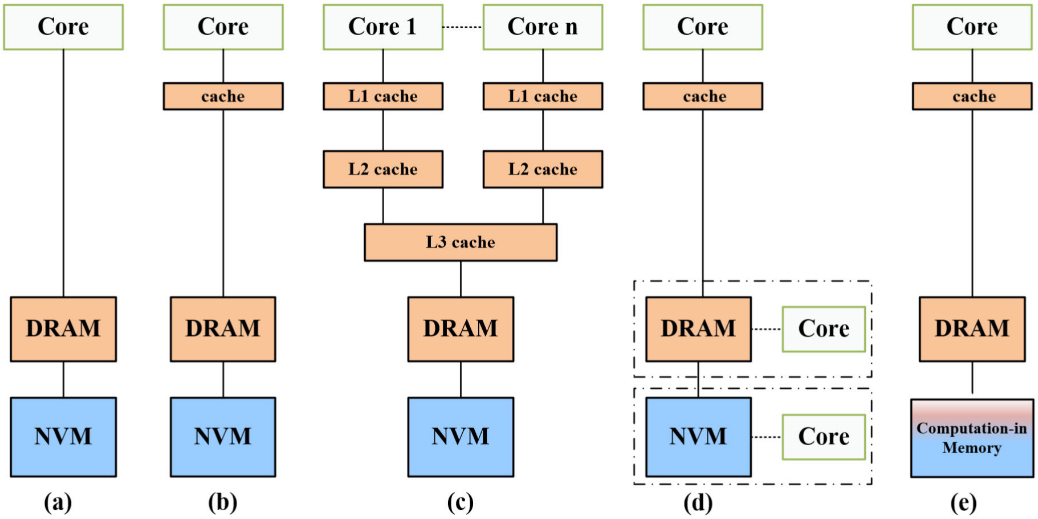


FIGURE 10. Categorizes computing systems by where they process data: (a)-(c) earlier CPU-centric models move data to the core, (d) newer models use near-memory processing, and (e) computation-in-memory (by using memories with built-in processing capabilities (e.g., phase change memory, memristors)) [257].

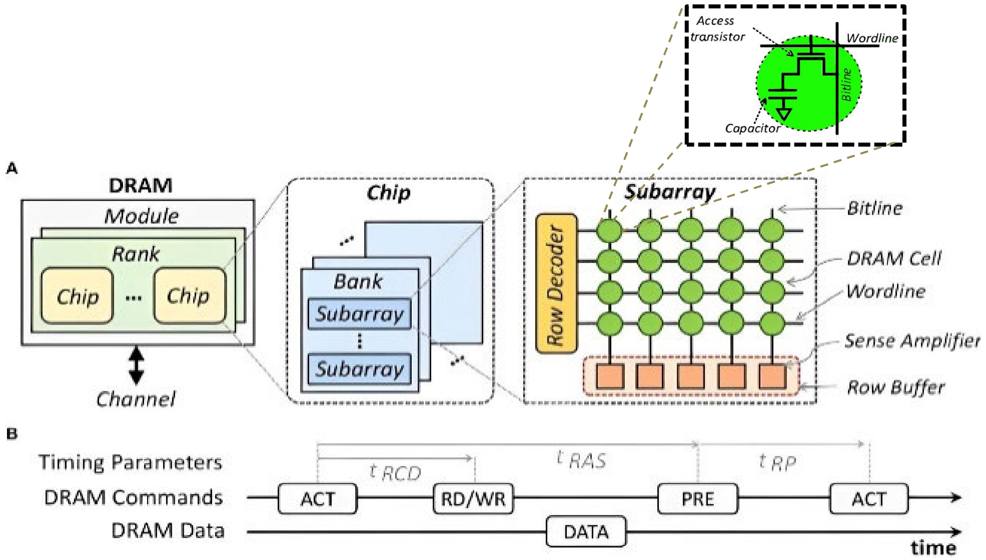


FIGURE 11. Illustration the Dynamic Random-Access Memories (DRAMs) (a) structures organization, (b) Instructions of DRAM access operations [261].

Increasing the refresh period beyond the typical 64 milliseconds utilized by the majority of DRAM-integrated circuits (ICs) nowadays is an effective method for lowering DRAM power consumption [259].

Flikker [12] pioneered one of the initial methods in the domain of approximate memory, specifically targeting low-power mobile DRAM. This approach begins by partitioning an application into two distinct segments: critical and non-critical, as shown in Figure 12. By employing a suboptimal refresh rate, errors are intentionally injected into the non-critical portion, thereby achieving refresh power savings. Consequently, Flikker introduced a software technique that facilitates two refresh controls, allowing for the segregation of DRAM into accurate and approximate sections, a feature particularly applicable to LPDDR DRAM.

A hardware-based method has been developed for approximating DRAM for generating high and low refresh rates for the most and least significant bits of the operand, respectively [268]. This method allows for the partitioning of DRAM pages into more than two parts, with the possibility of suboptimal refresh rates. Raha et al. [259] depended on different quality parameters for partitioning DRAM pages. These parameters are: error characteristics, frequency, critical data percentage, and location. In a specific study conducted by [269], extensive tests were performed on 8 chips of GC-eDRAMs. The results show this approach can save energy, reaching 55% and 75% with an acceptance error rate of 10^{-3} and 10^{-2}, respectively. The authors used refresh rates ranging from 11 to 24 ms.

For a more in-depth look at how DRAM defects affect error-tolerant applications, we recommend seeing the proposed works in Table 3. The approach developed by Enerj [205] allows developers to mark parts of an application that may tolerate errors and be moved into near-primitive RAM (SRAM) or direct-access memory (DRAM). This approach is very important for approximate memories. The concept of approximating non-critical data allows a loosening of accuracy against energy efficiency for these types of applications.

2) APPROXIMATE MEMORY BASED ON APPROXIMATE LSB OR COMPRESSION

The second strategy emerges from investigating the relationship between output quality and the bit error rate of LSB [265], [270], or the degree of compressing data in not-critical regions [256]. Using such techniques can reduce energy consumption. The output quality is little impacted by dropping the LSBs of a data word and setting them to a constant value (i.e., 0). This method requires simple circuits. You can power down or remove bit cells to save a lot of energy [270]. Within memory system technologies, selective ECC has been applied to SRAM and DRAM to reduce MSB errors. Many methods are focused on enhancing the memory word size in traditional ECC memory configurations. For example, a technique has been proposed that extends a 32-bit memory word to 36-bit, incorporating a 4-bit ECC [271]. Reducing the number of '1'bits in data can lower power use in both DRAM and SRAM, as DRAM power is tied to '1'-bit quantity and SRAM power to switch probability and voltage squared [272]. Another approach to enhancing memory efficiency involves the utilization of encoding/decoding [272] or compressing/decompressing techniques [256], [260], [273] for the data written to or read from the memory. These methods can be applied to error-tolerant applications such as machine learning and video and image processing, where slight imprecision is acceptable, to significantly improve memory usage and power requirements. This represents a strategic alignment with contemporary computational demands, offering a pathway to more sustainable and responsive memory management.

Machine learning algorithms often don't fit IoT devices like sensors due to their complexity, high memory, and energy needs. The growth of the IoT has given rise to a new subfield of machine learning known as "tiny machine learning" (TinyML) (IoT). TinyML relieves these challenges and makes the deployment of these algorithms on IoT possible. For example, Raha et al. [274] introduced the foundational concepts of an approximate TinyML system, including input-adaptive approximations [275]. Among these limitations are the technology scaling and memory technologies, which are major challenges in the application of deep learning systems in IoT devices. As technology scales below 20 nm, DRAM cells have shorter retention times, increasing their refresh power. This is especially problematic in memory-intensive applications, where DRAM's refresh power significantly impacts total system power [276]. Innovative solutions are being explored to address this challenge. For instance, a novel approach to enhancing memory efficiency was introduced by Nguyen et al. [276]. They developed a zero-cycle bit-masking (ZEM) technique integrated with ECC within the controller, specifically targeting the asymmetry of retention failures in DRAM. By applying this method, they were able to eliminate the need for DRAM refresh across various applications. The approach was tested on Tiny DNN architectures like AlexNet, DCGAN, and RNN using the Gem5 simulator. The results were promising, with performance improvements of 10.4%, 11.27%, and 17.31%, and total energy reductions of 30.2%, 34.38%, and 43.03% for LPDDR3, DDR4, and HBM, respectively.

3) APPROXIMATE MEMORY BASED ON VOLTAGE SCALING

Another technique to lower energy usage at the expense of decreased frequency is voltage scaling. In the power management strategy known as dynamic voltage scaling, the voltage that is applied to a component may either be raised or lowered, depending on the conditions that are present. The purpose of voltage scaling (reducing voltages) is to reduce energy consumption. SRAM is more sensitive and begins encountering mistakes at a lower operating voltage than logic parts. As voltage is scaled down, SRAMs become more susceptible to malfunction [97], [258]. For instance, a study conducted by Denkinger et al. [258] focused on evaluating the robustness of artificial intelligence (AI) methods, specifically convolutional neural networks (CNNs), to SRAM errors in edge devices. By operating at reduced voltages and employing quantization, they explored ways to enhance efficiency. Their findings revealed that quantization emerged as the most effective strategy, yielding energy savings as high as 61.3%. This was achieved with only a minimal accuracy loss of 7.1%. Further efficiency was gained through voltage scaling, leading to an additional reduction of up to 11.0%. However, these benefits were accompanied by a total accuracy loss of 13.6%.

In IoT nodes, the 6T Static Random-Access Memory (SRAM) cell, known for its compactness and minimal area, is widely used for data processing, as referenced in [266]. However, this cell suffers from several inherent limitations that need to be considered when using it. Inherent limitations of this cell include reliability issues at low voltage conditions, potential conflicts during read/write processes, data can be disturbed during reads, high data retention voltage, and half-select problems [277]. To address these challenges, several design strategies at the cell and architecture levels have been introduced. These strategies focus on reducing power usage in write, read, and leakage states, improving stable data retrieval and write efficiency, and addressing half-select

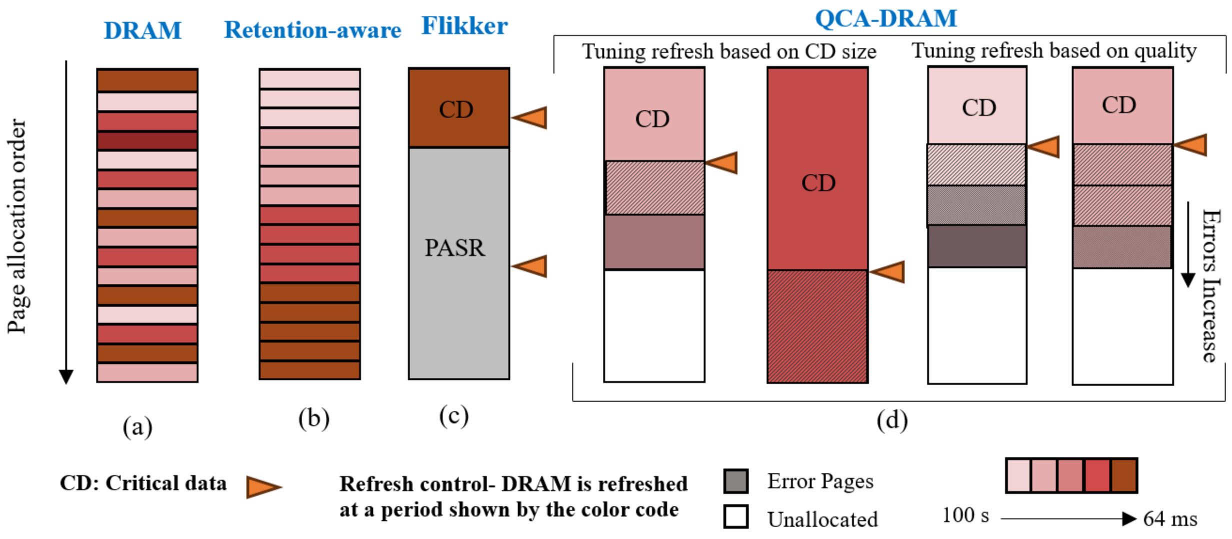


FIGURE 12. Memory allocation scheme (a) represents the baseline DRAM module, (b) module with sorting Data allocated in pages based on different refresh rates, (c) Flikker Approach [12], and (d) QCA-DRAM Approach [259].

problems [266]. One approach to enhancing the performance and stability of the 6T SRAM cell is to increase the number of transistors within the cell. This modification can lead to improved control and functionality, although it may also impact the cell's compactness [266], [277]. A novel approach to reducing read and hold power in SRAM architecture was proposed by Gupta et al. [29], utilizing a reconfigurable VDD scaling technique (R-VDD). This method significantly minimizes power consumption. To implement this R-VDD scaled architecture, they employed a "data-dependent low-power 10T" SRAM cell (D2LP10T).

4) APPROXIMATE MEMORY BASED ON APPROXIMATE READ/WRITE OPERATIONS

Emerging non-volatile memory based on memristor technology is proposed as the solution for approximate computing, which can balance performance and power consumption. When used for compute acceleration, approximation-augmented processing combines each processor with a tiny amount of controllable associative memory [278]. Emerging STT-MRAM (Spin Transfer Torque Magnetic Random Access Memory) memories, which offer higher density and lower static power consumption compared to SRAM, face challenges of high energy usage in read/write operations. QuARK [279] and Cast [280] are hardware and software approaches introduced for STT-MRAM caches. These approaches allow tradeoffs in reliability for saving energy in the on-chip memory hierarchy of multi-core systems operating approximate applications.

5) MEMORY REDUCTION BASED ON APPROXIMATE COMPUTING TECHNIQUES

AxC techniques for memory reduction are commonly implemented at the design stage, often in conjunction with specific memory handling methods [281]. While these strategies may be tailored to particular applications [282], they necessitate a comprehensive understanding of the data's computation and handling. This requirement can be time-consuming and often serves as a barrier to quick implementation. For instance, memory reduction can be achieved by reducing buffer sizes [282], memory reuse methodologies [283], and/or pruning and quantization approaches [284], [285], with the cost of sacrificing accuracy or throughput. In the context of Approximate Buffer (AxB) techniques, one innovative approach presented in [286] focuses on the reduction of buffer size. This method involves the concatenation of data into buffers using the fixed-point format with a chosen bit-width, where all data within an AxB adheres to the same format. The primary goal of this technique is to minimize the memory footprint, achieving reductions ranging from 27% to 68% in applications such as the full SKA SDP signal processing computing pipeline and wavelet transform. Remarkably, this reduction is accomplished without substantial degradation in output quality. However, it does come with the drawback of requiring manual and labor-intensive Design Space Exploration (DSE). To further simplify this process, an application DSE for buffer-sizing was proposed by [287]. This additional approach aims to reduce the memory footprint while ensuring that the output quality remains above a specified threshold.

6) EMERGING MEMORY DESIGN TECHNOLOGIES PROCESSING- IN-MEMORY (PIM)

PIM is a computing paradigm that enhances data processing efficiency by integrating processing capabilities closer to storage units. Traditional architectures store data in memory and make CPUs to move it between components, leading to time-consuming and performance bottlenecks. PIM integrates processing elements directly into memory cells or controllers, allowing data to be processed in place without transferring it to a separate unit. This results in significant speedup and energy efficiency improvements, particularly for data-centric workloads. There are different approaches to implementing PIM: Processing in DRAM (P-DRAM), Processing in NAND Flash (P-NAND) [288], Processing in 3D Stacked Memory [289], [290], and Near-Memory Computing/Processing (NMC/NMP). Instead of integrating processing into the memory cells themselves, the last approach (NMC) places specialized processing units near the memory, reducing data movement overhead.

Processing-In-Memory (PIM) technology is widely used in image and neural network processing which consist of two main types: analog-based PIM and digital-based PIM. In analog-based PIM, the arithmetic operations can be achieved using resistance networks. In digital-based PIM, the execution the additions and multiplications can be through basic operations like NOR which needs multiple clock cycles. The analog PIM is known for its high speed, but it encounters accuracy problems and demands a significant area footprint to accommodate the required analog-to-digital converter (ADC) and digital-to-analog converter (DAC) interface modules [291]. Byun et al. [291] proposed the analog processor-in-memory filter within a CNN setup, which features a 16 × 4 SRAM, 16 DACs, and 4 ADCs, as depicted in Figure 13. It includes a controller for SRAM, DAC, and ADC timing optimization and power reduction, alongside a main controller overseeing all operations. Inputs flow from the AI controller to the DAC controller, utilizing a charge sharing method. Power efficiency is achieved by activating components only as required.

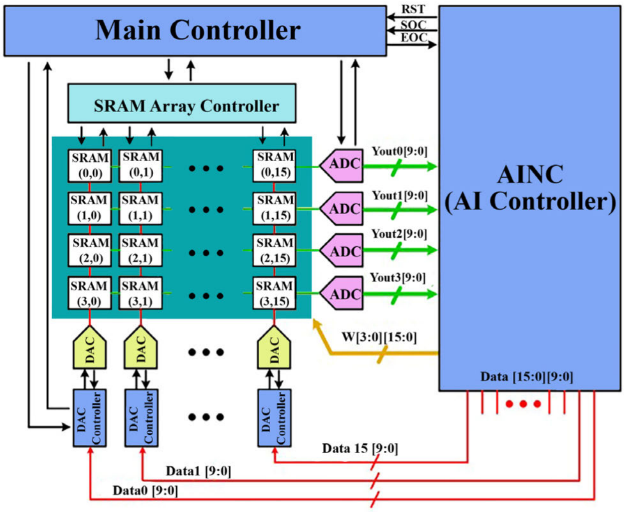


FIGURE 13. The architecture of the Convolutional Neural Network (CNN) implemented within an Analog Processor-In-Memory framework [291].

On the other hand, the digital-based PIM excels in accuracy but experiences higher latency due to the multiple clock cycles required for computations, particularly with multiplications. Through the strategic utilization of the intrinsic parallelism present within application algorithms, the acceleration of the computational process can be effectively achieved.

Memristor, also referred to as Resistive Random Access Memory (ReRAM), is a well-known technology in Processing-In-Memory (PIM) architectures due to its capability of analog computing that speeds up matrix-vector multiplications, which are essential for the systems. Nonetheless, convolutional neural network training using a high-precision backward propagation phase presents difficulties on account of the poor resolution of these analog PIM accelerators. Hai et al. [292] addressed this challenge by introducing a novel hybrid PIM accelerator for CNN training on ReRAM arrays. ReHy combines analog PIM (APIM) for performance in the feedforward propagation phase (FP) and digital PIM (DPIM) for accuracy in the backpropagation phase (BP), offering a comprehensive solution for CNN training. The study reveals that ReHy markedly improves CNN training efficiency, outpacing standard CPU/GPU architectures (baseline) and FloatPIM by 48.8 and 2.4 times, respectively, while also reducing energy usage by 35.1 and 2.33 times compared to each.

This involves integrating processing capabilities into memory storage to reduce the data movement between the CPU and memory. In the fields of image processing and computer vision, convolutional neural networks (CNNs) have emerged as a prevalent tool. While Graphics Processing Units (GPUs) are commonly employed to enhance the acceleration of CNNs, this approach is constrained by the substantial computational costs and memory demands associated with the convolution process. This limitation has led to a focus on approximate computing, a method explored in numerous studies to mitigate computational expenses [267]. The introduction of the Approximate Data Comparison processing-in-memory (ADC-PIM) solution by Choi et al. [289] marks a significant advancement in addressing the performance bottleneck caused by increased memory bandwidth intensity. Implemented in 3D-stacked memory, ADC-PIM strategically compares data for similarity before it is loaded onto the GPU, unlike conventional post-loading methods. This approach results in the transfer of only essential data to the GPU, reducing data movement and computational requirements. The application of ADC-PIM has led to a 43% boost in processing speed and a 32% reduction in energy use, with minimal accuracy loss below 1%.

The limitations inherent in processing-using-DRAM are primarily characterized by its limited support for a limited range of basic operations, including logic functions and addition. Such constraints have impeded the complete exploitation of the capabilities inherent in processing-using-DRAM, thereby necessitating the investigation of strategies to enable the execution of more complex and user-specified operations. Addressing this challenge, Hajinazar [293] proposed SIMDRAM, an extensive framework explicitly crafted to empower processing-using-DRAM that supports complex functions and efficiently handles sophisticated and user-defined functions without hardware changes. They evaluated its performance, showing its superiority over traditional CPUs and GPUs in throughput and energy efficiency, especially with 16 DRAM banks. SIMDRAM performed well in real-world applications with minimal overhead. This marks a significant advancement in processing-using-DRAM technology.

Within the field of Processing-In-Memory (PIM) or In-memory computing (IMC), the predominant focus of research has been the optimization of energy efficiency, specifically within a limited voltage range. This concentration on a narrow voltage spectrum has consequently restricted the applicability of IMC in scenarios characterized by dynamic workloads, where optimization across a wide dynamic voltage range (WDVR) is necessitated. In response to this limitation, a recent innovation has been introduced by Zhang et al. [294]. They have implemented a novel IMC-based Binary Neural Network (BNN) accelerator. This innovative development addressed a previously unmet need within the IMC domain by supporting energy-efficient operations over a broad voltage range.

Processing-in-memory (PIM) represents a paradigm shift in computing architecture that takes advantage of the distinctive physical characteristics of emerging memory systems to boost data processing. These systems include resistive random-access memory (ReRAM), spin-transfer torque magneto-resistive random-access memory (STT-MRAM), and phase-change memory (PCM) [295]. The principal merits of PIM lie in its ability to minimize data movement and reduce latency. The inherent characteristics of Processing-In-Memory (PIM) architectures significantly enhance performance and energy efficiency, especially in tasks that are data-intensive. However, the adoption of PIM is not devoid of challenges. These challenges include heightened design complexity, the imperative of efficient thermal management, and the need to maintain data integrity. Ongoing research in the development of advanced PIM circuits and systems remains a key area of focus, with potential for continued innovation in the domain.

7) APPROXIMATE CONTENT-ADDRESSABLE MEMORIES

In the field of Content-Addressable Memory (CAM), this memory system is notable for enabling data retrieval by content instead of location, enhancing parallel search capabilities essential for high-speed, memory-intensive systems. CAM's adaptability is evident in its application across network routing, digital signal processing, and microprocessor design. Recent advancements in CAM design have focused on improving efficiency in comparison-driven tasks. However, challenges remain in creating CAM systems that are cost-effective, energy-efficient, and capable of similarity searches. The use of approximate CAM is limited by factors like similarity, accuracy, speed, complexity, and cost. The exploration of Approximate Content-Addressable Memory (CAM) in computer memory systems reveals significant benefits and drawbacks. Its rapid associative searching capabilities are advantageous for applications like network routing and data retrieval.

However, CAM faces challenges including high costs, power consumption, limited scalability, and issues with data integrity. Additionally, its read and write speeds may not align with conventional RAM, its design complexity demands careful implementation, and its static nature complicates data updates, potentially leading to higher latency during certain write operations [296]. Despite its limitations, CAM is valuable for rapid associative searches, but a thorough analysis of its trade-offs is crucial for its effective integration in computing systems. Yinjin et al. [297] introduced CARAM, a novel hybrid PCM and DRAM primary memory system, to address Phase-Change Memory's (PCM) limitations like slow memory write speed and limited robustness, despite its high read throughput and low standby power. CARAM, addressing the challenges of limited primary memory capacity in modern DRAM-PCM combinations, improves memory efficiency through deduplication, line sharing, and optimized memory use. It reduces write traffic and duplicate line writes, thereby enhancing PCM wear-leveling and expanding memory capacity. CARAM also maintains high data access performance, which is essential for memory system optimization. Experimental results demonstrate CARAM's effectiveness, showing a 15%–42% reduction in memory usage, a 13%–116% increase in I/O bandwidth, and 31%–38% energy savings compared to existing hybrid systems. In conclusion, CARAM marks notable progress in memory technology, addressing PCM challenges effectively through its innovative design and deduplication strategy, making it a promising area for future exploration.

In contemporary computational systems, Hardware Search Engines (HSEs) represent a paradigm shift from traditional software search algorithms, offering enhanced location access and data association capabilities. Hardware Search Engines (HSEs), particularly Content Addressable Memory (CAM), mark a significant advancement in computational systems, offering improved data retrieval and association. However, CAM's high energy use, especially in cells and matchlines during searches, poses a challenge, notably in the energy-efficient multi-port CAM used in modern superscalar processors [298]. To overcome this, research has focused on low-energy alternatives like precharge-free CAM, which balances speed and power efficiency in associative memory [298]. Additionally, innovations include high-speed, energy-efficient single-port CAM designed for dual-port functionality, improving search performance, and addressing multi-port CAM limitations [299].

8) FRAMEWORKS AND SIMULATORS FOR APPROXIMATE MEMORY

An important role for approximate memory may be found in error tolerant applications, where sacrificing perfect accuracy in data processing in favor of saving energy is acceptable. It is possible to introduce probabilistic errors into read/write access in approximate memory. In most cases, energy-saving circuitry or architectural changes (such as reduced refresh rates or reduced voltages) are at blame for these malfunctions. Since the degree of error that may be accepted varies from application to application, the capacity to simulate these systems is crucial [271], [300]. Through simulation, one may examine an application's behavior and test its robustness against real-world error rates, thereby identifying the optimal trade-off between reduced energy use and improved product quality. Menichelli et al. [271], Stazi et al. [300], and Yayla et al. [301] proposed emulators to reveal the effects of errors introduced by approximate memory circuits and architectures on the hardware platform and software. Yarmand et al. [302] introduced a methodology for identifying suitable approximation degrees for approximable memories within a memory hierarchy for executing error-tolerant applications.

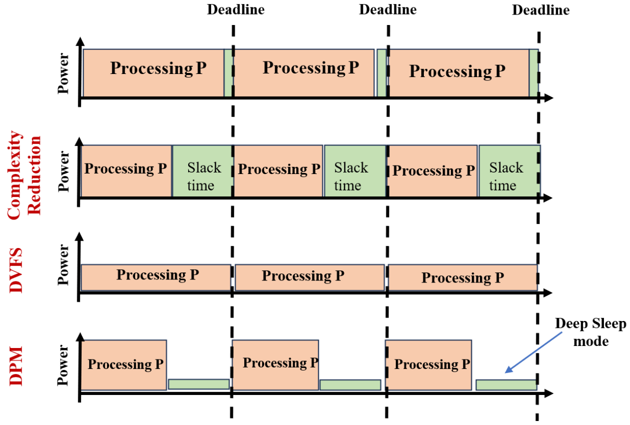


FIGURE 14. Optimization of slack intervals for energy efficiency in real-time system operations [305]. (a) Preliminary computation, (b) DPM technique, (c) DVFS technique.

B. VOLTAGE-FREQUENCY-POWER MANAGEMENT TECHNIQUES

One of the main trade-offs in system-level approximate computing is between the accuracy of the computation and its performance (i.e., speed and energy consumption). In general, increasing the level of approximation can lead to faster and more energy-efficient computation, but it can also reduce the accuracy of the results. Reducing processing complexity in real-time systems provides for more idle (slack) time. The slack time, as shown in Figure 14, is the period between the task's end and its deadline. Exploiting time slack refers to utilizing periods of idle time or low activity in a system to reduce power consumption without compromising performance [305]. Voltage-Frequency-Power Management Techniques are strategies used in electronic systems, particularly in processors, to optimize power consumption and performance. These techniques dynamically adjust the operating voltage and frequency of a system based on the workload, power budget, and thermal conditions. There are some approaches that can be used to exploit time slack and reduce power consumption: Dynamic Voltage and Frequency Scaling (DVFS), Thermal Design Power Management (TDP), Dynamic Memory Management (DMM), Dynamic Power Management (DPM), Task Migration, Adaptive Voltage Scaling (AVS), Frequency Scaling, Voltage Scaling, Clock Gating, Power Gating, Energy-Efficient Scheduling, Near-Threshold Voltage (NTV) Operation, Sub-Threshold Operation, etc. These techniques are energy-efficient approaches at the architecture or system level and have been widely adopted in the Internet of Things (IoT). We will discuss shortly some of these techniques:

1) DYNAMIC VOLTAGE AND FREQUENCY SCALING (DVFS)

DVFS is a technique where the processor's voltage and frequency are dynamically altered based on the according to the workload. When the workload is low, the voltage and frequency can be scaled down to reduce the power consumption [306]. This technology is most effective in dynamic power environments and is widely supported by chip manufacturers, often referred to as "turbo mode" in some contexts.

2) VOLTAGE OVERSCALING (VOS)

VOS is a method that reduces the supplied voltage of circuits to improve energy efficiency. This can lead to increasing the computation errors or failures due to insufficient voltage provided to the transistors to switch states robustly. To balance the energy gains with reliability, systems might need error management strategies. VOS is especially useful in energy-sensitive devices like battery-operated gadgets or IoT sensors, where longer battery life is crucial.

3) DYNAMIC POWER MANAGEMENT (DPM)

DPM is a technique that involves dynamically adjusting the power consumption of a system based on the workload. This can be done by selectively turning off or reducing the power to different components of the system [307]. In idle time, the system enters a deep sleep state. During this state, the total energy can be dramatically reduced using power-gating and clock-gating.

4) DYNAMIC MEMORY MANAGEMENT (DMM)

DMM is a technique used to optimize the memory usage by dynamically allocating and deallocating memory of a system based on the workload. For example, a portion of the memory can be turned off when it is not being used to save power [308]. This technique is particularly useful in systems with varying memory requirements and limited memory resources, such as embedded systems and mobile devices.

5) DYNAMIC THERMAL MANAGEMENT (DTM)

DTM are technique used to manage the heat generation of a system [309]. They monitor the temperature of the system and dynamically adjust the voltage, frequency, or workload distribution to prevent overheating. This can include techniques like thermal throttling, where the system reduces its

TABLE 3. Comparative analysis of various approximate memory implementation strategies.

Ref/Year	Platform	Design Approach	Application	Improvement	Quality loss
[279]/2017	Gem5+ STT-MRAM cache+ Multicore	Different levels of reliability for different cache +Approximate Read/write operation	recognition, mining and synthesis (RMS) benchmarks	40% write energy saving	Acceptable
[259]/2017	Altera Stratix IV GX FPGA-based Terasic TR4-230 with 1GB DDR3 DRAM + µC/OS-II	Refresh rate reduction	5 machine learning and 3 image processing algorithms	73% reduction in DRAM refresh power	Lossless-7%
[280]/2019	Gem5+ STT-MRAM cache+ Multicore	Different levels of reliability for different cache + full-Approximate write operation	Image processing+ Network+ Security+ Financial analysis	energy savings full-approximate, 57%+ mixed-criticality, 34%+ full-accurate applications 21%	Acceptable
[256]/2020	Stratix-IV FPGA + Intel UniPHY-DDR3 memory controller + NIOS-II processor + Hynix DDR3 DRAM or LPDDR3 DRAM or STT-MRAM	Approximate memory Compression	8 machine learning benchmarks applications	Energy reduction 1.18x DDR3 DRAM 1.52x LPDDR3 DRAM 2.0x STT-MRAM Execution time reduction 5.2% DDR3 DRAM 5.4% LPDDR3 DRAM 9.3% STT-MRAM	0.3%
[268]/2020	PC-Linux Virtex 7 VC707FPGA DDR3-D RAM	Bit Truncation + Refresh rate reduction	Deep learning-AlexNet, VGGNet, GoogLeNet	Refresh Energy saving 69.1%+ Total energy saving 26%	negligible
[273]/2021	Xilinx Zynq XC7Z045FFG900-2 +TSMC 28nm	DCT +Quantization + Sparse matrix compression	CNNs	403GOPS peak throughput and +1.4x~3.3x interlayer feature map reduction +2.16 TOPS/W energy efficiency	0.18%-0.45%
[303]/2022	Xilinx Artix 7 (XC7A35T-1C) Python API	Different levels of reliability for different cache + full-Approximate write operation	JPEG encoding + KNN	write energy saving ~47.5%	<5%
[262]/2022		decrease the DRAM supply voltage+ quantized weights to reduce the DRAM access energy	Spiking Neural Networks (SNNs)	84.9% of DRAM energy saving + 4.1x speed-up of DRAM data t	BER ≤ 10^{-3}
[272]/2023	Simulation DRAM and SRAM	encodes the image to effectively reduce the number of bit-'1' in the original pixel data	Discrete Cosine Transform (DCT) +quantization +inverse quantization and inverse DCT (IDCT)	39.8% power reduction for DRAM 25.9% write power reduction for SRAM	average 3.36 dB losses in (PSNR)
[304]/2023	Xilinx ZC702 FPGA	Model compression through data quantization on convolutions	Lightweight and Energy-Efficient Deep Learning	1.25× and 4.27× smaller logic and BRAM size, respectively 10.37× reduction in power consumption at 100MHz	93.1% accuracy

performance to decrease heat generation when it detects that it's getting too hot.

6) ADAPTIVE CLOCKIN

This technique involves adjusting the clock frequency of a processor based on the workload [310]. For example, the clock frequency can be reduced during periods of low activity to save power [311]. Li et al. [310] introduced a rapid and power-saving SNN processor that supports online learning. The researchers used various techniques, such as adaptive clocking and event-driven to reduce the power consumption and accelerate computation.

7) NEAR-THRESHOLD VOLTAGE (NTV)

This technique offers significant energy efficiency improvements by operating processors close to the threshold voltage of the CMOS transistors [312]. While this approach reduces energy consumption, it also presents challenges such as increased latency and sensitivity to transistor variability. Techniques like massive parallelism and temporary voltage boosts are proposed to mitigate these issues, and advanced semiconductor technologies like finFETs help reduce variability concerns. Operating near this threshold minimizes energy consumption while still maintaining a higher degree of reliability and lower error rates compared to VOS. NTV strikes a balance between energy efficiency and computational reliability. NTV computing requires careful design and technological choices to fully harness its energy-saving potential. NTV is particularly suited for IoT applications, such as wireless audio hearables, which require continuous operation but are not necessarily at full performance all the time [312].

8) TASK MIGRATION

Task migration involves moving tasks from high-power devices to low-power devices when the high-power device is not being fully utilized. For example, tasks that are not compute-intensive can be moved from a CPU to a low-power GPU [313].

By utilizing these approaches, systems can reduce power consumption during periods of idle time or low activity without impacting performance. D'Agostino et al. [314] provided an encouraged information about energy-effect computing in hardware and software.

C. APPROXIMATE PROCESSORS

As computing tasks become increasingly complex, there's a rising demand for new paradigms like approximate computing that enhance efficiency. However, the majority of existing hardware-based approximation solutions have been tailored to specific applications or limited to smaller computing units, necessitating significant engineering work for full system integration [315]. Approximate processors and accelerators are integrated approximate computing units consisting of the co-design of hardware and software. They were designed to enhance computational efficiency by allowing for controlled inaccuracies, which are particularly useful in error tolerance applications.

Research interest is growing in ARM processors, which, due to their low-power architecture and supporting by various tools., are prevalent in mobile devices. Furthermore, there are available open-source instruction set architectures (ISA) for processors, which are represented by open, royalty-free RISC-V architectures, supported by major tech firms [315], [316]. Aponte-Moreno et al. [316] proposed a fault tolerance approach to reduce the execution time by using approximate computing at the software level. The researchers used the ARM and RISC-V microprocessor architectures for testing the proposed approach. In another work, Baroughi and his colleagues [315] introduced AxE, the first general-purpose, heterogeneous RISC-V MPSoC platform that combines exact and approximate cores. This multiprocessor was supported by the capability of hardware approximation exploration across various applications through software instructions. The proposed task mapping method tested on AxE achieved a 32% speed-up and 21% energy savings while maintaining 99.3% accuracy across three mixed workloads. However, MPSoC architectures are becoming increasingly popular for demanding workloads in low-power devices like wearables and IoT sensors due to their high performance and exceptional QoS. Therefore, Ali et al. [307] introduced a comprehensive review of MPSoC architectures and explored that scheduling approaches and voltage-frequency-power management techniques are the most commonly used to reduce power consumption in MPSoC.

The growth of IoT has led to an increase in demand for low-cost, resource-constrained devices that have the capability of power budgets. To increase these capabilities, we need to plan new approaches, like approximate computing techniques, to build a new generation of low-power IOT devices. Therefore, Taştan et al. [317] proposed an approximate IoT processor using the RISC-V ISA, which was designed specifically for machine learning tasks like classification and clustering. The proposed processor achieves up to 23% power savings in ASIC implementations, maintaining over 90% top-1 accuracy on trained models and test datasets. The integration of IoT in smart cities has necessitated advanced solutions for processing mixed workloads, combining real-time data with historical records for enhanced analytics. Jawarneh et al. [318] introduced SpatialSSJP, an adaptive system that efficiently manages stream-static joins, optimizing for Quality of Service (QoS) and geo-statistical accuracy. SpatialSSJP was implemented on Spark Structured Streaming and tested on large datasets. Consequently, SpatialSSJP showed significant performance improvements over existing methods and achieved high accuracy levels, with notable gains in optimal scenarios.

Deep learning tasks require optimized memory bandwidth due to their intense resource and memory requirements. The requirements make them suitable for parallel computing architectures like TPUs, which feature deeply pipelined networks of processing elements for efficient dataflow and high performance [17], [319]. Google's Tensor Processing Units (TPUs) are specialized ASICs that accelerate machine learning by using less precise formats like bfloat16 instead of 32 bits floating-point format significantly cutting computation time and memory use while preserving accuracy for many task [17]. TPUs were used to implement the NN applications (MLPs, CNNs, and LSTMs) in datacenters. Elbtity et al. [319] proposed an approximate tensor processing unit (APTPU) consisting of two key components: approximate processing elements (APEs) with low-precision multipliers and approximate adders, and pre-approximate units (PAUs) that pre-process operands for the APEs within the APTPU's systolic array. However, Systolic array DNN accelerators are known for their cost efficiency but struggle with high energy use, limiting their use in low-power devices. Approximate computing offers a solution at the expense of slight accuracy losses, which could, however, make DNNs more prone to disturbances like permanent faults, already a concern in accurate DNNs especially in critical applications like autonomous driving where reliability is paramount. Ensuring the reliability of DNN hardware often requires extensive fault injection testing [320], [321]. Siddique et al. [320] and Ahmadilivani et al. [321] addressed the challenge of exploring approximation and fault resiliency of DNN accelerators. Siddique et al. [320] conducted a detailed analysis of fault resilience and energy consumption in various AxDNNs on a layer and bit level, using the Evoapprox8b signed multipliers. Their findings reveal that a single permanent fault in AxDNNs could result in as much as a 66% drop in accuracy, while the same fault might cause just a 9% accuracy reduction in a conventional DNN accelerator. At similar work, Ahmadilivani et al. [321] focused on enhancing DNN accelerators' fault resilience and approximation, using AxC arithmetic circuits for error emulation and a GPU-based framework for swift evaluation. It also delves into analyzing fault propagation and masking in networks.

TPUs enhance deep learning efficiency through optimized dataflow and low-precision support, but at the cost of potential accuracy drops and fault vulnerability. Advancements in approximate computing show promise in mitigating these issues, crucial for applications demanding high performance and reliability.

There are many interesting approximate processors and accelerators that have gained importance in different applications, such as energy-efficient IoT devices, real-time video processing, and machine learning inference tasks, where trade-offs between precision and performance can yield significant benefits. Some of these processors and accelerator will be mentioned in applications section.

VIII. CIRCUIT-LEVEL APPROXIMATIONS

The concept of approximating logical functionality is sufficiently generic that it is applicable to both software [148] and hardware [15]. When multiplication and division based on logarithms, were first being developed, the early 1960s marked the beginning of the acceptance of approximation computing [322]. The considerable research interest in designing approximate circuits has been propelled by the substantial potential for power consumption reduction. Approximate computing focuses primarily on arithmetic units, e.g., adders and multipliers, at the level of custom hardware, as these constitute the fundamental components of numerous error-tolerant applications and all computations. The current research in VLSI design focuses heavily on real-time DSP and machine learning for applications like surveillance and wearable technology. These areas need quick, accurate data analysis for pattern recognition. IoT and edge processing emphasize immediate, local processing over cloud computing due to latency and connectivity issues. However, local processing requires solutions that are low-power, accurate, fast, and cost-effective. Many algorithms in this field use basic functions like trigonometric and logarithmic functions. Calculating transcendental functions on computers typically involves software, leading to delays. Thus, hardware implementations have become vital due to their performance benefits over software. Numerous publications detail these hardware implementations for arithmetic units and elementary functions.

A. APPROXIMATE ADDERS

Approximate computing is an emerging paradigm that aims to optimize power consumption, area, and delay. This approach involves the strategic redesign of a system's logic circuit to allow for controlled imprecision in calculations by allowing for some degree of inaccuracy in the results. The computing error is generally undesirable, but there are some applications that can tolerate imprecise computation.

A critical focus within this domain has been on the design of arithmetic circuits, particularly adders. Adders represent a fundamental element in these arithmetic units that have received special attention from researchers and play an important role in error-tolerant applications. Accurate adders may suffer from high delays, complexity, or power consumption. A Ripple Carry Adder (RCA) works by adding the bits of the two numbers one by one, starting from the least significant bit (LSB) to the most significant bit (MSB) in a chain-like manner. The critical path of an adder is defined by its whole carry chain. Although the RCA is relatively slow, it is a simple and commonly used circuit for small addition operations. For larger additions, other types of adders, such as carry-lookahead adders or carry-select adders, are used, which have faster carry-propagation but suffer from overhead and higher power consumption. Approximate computing is becoming increasingly important as the demand for more efficient computing grows, as it allows for the same task to be completed with fewer resources. For computationally intensive processes like machine learning, this speeds up and improves outcomes.

In digital circuit design, the approximation computing technique provides a potential solution for decreasing power, area, and latency. This is accomplished by redesigning the logic circuit using many different implementing approaches that permit a decrease in accuracy [15]. Approximate computing can be applied to circuits at different levels: transistor level, logic gate level, and architecture level. In the literature, a wide variety of approximation adders [15], [16], [323], [324], [325], [326], [327], [328], [329] have been reported: segmented adders, where an n-bit adder is partitioned into k-bit subadders [15], [323], [324]; an approximate full adder, in which a single full adder can be approximated at the logic or transistor level [16], [325], [326]; carry-select adders, which are multi-stage subadders are utilized [327], [328]; and speculative adders, reimagining traditional designs, optimize performance by bypassing the infrequently used critical path [329], [330].

A more in-depth look at the many different kinds of approximation adders (and, more generally, approximate units) reveals that the many techniques now in use adopt one of three different methodologies towards inaccuracy [15], [331]: 1) insignificant and frequently errors, 2) significant and improbable errors, or 3) a combination of 2 and 3. In the first methodology, the designers engineer and approximate the lower significant bits of the arithmetic unit to obtain a small magnitude of errors that are frequent. The quality of the application is not substantially diminished by these errors, as they are overshadowed by the system's intrinsic truncation and noise errors. For instance, Optimized Feedback Lower-part Constant-OR Adder [15] and Lower-part OR Adder (LOA) [325] follow the first methodology. In the second methodology, the designers engineer the more significant bits of the arithmetic unit to appear as infrequent errors but large in magnitude. The thinking behind this is that applications are resilient enough to recover from occasional errors. Examples of this methodology are the Almost Correct Adder (ACA) [323], the Feedback Approximate Adder (FAA) [15], and the Generic Accuracy Configurable Adder (GeAr) [324]. In the third methodology, the designers engineer the arithmetic unit, including the two previous methodologies, to enhance the existing approximate adders, for example, the enhanced [15] and hybrid [331] approximate adders. Combining the best features of both previous principles is the preferred and sometimes necessary choice for real-world applications. The primary purpose of this work is to survey the current research and development status of various approximation approaches.

1) APPROXIMATE FULL ADDER AT THE TRANSISTOR LEVEL

As the one-bit full adder (OBFA) is the primary circuit for implementing an n-bit adder, the fundamental arithmetic circuit of any digital system, it plays a crucial role in the calculation process [332]. A full adder is a type of combinational logic circuit designed to add together three bits: two input bits and a carry bit from a previous stage. But an approximate hybrid full adder is a modified version of a full adder that uses a combination of two or more logic styles together to reduce power consumption and area overhead while maintaining reasonable accuracy [332].

There are seven different full adder cells based on static logic styles, which are: the Complementary Metal-Oxide Semiconductor (CMOS), Complementary Pass-transistor (CPL), pass-transistor logic (PTL), single-rail pass transistor (LEAP), double pass transistor (DPL), pseudo-NMOS (Ratioed logic), gate diffusion input (GDI), and hybrid full adders. As a result, a great deal of thought and care must be invested into selecting a particular topology of OBFA at the transistor level and designing the associated circuit to affect the overall performance and energy of the system. Also, we recommend to read these papers [332], [333].

The CMOS Full Adder is a widely used circuit for binary addition of two 1-bit numbers, employing NMOS and PMOS transistors. This logic style is widely used in digital circuits due to its high noise immunity, low power consumption, and reliable operation at low voltages. Despite its benefits, it has drawbacks like the presence of bulky PMOS transistors, increased transistor count, high input impedance, and high delay. The CMOS full adder is usually designed using multiple stages of CMOS inverters and transmission gates [334], [335]. For example, a 14-transistor (14T) CMOS complete adder cell which boasts a 50% reduction in the threshold loss issue and an increase in the output voltage swing, but has significant delays.

Pass-Transistor Logic (PTL) is a digital logic circuit design method that uses pass transistors to implement logic functions. It offers greater efficiency and energy savings compared to traditional static CMOS logic. PTL achieves smaller circuit sizes and lower production costs due to fewer transistors in its gates, leading to less power use and reduced propagation delays. However, PTL can face signal degradation from parasitic capacitances, especially in complex circuits, which may impact performance.

Hybrid logic circuits offer a balance between speed and power consumption, attracting increasing attention due to the proliferation of hybrid-based topologies in recent years [332], [333], [336], [337], [338], [339], [340]. The purpose of this review is to provide the designer with a sample but effective method for discovering which topologies are optimal in terms of power consumption, throughput, or a mix of these metrics.

This survey considers hybrid architectures and includes the most current topologies. Several requirements are traded off to attain distinct benefits in full adder designs. In this context, the number of transistors, delay time, power consumption, and output voltage swing are crucial [336]. In contrast to the typical CMOS full adder, which requires 24 transistors, the Mirror adder requires 28 transistors. Both provide precise output voltage levels, which results in a large area and significant energy use. In order to reduce the number of transistors required while maintaining the entire output voltage swing, several different designs have been proposed. Unfortunately, many designs have reduced the number of transistors to achieve low power consumption; however, this comes at the expense of a diminished output voltage swing [337], [341]. For example, the proposal in [337] employs just 8 transistors, based on two XNOR gates, each with three transistors and an inverter, making it a simpler architecture in terms of transistor count. However, this lower-transistor-based circuit has an issue with threshold loss, which causes the logic voltages '1' and '0' to be slightly off from Vdd and 0, respectively. Many designs have implemented solutions to this problem, often by expanding the allowed range of the output voltage or by increasing the voltage at each of the outputs. When operating at low supply voltages (i.e., Vdd=1.8 volts), the deteriorated output might lead the circuit to give incorrect outputs for certain input combinations, making it all the more crucial to minimize threshold loss [336]. In order to reduce the threshold loss problem and increase output voltage swing, Hassani et al. [336] proposed a 16-transistor accurate full adder (16T FA) design using a 10T CMOS FA. This design serves as a foundational element for proposing a Lower-part-OR (LOA) approximate adder [325]. This design reduces power consumption by 53% compared to the LOA adder, at the cost of a 12% drop in accuracy and increased delays.

Diverse methodologies for designing XOR-XNOR circuits have been published in recent years and are used to prevent glitches in the complete adder's output nodes. Kandpal et al. [340] proposed a 10-transistor XOR-XNOR circuit to provide full swing outputs with improvements in power consumption and performance. The outcomes show the power delay product (PDP) is 7.5% higher than that of the available XOR-XNOR modules in 2020. They used this XOR-XNOR circuit to design a 1-bit full adder (OBFA) called high-speed hybrid full adder design-4 (HSHFA-D4) using 20 transistors. The results show a 28.13% improvement in terms of PDP compared to other architectures. Another hybrid full adder design is called scalable low-power hybrid full adder (SLPHFA) which proposed by Hasan et al. [339]. Without resorting to an intermediary propagate signal, the carry signal and summing operation are generated via a novel AND-OR module and two XOR modules, respectively, utilizing transmission gates and CPL logic styles. Both HSHFA-D4, SLPHFA and HFA-22T [342] characterize by no driving capability. Figure 15 shows different classical and hybrid FA adders based on various logic styles.

The Gate Diffusion Input (GDI) method is an efficient technique for designing full adders, reducing transistor count and power consumption while offering compact design, but faces limitations in voltage scaling and operating speed. The GDI-10T full adder is proposed by Nirmalraj et al. [343], which consists of one 4T XOR gate and two 2:1 multiplexers. Combining the GDI and PTL logic styles produced a novel twist on the conventional full adder circuit; as a result, the design only required 10 transistors to perform addition.

The shrinking of MOSFETs leads to challenges like increased leakage current and higher manufacturing costs. To address these, feature size scaling in digital circuits is key for reducing power-delay product (PDP) and power consumption. Carbon nanotube field-effect transistors (CNFETs), including p-type and n-type, are emerging as alternatives to MOSFETs, offering higher switching speeds and similar mobility for equivalent sizes [344]. There is a substantial amount of published material that describes the circuit implementation using CNFETs. For example, Bhargav et al. [344] proposed 10T and 13T approximate adders, based on 32 nm CNFET technology.

2) APPROXIMATE FULL ADDER AT GATE LEVEL

In an effort to lessen the critical path and hardware complexity of precise adders, a number of approximation approaches have been developed. Approximate adders are based on the idea that they can complete the addition faster than precise adders by breaking the carry propagation chain. This kind of approximate adder separates the adder into two separate segments: an exact adder is used for the higher significant segment, while approximate full adders are used for the less significant ones. This group has a basic truncation method [15], [16], [325], [345], [346]. The Lower-part OR Adder (LOA) [325] is the most well-known design in this class, which proposed in 2010, where consists of two subadders: an accurate subadder and an approximate subadder. The higher significant (accurate) subadder achieves the error-free calculation by using a conventional precise adder like the ripple carry adder (RCA) or the carry-lookahead adder (CLA). The lower significant (approximate) subadder is constructed by only OR gates to approximately obtain LSB summations. Moreover, the accurate subadder's precision is enhanced by using the carry from the MSB input pair of the inaccurate subadder through AND gate. However, the precise subadder size determines the LOA critical path delay, and LOA has positive and negative errors.

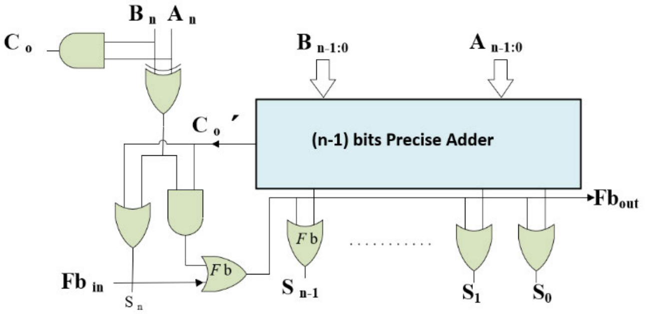
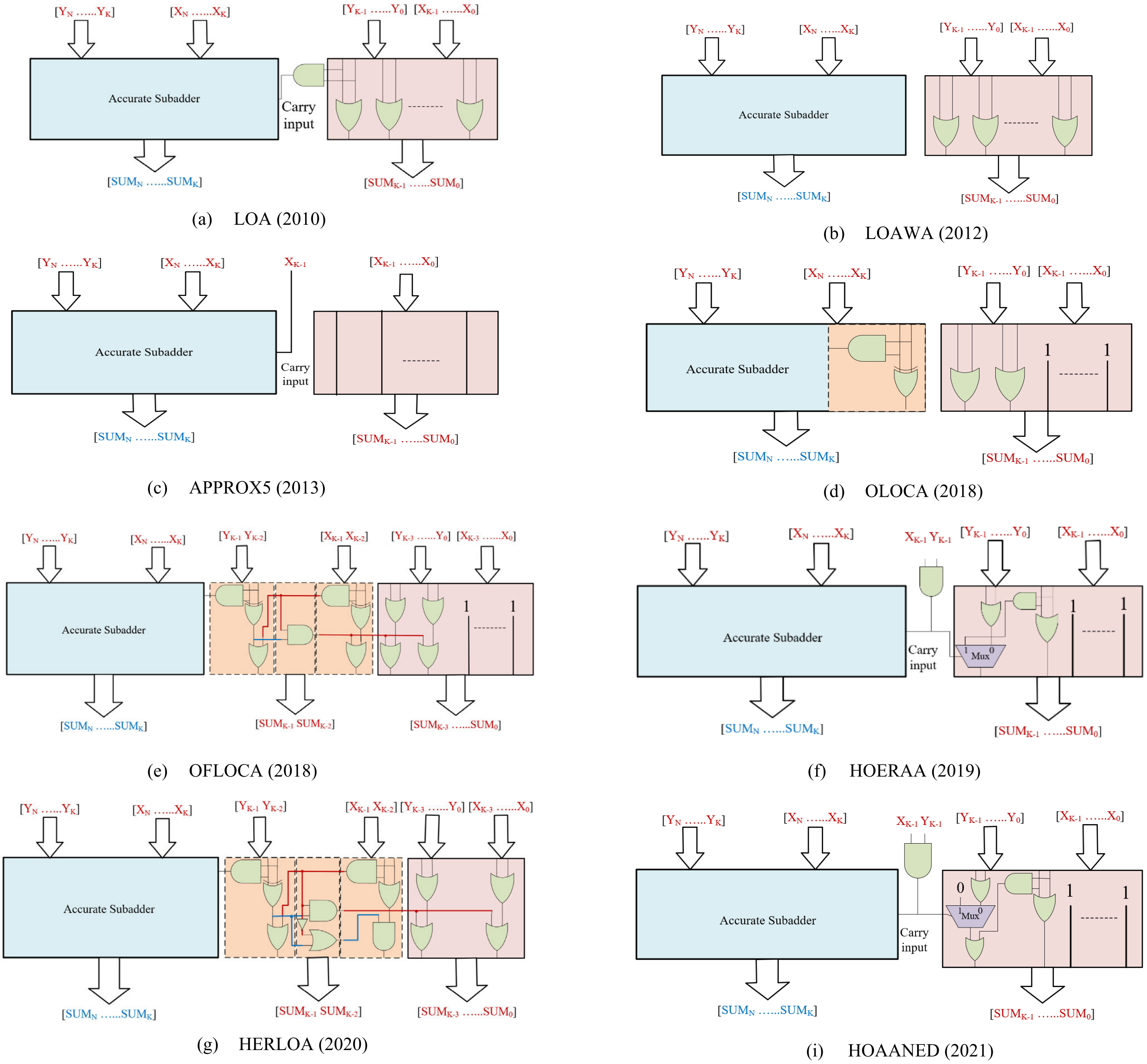


FIGURE 15. Architecture of Feedback approximate adder cell [15].

In 2012, Albicocco et al. [347] proposed LOAWA, a modified version of LOA adder, by removing AND gate that provides a carry from inaccurate part to accurate part, and this design has only positive errors. After a year, Gupta et al. [326] proposed an approximate adder, APPROX5, where an inaccurate part is composited by one of the input pairs. In 2018, Dalloo et al. [16] studied, analyzed, and systematically designed the approximate adder called Optimized Constant Lower-part OR Adder (OLOCA), where the inaccurate part is constructed by ones and OR gates. Dalloo et al. [15], [16] showed that the number of OR gates must not be less than two. In the same year, Dalloo [15] systematically designed an approximate adder segment (cell) called Feedback Approximate Adder Cell (FAA), which constructs an accurate adder segment with a unique logic circuit as shown in Figure 15. This cell has the capability of smoothly error-correction, which means it composites partly errors produced by the inaccurate subadder through returning carry feedback to the inaccurate subadder. The cell feeds an accurate subadder by carrying the MSBs of the inaccurate subadder. The authors pointed out that the cell can be repeated and connected through OR gate. Furthermore, Dalloo [15] modified OLOCA, called OFLOCA, to construct the inaccurate subadder with ones, two OR gates, and two bits of the cell. The cell can be repeated to lessen the critical path. OFLOCA outperforms the state-of-the-art architectures such as OLOCA, LOA, etc.



(a) LOA (2010)

(b) LOAWA (2012)

(c) APPROX5 (2013)

(d) OLOCA (2018)

(e) OFLOCA (2018)

(f) HOERAA (2019)

(g) HERLOA (2020)

(i) HOAANED (2021)

FIGURE 16. Block schematics of some approximate gate-level adders, (a) LOA [325], (b) LOAWA [347], (c) APPROX5 [326], (d) OLOCA [16], (e) OFLOCA [15], (f) HOERAA [348], (g) HERLOA [350], and (i) HOAANED [349].

In 2019, Balasubramanian and Maskell [348] proposed a modified OLOCA by using a 2-to-1 multiplexer (MUX21) in the MSB of an inaccurate subadder, and the carry of the MSB of the inaccurate subadder (denoted as C_k) is used to control the multiplexer and feed the accurate subadder. The multiplexer is fed the carries of C_k and C_{k-1} and generates the MSB's sum of inaccurate subadders. This adder is called Hardware Optimized and Error Reduced Approximate Adder (HOERAA), which cannot correct the error that occurs when the carries C_k and C_{k-1} are "01" and the inputs X_k and Y_k are ““X0” or "0X". To partly solve the issue, the same authors [349] modified HERLOA by adding OR gate after the multiplexer, as shown in Figure 16. The modified design is proposed in 2021 as a hardware-optimized approximate adder with a near normal error distribution (HOAANED).

In 2020, Seo et al. [350] proposed an approximate adder called Hybrid Error Reduction LOA (HERLOA) approximate adder. This design is a modified LOA with a similar structure to OFLOCA in using carry feedback to lower significant bits of inaccurate subadder through OR gates and the two-bit feedback cell, but with modifications. Lee et al. [345], 2021, proposed a new approximate adder called Error Reduced Carry Prediction Approximate adder (ERCPAA), which aims to reduce error metrics while increasing cost metrics.

Figure 16 shows the architectures of the aforementioned gate-level approximate adders, where n, k, and n-k refer to the size of the approximate adder, inaccurate subadder, and accurate subadder, respectively. X and Y are the inputs to the approximate adder, but SUM refers to its output.

Furthermore, the Parallel-Prefix Adders (PPA) are among the fastest adders, and their process of binary addition is segmented into three distinct stages [351], [352]: pre-processing, prefix-processing, and post-processing. The pre-processing stage calculates generate and propagate signals, The heart of the PPA, the prefix-processing stage, leverages the prefix operator for accelerated computation of carry. The nature of the prefix operator allows for the flexibility of executing individual operations in any sequence, which has led to the development of various parallel-prefix architectures. Lastly, in the post-processing stage, the sum bits are calculated adding the previous carries and propagate signals. Researchers have paid special attention to being approximated; for example, recently, Rosa et al. [344] proposed approximate parallel prefix adders (PPAs) using proposed approximate prefix operators (AxPOs), which consist of carry operator nodes. They used four well-known PPA adder architectures: Kogge-Stone, Brent-Kung, Ladner-Fischer, and Sklasky, to apply approximate AxPOs. Advanced digital design requires parallel prefix circuits like adders or priority encoders, which conventional design techniques often struggle to balance between area and delay effectively. Therefore, the NVIDIA Applied Deep Learning Research group [352] proposed a reinforcement learning-based method with a specialized environment and representation for efficiently designing parallel prefix circuits. There are interesting review papers [353], [354] on gate-level architectures of approximate adders.

To minimize the critical path and complexity of precise adders, numerous alternative approximation schemes have been proposed. These approximate schemes are the speculative adders such as Carry Cut-Back adder [355], Reverse Carry Propagate adder [356], VASP adder [330], and segmented adders such as Feedback adder [15] and a low-latency generic accuracy configurable adder (GeAr) [324].

In conclusion, the characteristics and performance of segmented and speculative adders diverge significantly from those of approximate full adders. Segmented adders split the carry chain, leading to larger but infrequent errors. In contrast, speculative adders offer high accuracy but at the cost of complex circuits. This creates a trade-off: speculative adders are less favorable due to their complexity compared to segmented and approximate full adders. The design of adders thus requires balancing efficiency with precision.

B. APPROXIMATE MULTIPLIER

Multipliers exhibit high complexity, which tends to consume energy and cause increased delays in computational operations. Multipliers are essential to microprocessors, digital signal processors, and embedded systems. Their applications vary from fundamental filtering operations to advanced convolutional neural networks [357]. This is especially important in large-scale machine learning tasks because convolution operations depend heavily on multiplication-accumulation processes. Consequently, there has been a notable shift in research focus towards developing low-power, high-performance approximate multipliers. This development stems from the need to optimize energy efficiency and processing speed in such tasks, addressing the inherent limitations of multipliers in comparison to simpler, more energy-efficient adders. The operational structure of a multiplier comprises three stages: partial product generation, partial product reduction (accumulation), and final addition. Approximations can be introduced in any of these stages, but the accumulation stage, in particular, is a focus of research for its significant power and delay consumption, highlighting the importance of designing low power and delay approximate multipliers. The Wallace tree, Dadda tree, and carry-save adder array are primary structures for partial product accumulation in multipliers. The Wallace tree uses parallel-operating full or half adders (FAs/HAs) without carry propagation, leading to a logarithmic delay ($O(log(n))$). Its FAs, acting as (3:2) compressors, can be replaced by other compressors, like (4:2), to reduce delay.

The Dadda tree is similar but uses fewer adders. In contrast, the carry-save adder array passes carry and sum signals from one row of FAs/HAs to the next, operating in series, resulting in a linear delay ($O(n)$), which is longer than the Wallace tree's. For example, Sabetzadeh et al. [358] proposed a new approximate multiplier which produced the least significant half of the product using an approximate multiplier with error compensation capability and the other half using an accurate multiplier. The proposed design enhances the energy-delay product by 77% over exact designs and 54% compared to existing approximate designs, on average.

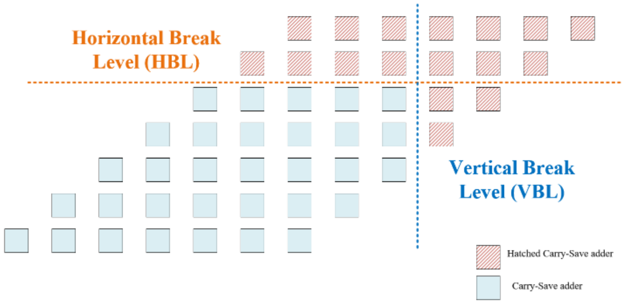


FIGURE 17. The structure of 7 × 7 BAM multiplier [325].

1) TRUNCATED MULTIPLIERS

In the quest for efficient computational operations, the design of approximate multipliers is a key area of focus, particularly for applications that demand a balance between accuracy and power consumption. Among the various strategies employed, truncated multiplication, simplifies operations by discarding the least significant bits of input operands or removing the partial products (AND gates) or Full adder cells, thus reducing the silicon area and speeding up the multiplier, but with a manageable loss in precision. By employing appropriate correction functions, the truncation error can be effectively minimized. For example, Broken-Array Multiplier (BAM) [325] is a variant of truncated array multipliers. This design shares foundational similarities with the conventional array multiplier but introduces a distinctive modification: the strategic omission of Carry-Save Adder (CSA) cells in both horizontal and vertical orientations. This alteration, as depicted in Figure 17, is not arbitrary but is governed by two critical parameters: the Horizontal Break Level (HBL) and the Vertical Break Level (VBL). These parameters determine the specific cells to be omitted, as marked by hatching in the figure. The primary advantage of this design lies in its compact and expedited circuitry, achieved at the expense of precision. Then, Farshchi et al. [359] modified BAM multiplier using booth encoding. Then, Roy and his colleagues realized the needs of computational applications in real-time precision demands through designing approximate and reconfigurable circuits, ensuring power consumption aligns with computational accuracy. Therefore, Roy et al. [360] proposed an accuracy reconfigurable version of approximate Broken-Array booth multiplier. This design incorporates partial error correction by adding sign bits to a Broken-Array multiplier. This new reconfigurable multiplier design significantly reduces power consumption compared to traditional and modern multipliers.

2) COMPRESSOR-BASED MULTIPLIERS

Another approach, compressor-based designs, which stand out for their ability to streamline the accumulation stage. These designs utilize various compressor configurations, such as 7:3, 5:2, 4:2, and 3:2 compressors [361], [362]. Among these, the 4:2 compressor is often favored for its structural regularity, particularly when implemented in cascading configurations. This preference is also reflected in its widespread application in the design of Dadda multipliers [361], [362]. For example, Edavoor and colleagues [362] introduced an innovative 4:2 compressor design. This approximation-based approach yields significant improvements. Specifically, it achieves a 56.80%, 57.20%, and 73.30% reduction in area, power consumption, and delay, respectively. These improvements are in comparison to a conventional, accurate 4:2 compressor. However, this is balanced by an error rate of 25% and a maximum error distance of ±1. The 4:2 compressor efficiently executes four additions at once, enhancing parallelism which in turn minimizes the critical path and dynamic power dissipation. Dornelles et al. [363] proposed two topologies based on CMOS+ gates to decrease the power, area, and delay of the 4:2 compressor.

3) BOOTH ENCODING MULTIPLIER

The ever-growing demand for efficient and compact digital circuits has fueled the development of approximate computing techniques. In the domain of multiplication, Booth multipliers represent a popular choice due to their versatility and ease of implementation. The use of modified Booth encoding significantly streamlines the multiplication of large numbers by reducing partial products. The modified Booth encoding (MBE) can reduce the number of PPs by half [364]. Zhu et al. [365] introduced a novel approach to designing Approximate-Truncated Booth Multipliers (ATBMs). These ATBMs are crafted using a combination of Modified Radix-4 Booth Encoders (AMBEs), Approximate 4:2 compressors (ACs), and a technique of gradually truncating partial products. A key feature of this design is its ability to adjust accuracy levels. This adjustability is achieved by varying the number of AMBEs and ACs incorporated into the system, thereby allowing for a customizable balance between precision and computational efficiency.

However, traditional Booth multipliers suffer from high hardware complexity, limiting their applicability in resource-constrained scenarios. Haider and colleagues [366] addressed this challenge by introducing an innovative approximation approach to enhance the efficiency and reduce the hardware complexity of Booth multipliers while maintaining negligible error rates. The new approach requires only N/4 Booth decoders, reducing the Normalized Mean Error Deviation (NMED) and Power-Area-Product (PAP) in the 16-bit BD16.4 approximate Booth multiplier compared to existing advanced multipliers.

4) SEGMENTED (RECURSIVE) MULTIPLIERS

Approximate segmented (recursive) multipliers offer another way of dividing the multiplication process into smaller multiplier blocks. The simplest method in this category involves using smaller, approximate multipliers to develop larger multipliers, leading to the generation of approximated partial products [367], [368]. The low-power approximate techniques are applied more aggressively to the segments dealing with less significant bits. The research teams focus on developing approximate 2 × 2 or 4 × 4 multipliers, utilizing near-exact half (HA) and full (FA) adders, or alternatively, employing approximate counters or compressors for this purpose. In 2011, Kulkurani et al. [367] proposed under-designed approximate multiplier (UDM) based on the proposed approximate 2 × 2 multiplier block. The approximate 2 × 2 multiplier produced the output error only when the inputs are one where the output is "111" instead of the accurate output is "1001", reducing the accumulating stage. In 2016, Rehman et al. [369] also proposed a 2 × 2 approximate multiplier block with a lower magnitude of maximum error. After a year, Venkatachalam et al. [370] used the statistical analysis to transform the partial products $a_{m,n}$ and $a_{n,m}$, to form propagate $p_{m,n}$ and generate $g_{m,n}$ signals as follows:

$$\begin{aligned} p_{m,n} &= a_{m,n} + a_{m,n} \\ g_{m,n} &= a_{m,n} \cdot a_{m,n} \end{aligned} \tag{1}$$

In comparison, the chances of $g_{m,n}$ being one are substantially lower at 1/16, unlike $a_{m,n}$, which has a higher probability of 1/4. On the other hand, the probability for $p_{m,n}$ to be one is 7/16, exceeding the likelihood for $g_{m,n}$. Using this transform concept, Venkatachalam and his colleagues proposed approximate 4 × 4 multiplier blocks and a 4 × 2 compressor using the proposed approximate half and full adders. Furthermore, they used the proposed blocks and compressor to build higher approximate multipliers, which characterize a high error rate and cost. But Waris et al. [368] achieved similar methodology to transform the partial products to form propagate- and generate-signals and design NOR based two approximate HF (NxHA) and one FA (NxFA) adders. The authors used these adder cells to build two 4 × 4 approximate multiplier blocks (more-approximated (MxA) and less-approximated (LxA) multipliers) and then larger multipliers. Waris and his colleagues designed the multiplier with a lower error rate (approximately half), lower cost, and higher performance than Venkatachalam's design.

5) FPGA BASED DESIGNED MULTIPLIER

FPGA provided high-speed multipliers, which are characterized by their flexibility of reconfiguration and different precision formats to optimize performance for a specific task. FPGA has a limited number of depicted multipliers; therefore, we need to design this operation using FPGA LUTs. The designers will face the challenge of design complexity, and then creating efficient FPGA designs requires specialized knowledge. For example, Ullah et al. [371] introduced a new approximate multiplier architecture tailored for FPGA-based systems, offering a methodical design approach and an accessible online library. This innovation outperforms traditional ASIC-based approximations in terms of area, latency, energy efficiency, and accuracy. Specifically, it surpasses Xilinx Vivado's multiplier IP, showing up to 30% area, 53% latency, and 67% energy improvements with minimal accuracy compromise. The provided open-source library aims to spur further research within the FPGA community, marking a significant shift towards optimized reconfigurable computing.

6) LOGARITHMIC MULTIPLIER

Logarithmic multipliers (LM), especially the base-2 logarithm, offer a highly efficient approach for converting multiplication to addition and shifting operations. They significantly improve the hardware efficiency of error-tolerant applications [372], [373]. The implementation of these multipliers comes with accuracy and design complexity bottlenecks, and it requires a dedicated circuit to compensate for errors and improve both hardware and accuracy. For example, Pilipovi ´c et al. [374] proposed a two-stage approximate logarithmic multiplier that uses less area and energy. Makimoto et al. [375] proposed two-segment piecewise-linear compensation to Mitchell's logarithmic multiplier to improve its accuracy. Yu et al. [376] proposed an approximate LM, named HEALM, that integrates error compensation with mantissa truncation, using a lookup table to enhance accuracy and efficiency.

7) HYBRID MULTIPLIER

Lastly, hybrid techniques that combine two or more of these methods can offer a balanced solution, optimizing both accuracy and power consumption. For example, Ansari et al. [377] proposed and developed a new 4 × 4 and higher approximate multipliers using a combination of booth input encoding and a proposed approximate (4:2) compressor. The proposed design achieved a 52% reduction in the PDP-MRED product and outperformed other similar-accuracy approximate Booth multipliers. Choudhary et al. [378] introduced an automated method for generating approximate circuits with formal worst-case relative error (WCRE) guarantees using Look-Up Tables (LUTs) and SAT-based techniques. The proposed 8-bit approximate multiplier reduced the power consumption and delay by 83.33% and 25.3%, respectively, with only a 1.2 dB SNR degradation in a Finite Impulse Response (FIR) filter.

C. APPROXIMATE DIVIDER

There are many exact algorithms that have been proposed for implementing division operations. Digit recurrence, which is a trusted and exact division algorithm and offers simple logic but faces latency and space inefficiencies, is therefore limiting its use in high-speed applications [379]. The digit recurrence is an iterative algorithm including Rostering, Non-Rostering, and SRT dividers (as a sub-branch of non-restoring). For example, Patankar et al. [379] introduced the exact USP-Awadhoot divider, as a digit recurrence design can be adaptable as restoring or non-restoring and optimized for space-efficient electronic applications.

Designing an efficient divider necessitates using an inexact computation to address the inherent issues of high latency, large area, and significant power consumption in typical traditional division circuits [380]. Piso et al. [381] showed that a 1% improvement in a division circuit block can boost system performance by up to 20%. An approximate divider is a computational unit designed to perform division with a trade-off between accuracy and efficiency and used in various error-tolerant applications, including image processing, machine learning, wearable electronics, etc. Recent research on approximate dividers focuses on finding effective trade-offs by reducing the complexity, such as employing approximate subtraction or reciprocal [382], [383], [384], [385, logarithmic [382], truncating [383], reducing the number of iterations [385], using lookup tables, or applying other approximate techniques). Furthermore, the researchers focus on developing new methods for error analysis and management to minimize errors.

1) FLOATING, FIXED POINTS, AND FPGA DIVIDER

The floating-point divider is a complex component in arithmetic-heavy digital designs, categorized into combinational and sequential types, with a focus on the latter. Peter Malik [386] implemented three iterative division algorithms, including Newton-Raphson, Goldschmidt, and combined Goldschmidt and binomial divisions. The key principle of these algorithms is that the implementation of the division operation depends on an inverse process of the multiplication operation, where the denominator is iteratively subtracted from the numerator. The accuracy depends on the number of iterations and the computation complexity of the iteration. Bureneva et al. [387] proposed a fixed point version of Newton-Raphson division. Recently, Ebrahimi et al. [388] proposed RAPID, the tunable accuracy multiplier and divider architectures, customized for FPGAs.

2) TRUNCATED, APPROXIMATE RECIPROCAL AND DYNAMIC ITERATION STOPPING DIVIDERS

Approximate floating-point (FP) multipliers have been extensively explored in recent applications, which overwhelm the study and development of approximate FP dividers, despite their significant utility [382]. We noticed that a number of significant research efforts have been dedicated to developing approximate dividers using different approximate computing techniques. For example, Oelund et al. [382] proposed an approximate floating-point divider using an approximate hardware-friendly reciprocal and iterative logarithmic multiplier. The authors corrected the errors by storing them in a lookup table. The accuracy of this design can be configured in real-time. Vahdat et al. [383] also used the approximate reciprocal multiplied by the truncated value of the dividend for designing the approximate divider. Truncated dividers are a basic approach to approximate division by limiting calculations to a certain number of bits, offering speed and simplicity at the cost of potential errors, which vary by application. Behroozi et al. [385] introduced SAADI, an approximate divider design which boosts energy efficiency in error-tolerant applications by allowing dynamic adjustments in accuracy for energy-quality balance. SAADI can dynamically balance the accuracy, speed, and energy in a division circuit by adjusting iteration counts for reciprocal approximation, diverging from traditional fixed-accuracy designs. It achieves 92.5% to 99.0% accuracy in divisions while providing flexibility in latency scaling, showcasing its potential in low-power signal processing. Wang et al. [389] introduced an approximate divider called "HEADiv" based on the truncated Taylor series, and the induced error is compensated by carefully considering the associated hardware complexity.

3) APPROXIMATE SUBTRACTOR-BASED DIVIDER

Designing approximate dividers can be achieved by employing an approximate subtractor. This method is characterized by the ability to fine-tune error management through adjustable approximation levels in the subtractors, but it may influence the overall efficiency. The subtractor is a common unit in the class of division algorithms called digit recurrence algorithms. For example, Jha and Mekie [390] proposed inexact restoring-array dividers (IRADs) using four different proposed approximate subtractors based on CMOS technology. The authors also in-depth analyze the designs based on PTL and CMOS technologies.

4) APPROXIMATE LOGARITHMIC DIVIDERS

Approximate Logarithmic Dividers operate on the principle of logarithmic computation to perform division, which is a fundamentally different approach from traditional division algorithms. When multiplication and division based on logarithms were first developed by Mitchell in the early 1960s, it marked the beginning of the acceptance of approximation computing [322]. Logarithmic Dividers (LDs) introduced significant errors, which makes them unsuitable for applications where high precision is required. However, LDs are characterized by low complexity, low power consumption, and high speed. This makes them well-suited for use in error-tolerant applications such as digital signal processing, image and video processing, and machine learning algorithms, where they contribute to more energy-efficient designs [391], [392]. Liu et al. [391] addressed the issue of high errors by combing restoring-array and logarithmic dividers to design approximate hybrid dividers. Also, Wu et al. [392] introduced a low-power, high-performance approximate divider using logarithmic operation and piecewise constant approximation. The design was optimized using a heuristic algorithm to minimize errors.

5) APPROXIMATE HIGH-RADIX DIVIDERS

The importance of high-radix dividers is their ability to significantly improve computing speed and efficiency, but they also need careful consideration of hardware complexity, power consumption, and precise control. For example, Chen et al. [393] proposed a high radix divider and analyzed and compared this design with other different approximate dividers. They showed that the approximate radix-2 divider is particularly beneficial in constrained-resource applications, but the high-radix divider is useful for applications requiring high-speed computations. The decision to implement a high-radix divider should be based on a comprehensive analysis of these factors (computational efficiency, precision, latency, scalability, and circuit complexity) in the context of the intended.

D. APPROXIMATE ELEMENTARY AND ACTIVATION FUNCTIONS

The importance of elementary and activation functions in various computational paradigms cannot be overstated. Elementary functions such as trigonometric, exponential, and logarithmic forms are the bedrock of numerous applications in science and engineering. However, the complexity of these functions often poses challenges in real-time or high-performance computing environments. The need for rapid calculations in applications like signal processing and control systems makes it imperative to find efficient ways to implement these functions, often leading to a trade-off between speed and accuracy.

On the other hand, activation functions are the linchpin of deep learning algorithms, particularly in the architecture of neural networks. These specialized functions introduce the necessary non-linearity that enables the network to learn from data and adapt to various complexities. They are crucial in applications that require pattern recognition, such as image and speech recognition, natural language processing, and even in complex game theory problems. However, the choice of an activation function and its implementation can significantly impact the learning efficiency and operational complexity of a neural network. The challenges here are multifold, including but not limited to, the vanishing and exploding gradient problems, computational cost, training convergence and the risk of overfitting. Therefore, understanding the mathematical properties and computational complexities of these functions is crucial for both academic research and practical implementations.

These characteristics collectively ensure that activation functions can effectively support the diverse needs of neural network training. For instance, the Gaussian Error Linear Unit (GELU), the Rectified Linear Unit (RELU), the Leaky RELU, the Sigmoid, and the Hyperbolic Tangent (Tanh) are all examples of popular activation functions. GELU is known for its smoothness and is often used in transformer models like GPT-3, BERT, and most other Transformers [394]. For models like BERT and GPT-2 that employ the tanh approximation, utilizing this method to approximate the GELU activation function is advisable for model reproduction. However, it's worth noting that this approach generally yields less accurate results and can be slower for large input sizes compared to directly computing the accurate GELU function [395]. RELU, characterized by its simplicity and computational efficiency, is widely used in convolutional neural networks but suffers from the "dying RELU" problem where neurons can sometimes become inactive. Leaky RELU addresses this issue by allowing a small, non-zero gradient when the input is less than zero. Sigmoid and Tanh functions are among the earliest used activation functions and are particularly useful in scenarios where the output needs to be scaled between specific ranges; however, they are less popular in deep networks due to the vanishing gradient problem. Each of these activation functions has its own advantages and disadvantages, and the choice often depends on the specific requirements of the neural network architecture and the problem being solved. Figure 18 shows the most used activation functions over the last six years [396].

Typically, five prevalent computing techniques are used to implement these functions, including look-up table (LUT) approach [397], [398], the polynomial approximation methodology [399], [400], Piecewise linear, nonlinear and polynomial approximation, shift-and-add algorithms [401] like the coordinate rotation digital computer (CORDIC) algorithm [402], [403] and Hybrid Approaches [404]. The approximate and stochastic computing approaches [405], [406], [407], [408], [409], [410] have also garnered a considerable interest in recent years.

There are benefits and drawbacks of strategies mentioned above [404]. The LUT method, known for its simplicity and speed in computing elementary functions, demands significant silicon space due to its memory-based accuracy. While it's computationally simple and fits stable functions, its accuracy hinges on memory size [411]. Recent advancements have focused on enhancing LUT methods through techniques like linear interpolation [412], range addressable LUTs (RALUT) [413], table-lookup-and-addition methods like multipartite methods [414], input-aware quantized table lookup [415] and twofold lookup methods [416] though these also introduce challenges in terms of hardware complexity. Polynomial approximation, used for finer estimates, necessitates numerous multipliers, adders, and coefficient-storing LUTs, making it area-inefficient and slow [417]. The CORDIC algorithm is a cost-effective iterative method using adders, shift operations, and registers. However, it's limited by serial multiplier-like delays and a narrow input range, making it slower for exponential and hyperbolic functions. Despite this, enhancements over the past two decades show promise for efficient real-time computing solutions. Function approximation varies in complexity and suitability [418]. Piecewise Linear Approximation (PLA) is basic, offering low computational needs and ease of use, ideal for simple control systems and initial data analysis but struggles with complex functions. Piecewise Nonlinear Approximation (PNA) addresses this by using nonlinear functions for better complexity handling, useful in machine learning and financial models but with higher computational demands. Piecewise Polynomial Approximation (PPA) uses segmented polynomials for function approximation, common in signal processing and scientific computing but can have boundary issues. Hybrid methods mix various techniques to enhance accuracy or efficiency [404]. Recently, there's been significant interest in approximation and stochastic computing, known for high speed, fault tolerance, and low cost. While stochastic computing offers low power usage, it faces challenges like reduced precision and longer latency [419]. Approximate computing, on the other hand, balances hardware cost and accuracy, showing potential for improving integrated system performance [410], [420].

For example, Dong et al. [41] introduced a piecewise linear approximation computation (PLAC) method for nonlinear unary functions, which includes an optimized segmenter and quantizer, enhancing the universal and error-flattened piecewise linear approximation approach. Then Wu et al. [42] developed PLAC without a multiplier, later optimized by Zhang et al. [43] to minimize segment count and reduce the maximum absolute error (MAE). For their circuit designs, all authors focused on the [0,1) interval. However, this approach requires the use of the exponential function's scaling property for processing inputs and outputs. Recently, Dalloo et al. [404] proposed hybrid approach for

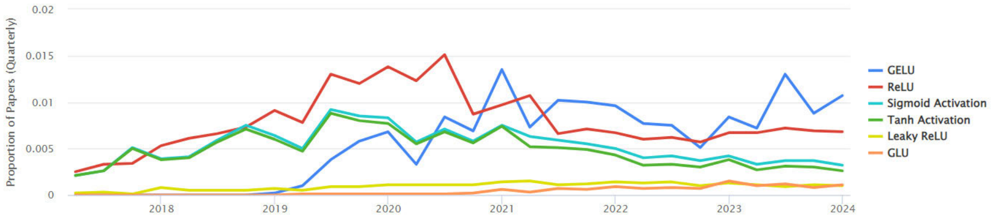


FIGURE 18. The most popular activation functions are used in over the last six years [396].

implementing exponential and hyperbolic functions with input range [10, -10]. Hajduk and Dec [417] proposed a simple FPGA-based method for implementing the hyperbolic tangent function using ordinary or Chebyshev polynomial approximations. The authors examined different implementation configurations to show their effects on FPGA resources and calculation time. For more details, we recommend two references [404] which provides a valuable literature review about methods of implementing these functions. For designing Nth root and power operations, Changela et al. [421] proposed a low-complexity VLSI architecture using three classes of radix-4 CORDIC algorithms. They computed logarithms, division, and exponential operations using the radix-4 of the modified hyperbolic vectoring, linear vectoring, and the modified scaling-free hyperbolic rotation CORDICs, respectively.

In our analysis, we found that although there have been significant improvements in these techniques, challenges persist in attaining balance among energy efficiency, latency, accuracy, and hardware complexity. Specifically, certain existing systems face constraints in terms of scalability, adaptability, and performance, especially when confronting the rigorous demands of real-time digital signal processing (DSP) and artificial intelligence tasks. We believe addressing these challenges demands innovative approaches that can effectively achieve the trade-offs of hardware designs in elementary and activation function computations.

IX. APPROXIMATE LOGIC SYNTHESIS AND FRAMEWORKS

Approximate logic synthesis (ALS) is an automated design approach to approximate digital circuits that can achieve a balance between accuracy and efficiency in terms of power, area, and performance. It automates combinational and sequential circuits and accelerator design to be adapted to various applications and technological and user constraints [422]. In real-world applications, the current challenge that faces ALS is how to adapt to different accuracy needs while managing power and delay variations. To accomplish that, it requires the design of quality-configurable circuits that can adjust to varying accuracy levels in real time. Furthermore, it must not only focus on gate-based netlists or Boolean circuit representations but also on inexact operators.

There are two main approaches to ALS: deterministic and stochastic. Deterministic ALS uses predictable techniques to design approximate digital circuits, making definitive and predictable modifications to enhance efficiency. Stochastic ALS uses probabilistic algorithms to randomly simplify or modify digital circuits, leading to varied outcomes in different iterations [423], [424]. Furthermore, ALS can be categorized into four main categories, each of which has unique methodologies and applications: structural netlist transformation, Boolean rewriting, approximate high-level synthesis (AHLS), and evolutionary synthesis.

A. STRUCTURAL NETLIST TRANSFORMATION

Structural netlist transformation involves the optimization of a given logic circuit by transforming its netlist structure. This can be achieved through various techniques such as gate replacement, reordering, or removing [355], [422], [423], [425]. Several Approximate Logic Synthesis (ALS) methods function by manipulating the circuit netlist. For instance, Gate-Level Pruning (GLP) [355] and Circuit Carving [425]achieve this through the removal of gates from a circuit. Conversely, SASIMI [5] employs a different approach by altering the circuit's wiring. The primary goal is to reduce the overall complexity of the circuit without significantly affecting its functionality. The AxLS framework [426] is an open-source tool dedicated to the exploration and testing of existing netlist transformation techniques, serves as a pivotal resource in the field of Approximate Logic Synthesis (ALS). This ALS converts Verilog netlist to synthesized netlist (in XML format) based on standard-cell library and then applied the approximate techniques under user and application constraints. Finally, AxLS uses external synthesis and simulation tools for analyzing and evaluating the approximate netlist. Figure 19 shows the simplified version of the AxLS framework.

In another work, Witschen [427] introduced an innovative methodology for ALS called MUSCAT, which generates valid approximate circuits by inserting cutpoints into the netlist. It utilizes formal verification engines to identify minimally unsatisfiable subsets, ensuring optimal cutpoint activation without violating quality constraints. MUSCAT outperformed the state-of-the-art methods, including AIG-rewriting [428] and EvoApproxLib [429], achieving up to 80% higher savings in circuit area with lower computation times.

B. BOOLEAN REWRITING

Boolean rewriting focuses on the manipulation of Boolean functions to achieve a more efficient representation. This

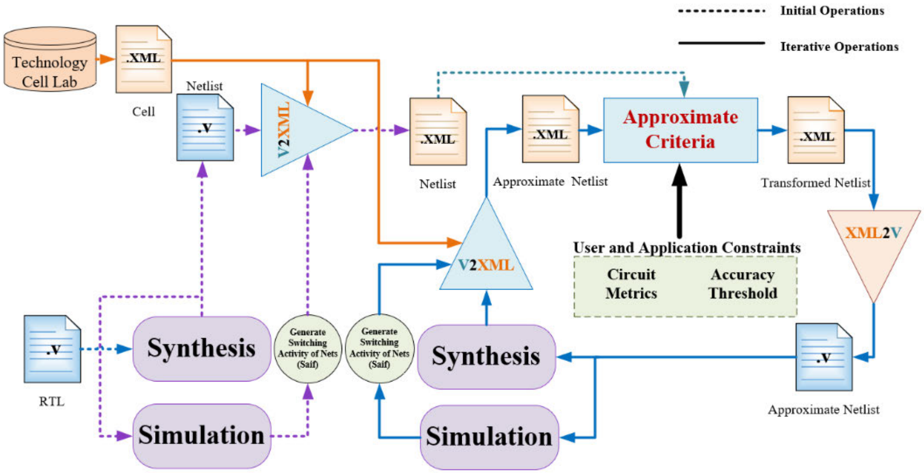


FIGURE 19. The simplified version of the AxLS framework [426].

can involve the use of approximation techniques to simplify complex Boolean expressions. The main objective is to reduce the computational complexity of the function while maintaining an acceptable level of accuracy. For example, Hashemi et al. [430] introduced BLASYS, a novel paradigm that uses Boolean matrix factorization (BMF) to synthesize approximate circuits. This method allows for a balance between accuracy and circuit complexity. This approach saved the power up to 63%, with a cost of 5% of the average relative error.

Recently, Rezaalipour et al. [431] proposed a new algorithm named XPAT, which is designed for creating approximate circuits through Boolean rewriting. The XPAT algorithm uses an SMT solver to customize circuits based on a sum of products template. It outperforms existing methods (MUSCAT and BLASYS) in reducing circuit areas by 9.85% on average, with up to 60.4% improvement in some cases. Figure 20 shows the test results which indicate savings in area for different error thresholds (ET). However, XPAT has longer runtimes for larger benchmarks, potentially addressable by using multi-level templates or applying XPAT iteratively to circuit parts.

Ammes et al. [432] introduced a two-level approximate logic synthesis method using cube insertion and removal, demonstrating scalability for large circuits with high error thresholds. The method achieved literal number reductions ranging from 38% to 93%, depending on the error rate, ranging from 1% to 5%. The authors provided the codes online. While the authors made significant strides with their two-level synthesis method, it's essential to recognize the broader landscape of research in this domain. Diverse methodologies have been introduced, each with its own unique approach and emphasis. Among these, the work of Wu et al. [433] stands out, offering a multilevel perspective on the problem by proposing ALFANS as an advanced multilevel approximate logic synthesis framework, utilizing the Boolean network representation of circuits. Central to ALFANS is its capability for node simplification in Boolean networks. Another approach, Barbareschi et al. [434], introduced an open-source systematic approximate design approach tailored for combinational logic circuits. The authors potentially minimized hardware resource needs by using the non-trivial local rewriting of and-inverter graphs (AIG) to reduce the AIG-node count. Through multi-objective optimization, the approach judiciously balances approximation with optimal error and hardware trade-offs and includes the synthesis of Pareto-optimal configurations to ascertain tangible benefits. Meng et al. [435] introduced ALSRAC, an open-source simulation-based Approximate Logic Synthesis (ALS) flow, employing approximate re-substitution with an approximate care set. Utilizing logic simulation, the authors recommend approximating the care set in ALSRAC by identifying external don't-cares (EXDCs) through the maximum error distance constraint. They translated the proposed care patterns to internal nodes rather than primary inputs (PIs) to enhance scalability. Also, they noticed that in the larger circuits, the number of PIs increases exponentially with increasing EXDCs. Experimental outcomes indicate that the proposed approach results in an area reduction of 7%−18% in comparison to existing state-of-the-art methods.

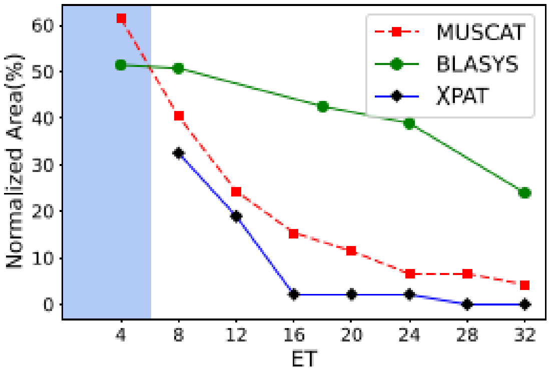


FIGURE 20. Comparison of areas of Multiply-add obtained by XPAT, MUSCAT, and BLASYS [431].

C. APPROXIMATE HIGH-LEVEL SYNTHESIS (AHLS)

In contrast to traditional approximate logic synthesis (ALS) techniques, which focus on gate-based netlists or Boolean circuit representations, Approximate High-Level Synthesis (AxHLS) aims to utilize inexact operators. AxHLS is a strategy that aims to efficiently implement designs in high-level languages such as behavioral Verilog or C language. It focuses on the design and synthesis of hardware at a high abstraction level. The strategy involves transforming high-level descriptions, like C/C++ code, into hardware descriptions, like VHDL or Verilog. The main goal is to create hardware that meets specific performance, power, and area constraints while often sacrificing accuracy.

One of the early works in this field, Nepal et al. [436], introduced an advanced ABACUS methodology for the autonomous generation of approximate designs from behavioral RTL descriptions, expanding potential approximation avenues. The Automated Behavioral Approximate CircUit Synthesis (ABACUS) methodology is an approximate logic synthesis tool that transforms RTL descriptions into ASTs through applying various operators, such as data type simplifications, arithmetic operation approximations, and loop modifications. It utilizes a design space exploration technique for identifying optimal designs on the Pareto frontier, considering accuracy and power balance. ABACUS focuses on optimizing critical paths post-synthesis for additional power savings through voltage scaling. This tool, featuring a recursive stochastic evolutionary algorithm, generates optimal approximate hardware variants from high-level Verilog inputs, with the codes accessible online.

Recently, Leipnitz and colleagues [437] developed an AHLS design framework for FPGAs capable of autonomously determining the most effective combinations of multiple approximation techniques. This approach could be suitable for specific applications and constraint design. The proposed method outperformed single-technique approaches in various benchmarks, reducing mean squared error by up to 30% and increasing accuracy by up to 6.5%. Additionally, Castro-Godínez et al. [438] developed a new approximate high-level synthesis framework for approximate accelerators based on a library of approximate functional units. Furthermore, this framework addresses the challenge of optimizing resources while meeting accuracy constraints. It features "AxME," which represents analytical models for resource estimation, and "DSEwam," which represents a methodological approach for the exploration of design space in applications that exhibit a tolerance for errors. These tools enable the automatic generation of optimal approximate accelerators from C language descriptions. The framework is released as open-source to advance research in approximate accelerator generation.

D. EVOLUTIONARY SYNTHESIS

Evolutionary synthesis employs evolutionary algorithms, such as genetic algorithms, to optimize digital circuits. It involves iterative processes of selection, crossover, and mutation to explore the design space and find optimal or near-optimal solutions. Evolutionary algorithms are heuristic and metaheuristic search algorithms such as the genetic algorithm (GA), genetic programming (linear and cartesian genetic programming), machine learning, deep learning, etc. For example, Ranjan et al. [439] proposed a novel approach that leverages state-of-the-art AI generative networks to synthesize constraint-aware arithmetic operator designs to be optimized specifically for FPGA.

Despite the focus of the most existing approximate logic synthesis methods primarily on ASIC designs, there are not many works of ALS for FPGA design. For example, Wu et al. [440] introduced a novel method specifically tailored for FPGA design. They used the adaptability of lookup tables and developed a technique that combines wire removal and local function alteration.

One of the evolutionary synthesizers was the development of a reinforcement learning-based logic synthesis framework known as AISYN by Pasandi et al. [441]. This study advocated for the incorporation of Artificial Intelligence (AI), particularly Reinforcement Learning (RL), into logic synthesis procedures. The hypothesis is that AI and RL can aid in increasing Quality of Results (QoR) by avoiding local minima, thereby transforming logic synthesis optimization into an AI-guided process. Experimental evaluations show AI-guided logic synthesis can significantly improve key metrics like area, delay, and power. A RL-aided rewriting algorithm improved total cell area by 69.3%, highlighting the transformative potential of AI and RL in enhancing logic synthesis efficiency. Furthermore, Pasandi et al. [442] developed Deep-PowerX, a framework combining deep learning, approximate computing, and low-power design for logic optimization at the synthesis level. It significantly reduces the dynamic power consumption and area of digital CMOS circuits with acceptable error rates. Compared to exact solutions, it achieves up to 1.47× and 1.43× reductions in power and area, respectively, and surpasses current approximate logic synthesis tools by 22% and 27%, with much lower run-times.

Within the field of Genetic Programming (GP), the absence of a Boolean function benchmark suite for logic synthesis (LS) has been recognized as a significant issue [443], [444]. Roman et al. [444] developed a benchmark suite for logic synthesis, encompassing various Boolean functions used in evaluating genetic programming systems. They presented baseline results from previous studies and their own experiments using Cartesian genetic programming (CGP). To automate the functional approximation of combinational circuits at many levels, including gate and register-transfer levels, Sekanina et al. [445] proposed a genetic programming-based approach structure.

E. ERROR ESTIMATION AND EVALUATION FRAMEWORKS

Error Estimation Frameworks offer structured methodologies for evaluating potential inaccuracies in computational systems. Tailored for contexts employing approximations, these frameworks leverage sophisticated algorithms to balance accuracy and efficiency trade-offs, thereby serving as benchmarks for assessing computational result reliability. In the ALS process, as depicted in Figure 21, integration with error modeling and Quality of Results (QoR) evaluation is fundamental [422]. The process commonly starts with an error-modeling phase, which is designed to evaluate the effects of removing individual gates on the circuit's accuracy and give annotating them with the error percentage. This step can guide ALS methods in identifying the least error-prone transformation. Then, it is followed by a post-synthesis error estimation phase, or QoR evaluation. This evaluation is significant for ensuring that the resultant circuit complies with specified demands [422]. Monte Carlo sampling methods are commonly used in ALS approaches to determine the actual error introduced by approximations in the process, making it a prevalent technique in error evaluation in approximate computing. This method is notably utilized in Approximate Logic Synthesis (ALS) methods, including BLASYS [430] and Vasicek [446] for Quality of Result (QoR) evaluation, as well as in Su's approach [447] for error modeling. However, a significant constraint inherent to Monte Carlo sampling is the absence of definitive guarantees, as its worst-case error only reflects the highest error within a limited sample space, thus providing a limited exploration scope. Error Estimation Frameworks, for example, VECBEE [448], is a key framework in Approximate Logic Synthesis (ALS), combining Monte Carlo simulation with signal propagation for error estimation. It's adaptable to various error metrics and circuit representations, balancing accuracy with efficiency. This approach was integrated into the open-source ALS methods, which significantly contribute to optimizing circuit approximations.

Recently, Rezaalipour et al. [449] proposed a novel SMT and SAT solver-based algorithm for error evaluation in approximate computing, adaptable to any circuit and error metric. This approach significantly outperforms traditional methods, including AIG-rewriting [428] and [450], by efficiently and systematically navigating the error space, ensuring more accurate and reliable design validations.

In sum, Approximate Logic Synthesis (ALS) techniques, essential in digital circuit design, are categorized into four main types: structural netlist transformation, Boolean rewriting, approximate high-level synthesis (AHLS), and evolutionary synthesis. These categories utilize unique methods for introducing approximations in circuits to enhance efficiency and performance. Key insights into these techniques are also provided by review papers such as [451] and [452]. These works highlight ALS's significance in optimizing digital circuits, especially in applications that balance computational accuracy with efficiency.

X. EMERGING COMPUTING FRAMEWORKS

A. CROSS-LAYER AND END-TO-END AXC FRAMEWORKS

The cross-layer approximation approach, aimed at leveraging the error resilience of applications across various abstraction layers, is becoming popular. This approach involves several approximation techniques from circuit to application level. FPGA approximate computing frameworks harness the flexibility of FPGAs to optimize computational tasks by trading off accuracy for improved performance, energy efficiency, or reduced resource usage [453]. They support adaptable precision for diverse tasks, making them efficient for applications requiring high computational power.

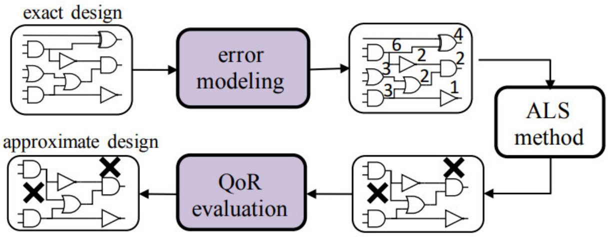


FIGURE 21. Overview of the standard ALS process with the error modeling and quality of results (QoR) evaluation [422].

Efforts to enhance program efficiency and error tolerance have led to SIMD use in FPGAs, facing issues like limited approximation operations, isolated kernel adjustments without thorough evaluation, and a lack of targeted optimization strategies. Furthermore, a comprehensive multi-level approach is essential across all layers from application to circuit [24]. To address these challenges, Ebrahimi et al. [24] proposed a cross-layer methodology for multi-kernel applications using toolchains across layers of abstraction. For designing hardware with runtime-adjustable accuracy, Alan et al. [23] introduced a unique cross-layer approach that reduces energy use with sightly increasing area. Also, Hanif et al. [25] tried to facilitate DNN implementing on resource-constrained devices through introducing a cross-layer approach using various optimization techniques across the computing stack's layers.

Several research studies have shown that applying the principles and techniques of approximate computing, initially designed for the ASIC platform, yields different advantages when implemented on FPGA platforms [454]. Another challenge is that it is time-consuming to explore numerous approximate accelerator variants due to the vast architecture space required for simple applications like Gaussian filters. Therefore, Ullah et al. [371] developed a method to systematically create various effective approximate multiplier designs for FPGA platforms. Subsequently, the authors deployed a range of machine learning models to assess and choose configurations that meet the specific accuracy and performance requirements of the application. Furthermore, Prabakaran et al. [455] introduced a novel end-to-end automated framework named Xel-FPGAs, aimed at enhancing the efficiency of exploring FPGA-based approximate accelerators integrated with advanced statistical and machine learning methodologies. This approach is designed to significantly cut down on the traditionally lengthy exploration time, making the process more efficient and effective. The Xel-FPGAs framework reduced FPGA-based accelerator exploration time by 95%, boosting efficiency with minimal performance impact. It's open-source and available online.

Developed primarily by Xilinx Lab researchers [456], FINN is an open-source tool designed for building fast and flexible deep learning inference on FPGAs. Unlike general DNN accelerators, FINN provides an end-to-end flow that emphasizes co-design and exploration to optimize quantization and parallelization for specific resource and performance requirements. On a ZC706 FPGA platform consuming less than 25 W, FINN delivers record-breaking image classification speeds up to 12.3 million classifications per second at 0.31 microseconds latency with 95.8% accuracy on MNIST, and 21,906 classifications per second at 283 microseconds latency with over 80% accuracy on CIFAR-10 and SVHN datasets.

B. SHANNON-INSPIRED STATISTICAL COMPUTING

In 1948, Shannon established information as a statistical quantity and introduced a theory of communication over noisy channels. He defined channel capacity based on noise statistics and demonstrated that reliable communication is possible if the transmission rate is below this capacity. Shannon also showed that error control codes can approach this channel capacity. Shanbhag et al. [457] inspired to Shannon theory to design and develop principles and fundamental limits for realizing statistical information processing systems using stochastic components.

The Shannon-inspired Statistical Computing framework [457] leverages the statistical properties of both application data and nanoscale hardware to create robust, energy-efficient, and scalable computing systems. By integrating computation within memory (DIMA) and sensor arrays (DISA), and employing statistical design techniques like Data-driven Hardware Resilience (DDHR), Statistical Error Compensation (SEC) and Hyperdimensional Computing (HD), it ensures high reliability even in the presence of significant hardware noise and errors. This framework is particularly advantageous for data-centric applications, offering enhanced performance and energy savings by minimizing data transfer and adapting dynamically to errors. Furthermore, this framework allows circuits to operate at lower SNR levels, significantly saving energy. However, it comes with complexities in design and implementation, requiring sophisticated error-aware models and initial training overheads. This framework is used in advanced machine learning accelerators, low-power medical devices, and large-scale sensor networks, where traditional deterministic computing falls short due to increasing stochasticity at the nanoscale level.

Another example, Kim et al. [458] introduced a maximum likelihood (ML)-based statistical error compensation (MLEC) technique to enhance the compute signal-to-noise ratio (SNR) in 6T SRAM-based analog in-memory computing (IMC) architectures. These architectures, known for their energy efficiency and compute densities in machine learning, are limited by device variations and noise. The proposed MLEC technique improves the accuracy of binary dot-products (DPs) in these systems.

By integrating Shannon-inspired Statistical Computing with Approximate Computing, we can significantly enhance the robustness, adaptability, and energy efficiency of approximate computing systems. Leveraging information-theoretic principles allows for precise error management, optimal design, and dynamic adaptation, making it possible to exploit the benefits of approximation without sacrificing reliability or performance. This hybrid approach provides a powerful framework for developing advanced computing systems capable of meeting the demands of modern data-centric applications and nanoscale technologies.

C. BRAIN-INSPIRED COMPUTING

Astounding progress in several tasks has been driven primarily by advances in deep learning, which form the backbone of today's Artificial Intelligence (AI) developments. The rapid development of artificial intelligence (AI) demands the rapid development of domain-specific hardware specifically designed for AI applications. Neuro-inspired computing (Neuromorphic computing) chips integrate a range of features inspired by neurobiological systems and could provide an energy-efficient approach to AI computing workloads. Neuromorphic computing refers specifically to the design of hardware systems that emulate the neural architecture of the brain. It aims to create physical circuits and devices that operate similarly to biological neurons and synapses. Neuromorphic computing, inspired by neuroscience, is key to next-generation AI. It focuses on three levels: computing models, architecture, and learning algorithms [459]. Spiking Neural Networks (SNNs), with more realistic neuronal dynamics than Artificial Neural Networks (ANNs), serve as the computing model. Architecturally, SNNs enable efficient in-memory computing. Neuro-inspired learning paradigms, including online, learning-to-learn, and unsupervised learning, allow continuous adaptation and form the basis for low-power, accurate, and reliable neuromorphic systems. This includes designing and fabricating neuromorphic chips that replicate the brain's parallel processing capabilities. Neuromorphic systems often use spiking neural networks (SNNs), where information is processed in discrete spikes, similar to neural spikes in the brain. This leads to event-driven, as opposed to clock-driven, computation, which can be more energy-efficient [460]. Neuromorphic computing can be implemented by combining analog and digital circuits to replicate the analog nature of biological processes. This can involve using memristors, specialized transistors, and other nanoscale components to build artificial neurons and synapses [461].

To further enhancement, Sen et al. [462] introduced AxSNN, an approach applying approximate computing to enhance the efficiency of Spiking Neural Networks (SNNs) by selectively skipping low-impact neuron updates, utilizing static and dynamic parameters to identify approximable neurons. They implemented AxSNN in both hardware (SNNAP, synthesized in 45nm technology) and software (on a 2.7 GHz Intel Xeon server) to achieve 1.4–5.5x reduction in scalar operations across six image recognition benchmarks, demonstrating significant improvements in computational efficiency and energy savings with minimal quality loss.

XI. APPLICATIONS

A. APPROXIMATE INTERNET OF THINGS (IOT)

The widespread use of smart devices and sensors has led to a demand for intelligent computing to handle vast amounts of data. Conventional exact computing is commonly used in these devices which suffer of high-power consumption and low performance [463]. To address these challenges, approximate IoT (AxIoT) is the optimal solution and an emerging paradigm in this field, which depends on approximate computing techniques. These techniques manage the intensive computational processing and analysis demands of IoT devices. The exact results are not always required, this feature allows us to accept some errors. Approximate computing offers significant benefits in the resource-constrained IoT devices. Additionally, the big challenges that are faced by a designer are maintaining the quality of computations to meet specific application needs, reliability, and security concerns. To address these challenges, various strategies have been proposed, for example, energy of IOT can be saved using Bloom filter [93], [94], 6T SRAM [266], voltage-frequency-power management techniques [307], [312], approximate IoT processor [317], In-memory computing IMC-based Binary Neural Network (BNN) accelerator [294], DRAM Refresh rate [12]. For example, Ghosh et al. [464] illustrated synergistic approximation by utilizing a smart camera system that performs DNN-based image classification and object detection, highlighting how the sensor, memory, compute, and communication subsystems can all be effectively approximated. Adaptive approximation levels, which allow for dynamic adjustment based on application needs, can manage the accuracy-efficiency trade-off. Hierarchical sampling algorithms, such as stratified reservoir sampling, offer rigorous error bounds while enhancing computation efficiency, as demonstrated in the APPROXIOT system [51]. APPROXIOT was implemented based on Apache Kafka. Fabjančič et al. [465] introduced "Mobiprox," a framework for on-device deep learning with adjustable accuracy. It features tunable tensor operation approximations and runtime layer adjustment, utilizing a profiler and tuner for optimal configuration. The results show Mobiprox's implementation on Android OS reduces energy use by up to 15% in mobile applications like activity recognition and keyword detection, with minimal accuracy loss. Mobile devices, reliant on battery power, face constraints that make battery life a crucial factor. Reducing computational energy could significantly benefit these systems. Therefore, exploring mobile approximate computing emerges as a promising research and development avenue for advancing approximate computing paradigms [464], [466]. Qaim et al. [467] surveys energy-efficient solutions including data compression and approximate computing techniques for IoWT applications from 2010 to 2020. It categorizes these solutions, highlighting their pros, cons, and key performance parameters. The study discusses trade-offs and suggests future research directions to improve wearable device performance and address challenges. Recently,

B. DEEP AND MACHINE LEARNING

Strategic approximate computing reduces precision in neural network computations on a need basis that saves power and time, particularly for multiply-and-accumulate (MAC) operations, which dominate the energy use in DNNs [468]. Studies indicate that these operations are responsible for consuming up to 99% of the energy in Deep Neural Networks (DNNs) [469]. Concentrating on simplifying multipliers within MAC units results in huge energy savings with hardly any degradation in accuracy. These approximate designs can be customized to suit different accuracy specifications as they are error-configurable. Efficiency is further optimized through dynamic reconfigurability and temperature-aware methods that control chip temperatures while ensuring computational speed, energy savings, and output quality remain balanced. Sarwar et al. [470] proposed an approximate multiplier and a Multiplier-less Artificial Neuron (MAN) to improve neural networks' energy efficiency by exploiting error tolerance and computation sharing. They also recommend retraining to offset accuracy losses. Evaluations show MANs significantly reduce power and size with minimal accuracy impact, maintaining consistent speed. Peng et al. [471] proposed "AXNet," a unified neural network that simplifies training and improves efficiency by integrating approximation and prediction tasks. The results show a 50.7% increase in safe approximation rates and significant reductions in training time. The codes are available online. Ashar et al. [472] proposed a novel quantize-enabled multiply-accumulate (MAC) unit with a right shift-and-add computation for runtime truncation without extra hardware. Applying this MAC to a LeNet DNN model reduced resources by 42% and delay by 27%, ideal for high-throughput edge-AI applications.

Balancing CNN accuracy, efficiency, and resource use presents challenges, exacerbated by high storage and computational needs and inefficient hardware deployment. Despite this, CNNs are crucial in computer vision, though at the expense of greater computational demand, as seen in models like VGG-16. Addressing these issues requires software-hardware co-optimization, including model compression techniques like pruning and parameter quantization, to enhance CNN efficiency on FPGA platforms [473], [474]. For example, Sui and his colleagues proposed new CNN pruning [474] and Quantization [473] methods aimed at reducing storage requirements, computational load, and enhancing hardware deployment efficiency. The study [474] presented a new CNN pruning method (KRP) which combined with GSNQ quantization [473] to achieve a 27× reduction in model size and improving FPGA efficiency. There are many techniques to enhance deep CNN and machine learning algorithms including fixed-point arithmetic, zero-skipping and weight pruning for enhancing CNN on FPGA [475], compression through innovative use of parallel layer processing and pipelining [476], Compression though using reversed-pruning, peak-pruning and quantization [477], cross-layer approach using the structure pruning and inputs and network parameters quantization at the software and approximate arithmetic units at hardware level [25], quantization in PIM [478], approximate accelerators [455], approximate adder [420], approximate logarithmic multiplier [29], [374], Computation skipping [124], ApproxTuner [233] frameworks [236], [240], Approximate Memory based on Voltage Scaling [258], analog processor-in-memory [291], hybrid PIM accelerator [292], DCT, Quantization, Sparse matrix compression [273], and other techniques mentioned in Table 3.

Approximate processors and accelerators, which embody the synergy of hardware and software co-design in approximate computing, are engineered to boost computational efficiency through permissible inaccuracies, making them well-suited for applications where error tolerance is acceptable. There are many proposed approximate processors depend on approximate computing techniques to enhance the overall system efficiency, including relaxed precision in TPUs for implementing NN applications (MLPs, CNNs, and LSTMs) in datacenters [17], mixed precision for GPU Tensor Cores for Deep Neural Networks (DNNs) [479], processing elements (APEs) consisting of a low-precision multiplier and an approximate adder in TPU [319], parallel analog convolution-in-pixel, and low-precision quinary weight neural networks [480]. Gharavi et al. [481] proposed enhancing multicore performance with configurable approximate Arithmetic units. Their machine learning framework dynamically adjusted frequency and precision to optimize performance within TDP constraints. Experiments showed a 19% speed increase using a floating point approximate ALU with three configurations per core, all within the same TDP limit.

Machine learning enhances IoT by analyzing vast data for actionable insights, crucial for applications like wearables and smart devices [463]. The embedded processing near the sensor is often preferred to cloud processing due to privacy, latency, and bandwidth constraints. Despite these advantages, sensor devices face significant challenges related to energy consumption, cost, throughput, and accuracy. Circuit designers are key to developing energy-efficient solutions for these tasks. For example, Younes and his colleagues [482] published a couple of research papers which applied algorithmic level approximate computing techniques (AxCTs) to supervised machine learning algorithms, specifically K-Nearest Neighbor (KNN) and Support Vector Machine (SVM), for applications in touch modality and image classification. These techniques, including reduced sampling and precision as well as loop perforation, aim to decrease complexity and latency while maintaining a balance between speed and accuracy. Comparing to traditional exact implementations, the results in showed the proposed method achieved a 49% reduction in power consumption and a 3.2× speedup. Furthermore, it used 40% fewer hardware resources and consumed 82% less energy for classifying touch inputs, all with a minimal accuracy loss of less than 5%. In addition, Mienye and Jere [483] addressed a gap in the literature by providing a comprehensive overview of decision tree-based methods in machine learning. It explored core concepts, algorithms, and applications, from early development to recent high-performing ensemble algorithms. Also, they discussed the methods. tree pruning to enhance the performance of model and reduce the overfitting.

C. DATA MINING

Redundant computations and data are considered as big challenges for algorithms in terms of speed, scalability, memory, and efficiency, for example, unnecessary computations, function calls, Redundant iterations, redundant memory access, etc. These inefficiencies increase execution time, waste computational resources, and reduce the scalability of data mining analyses [484]. The core challenge of data miming algorithms is to extract extracting hidden knowledge from large datasets and mitigating use of redundant computations. Sampling is a data reduction strategy, addresses these volume-related challenges in environments running big data tasks like classification and clustering. There are several papers discussed the concept of approximate data mining. For example, stratified random sampling were used from streaming and stored data [36], [37]. Graph sampling [38] is a very effective method to deal with scalability issues when analyzing large-scale graphs. There are others data mining techniques such as memoization (storing and reusing previous computations), efficient data structure design, and careful algorithm optimization, loop perforation, iteration skipping, memory access skipping, Computation skipping, Function approximation, etc., we discussed these techniques in previous sections.

The machine learning algorithms are considered common data mining algorithms. For example, approximate nearest neighbor search (ANNS) is considered as core solution in data-mining and is widely used in different applications such as computer vision, information retrieval, etc. [485]. Approximate nearest neighbor search algorithms are used for fast retrieval of relevant information. Instead of perfect matches, these algorithms find items that are "close enough" in high-dimensional data spaces and saving computational expense during large-scale searches [486]. For instance, numerous major corporations, including google, employ this strategy [487].

D. SECURITY

We know that approximate computing promises significant advantages but there are security implications, particularly in sensitive applications, require careful consideration and further research. Approximate computing can complicate reverse engineering efforts but could introduce new target areas for hardware Trojans, particularly in circuits controlling the level of approximation. Approximate circuits' defense against passive side-channel attacks can differ with voltage-frequency settings, making security assessments challenging. Approximate circuits, particularly at operational limits, may be vulnerable to fault injection attacks, but the full effects and countermeasure effectiveness are still unclear [488], [489]. Also, Processing-In-Memory alters security models due to factors like architecture changes, different programming models, side-channel risks, device reliability, and potential hardware Trojans. To address these challenges, Yellu and Yu [490] proposed obfuscating the boundary between approximate and precise computations by blurring the entry point and broadening the transition zone. The entry-blurring scheme uses a hidden quality metric correlated with approximation errors to conceal the switch between modes, enhancing resilience to attacks. The boundary-broadening scheme extends the transition zone with dual thresholds and random AC module selection, further securing AC systems. Their methods significantly improve application quality (up to 168% over baseline) with minimal impact on latency, area, and power costs (increases limited to 6% and 8%, respectively). Another work, Islam [491] highlighted the security risks in AC synthesis for implementing approximate Computing (AC). He showed how vulnerabilities could be exploited to insert malicious elements like Hardware Trojans without affecting efficiency. Therefore, he proposed a defense mechanism using input vectors and path profiling to detect such threats and emphasized the necessity of incorporating security into AC systems to prevent exploitation and suggesting future enhancements to synthesis tools for improved security. The study [485] introduced a cloud-assisted LSH scheme for efficient Approximate Nearest Neighbor searches. He tackled the high computational demands of traditional LSH, especially on devices with limited resources. This approach ensures data privacy and includes a method to verify the integrity of cloud-processed results. Experiments and analyses confirmed the scheme's effectiveness, security, and practical applicability, offering a viable solution for resource-constrained environments. The research [492] introduced a multilevel approximate architecture for Ring-Learning-with-Errors (R-LWE), a quantum-resistant cryptographic scheme ideal for IoT due to low area and memory requirements. The proposed novel AxRLWE approach is tailored for resource-constrained IoT devices, achieves substantial reductions in area and energy on FPGAs and ASICs, with some compromise on quantum security.

We conclude that while approximate computing offers significant advantages in certain applications, its security implications, particularly in sensitive applications, require careful consideration and further research.

XII. TOOLS AND LIBRARIES OF APPROXIMATE CIRCUIT

Approximate computing is an emerging paradigm that allows trading off design accuracy and improvements in design metrics such as design area and power consumption. This paradigm is widely used in applications across various abstraction layers through managing and controlling the error. Numerous researchers share code or plan to release libraries of approximate components for application use, or offer benchmark suites of diverse applications as open-source to assist others in their research. Additionally, another group of researchers makes available free software tools to support scientific research. In Table 4, we present a selection of libraries for approximate components and established benchmark suites. At end of this table, we include two websites: one offering access to published papers with accompanying codes, particularly in the DL/ML fields, and the other hosting a comprehensive collection of open-source benchmark suites, along with tested and recommended platforms for their use. In this review paper, we have noted at the conclusion of each discussed work that the authors have made the code available.

There are several free software programs provided the teams of researchers that can be used for circuit or processor design at the transistor and logic levels, for example BLASYS tool-chain framework which is used to design approximate circuits through a couple of free tools. Recent advancements have facilitated rapid DNN deployment on FPGAs through automation tools. FP-DNN [504] streamlines converting TensorFlow DNN models into efficient FPGA implementations, supporting networks like CNNs with improved performance and flexibility. ARTLCNN compiler [505] streamlines FPGA hardware customization for CNN inference, significantly boosting performance by using an optimized RTL module library and a flexible system template. Tested on Intel FPGAs with complex CNNs, it achieves over double the efficiency of current automated solutions. Another open-source tool called FINN which primarily developed by Xilinx Lab researchers [456] for building fast and flexible deep learning inference on FPGAs. FINN provides an end-to-end flow and focuses on co-design and exploration to optimize quantization and parallelization tuning for specific resource and performance needs. It's not a general DNN accelerator [506].

The choice of the tool is determined by many factors, including the amount of simulation and synthesis needed, the complexity of the circuit, and the designer's expertise with the program.

XIII. CHALLENGES AND FUTURE DIRECTIONS

A. CHALLENGES IN PROCESSING-IN-MEMORY (PIM) IMPLEMENTATION

Research and development in the field of Processing-In-Memory are ongoing, and it has the potential to play a crucial role in addressing the performance bottlenecks faced by traditional computing architectures in dealing with massive amounts of data in modern computing scenarios. In previous

TABLE 4. Open-source libraries and benchmark suits of approximate computing techniques and applications.

Ref./Year	Name	AxC Units/ Benchmark Suite	Platforms	Source code	Implementing Tools /libraries used
[493]/2004	SciMark 2.0	Fast Fourier Transform, Jacobi Successive Over-relaxation, Monte Carlo, Sparse Matrix Multiply, and dense LU matrix factorization	Intel and AMD processors[1]	C++	N/A
[494]/2009	Rodinia_Bench	Applications from different domains (e.g. Image/Video Compression, Data Mining, etc.)	GPU and CPU Heterogeneous computing	OpenMP, OpenCL and CUDA	N/A
[324]/2014	ApproxAdderLib	Adders (GeAr, ACA-I, ETAII, ACA-II and GDA)	FPGA and ASIC	MATLAB, VHDL/Verilog	MATLAB R2013a and ISE Design Suite 14.5
[495]/2017	Axbench	Applications from different domains (e.g. Computer Vision, Data Analytics, Multimedia, Web Search, Finance, etc.).	GPU, CPU, FPGA, ASIC	C++, Verilog, CUDA	Boost Libraries, G++, Python, Fast Artificial Neural Network Library, NPU compiler, CUDA Toolkit, Rodinia
[429]/2017	EvoApprox8b	8-bit approximate adders and 8-bit approximate multipliers	FPGA and ASIC	Verilog, Matlab and C	multi-objective Cartesian genetic programming
[496]/2018	SMApproxLib	Multipliers	FPGA, CPU	VHDL, Matlab	Matlab, Vivado 17.1
[497]/2019	PaderBench	Adder, FFT, Cordic, array multiplier, filters, MAC….		Verilog	N/A
[422]/2020	BACS	FFT, SVM classifier, Adder, Multiplier, filter, square	FPGA and ASIC	Verilog and python	open-source tool ABACUS together with the FreePDK 45-nm library.
[498]/2020	DSPBench	Distributed Data Stream Processing Systems: big data, data-stream, apache-spark, etc. and different domains like Finance, Telecommunications, Sensor Networks, Social Networks and others	Azure Cloud computing service	Java	N/A
[499]/2021	RTRBench	Benchmark suite for real-time robotics: 6 kernels for example, Extended Kalman Filter, Reinforcement learning using Bayesian Optimization	CPU	C++	N/A
[500]/2022	SOMALib	Library of Exact and Approximate Activation Functions for Hardware-efficient Neural Network Accelerators	CPU, GPU, and FPGA, ASIC	RTL (VHDL, Verilog)	N/A
[501]/2023	TransPimLib	Transcendental Functions on Processing-in-Memory Systems: CORDIC-based and LUT-based methods for trigonometric functions, hyperbolic functions, exponentiation, logarithm, square root, etc.	PIM	C	N/A
[502]	Open-benchmarking	This website developed by Phoronix Media. They collected free and open-source benchmark suits, analysis and tests on different platforms.			
[503]	paperswithcode	This website developed by Meta AI Research team, which collect a free and open resource with Machine Learning papers, code, datasets, methods and evaluation tables. It also archives the analysis of the activity of works			

section, we primarily discussed approximate memory techniques, focusing on strategies to overcome the "Memory Wall" through various circuit and architectural advancements, including Approximate Computing. It details methods such as voltage scaling, lowering refresh rates, and data compression or encoding to enhance energy efficiency in memory systems, particularly for applications like machine learning that can tolerate some level of errors. These strategies aim to balance power conservation with acceptable error rates, contributing to the broader field of energy-efficient and performance-optimized memory design.

As new memory technologies continue to evolve, future directions for PIM may involve the integration of processing logic with emerging memory technologies like resistive RAM (ReRAM) or phase-change memory (PCM). The concept of Processing-In-Memory (PIM) holds immense potential for revolutionizing computing architectures by bringing processing capabilities closer to data storage, thereby reducing data movement overhead and improving system efficiency. PIM is a nascent technology with ongoing advancements in materials, devices, and circuit design. However, the realization of PIM faces several challenges must be addressed to realize the full potential of PIM.

The integration of processing logic into memory cells or controllers faces challenges of increasing hardware complexity and design. This necessitates careful design and using efficient power management techniques (e.g. DVFS, DPM, etc.) to handle unacceptable increased power consumption and heat dissipation that come with added processing capabilities in memory. As processing tasks move closer to the memory, ensuring data reliability and integrity during in-memory computation poses a critical challenge. Safeguarding data against errors and corruption during processing necessitates developing robust error correction techniques to ensure maintaining the integrity of data during the processing cycle. Furthermore, scalability issues arise as PIM is extended to larger memory sizes and higher bandwidths. These include maintaining performance efficiency and managing the complexities of larger PIM systems. To address scalability issues in PIM, we need to trend to use approximate computing and management strategies, for example by reducing precision or using adaptive precision scaling or lossy compression, using approximate memory access including partition data on critical important and refresh rate.

For successful PIM implementation, developing appropriate programming models and software support is crucial. This involves creating new programming paradigms and tools that can efficiently leverage the PIM architecture to minimize data movement and maximize computational efficiency. As future directions, PIM shows promise in advancing artificial intelligence and machine learning workloads, big data analytics, and high-performance computing. The exploration of memory-centric architectures and emerging memory technologies, alongside considerations for security and privacy, can further enhance PIM's capabilities [507]. As new memory technologies like ReRAM and PCM evolve, integrating these with processing capabilities poses challenges in terms of compatibility, performance optimization, and leveraging their unique properties for PIM. Also, the ensuring the security and privacy of data processed within memory becomes an important consideration. This includes addressing potential vulnerabilities and safeguarding against unauthorized access or tampering. The limited precision of analog PIM accelerators, particularly during the high-precision backward propagation phase in CNN training, presents challenges that necessitate innovative solutions like hybrid PIM accelerators and Shannon-inspired statistical computing principles.

Instead of just augmenting existing CPUs with PIM capabilities, future directions might involve the exploration of memory-centric architectures where the memory is at the center of computation, and traditional CPUs are reimagined as accelerators. Additionally, the integration of approximate computing techniques with PIM holds promise for optimizing computation in memory-intensive tasks and further improving energy efficiency while providing satisfactory output quality for specific applications. For example, recently, Jinyu et al. [478] introduced CIMQ, a quantization framework for improving neural network accelerator efficiency using Computing in Memory (CIM) architectures.

Embracing these challenges and future directions can pave the way for the widespread adoption of Processing-In-Memory and revolutionize modern computing paradigms. In conclusion, while challenges exist, ongoing research and development efforts in the field will unlock the full potential of Processing-In-Memory for next-generation computing systems.

B. ADDRESSING DESIGN AND VERIFICATION COMPLEXITIES IN APPROXIMATE COMPUTING CIRCUITS

In the pursuit of energy-efficient computing, a pivotal challenge emerges in meeting the real-time precision demands of various applications. Conventional circuits typically function at constant power levels without adjusting for the specific precision needs of individual tasks. This one-size-fits-all approach to power consumption, irrespective of the required accuracy for distinct operations, represents a significant hurdle in optimizing energy efficiency across varied computing applications. The optimal approach for energy-efficient computing involves designing circuits that are both approximate and reconfigurable, ensuring that power consumption is closely aligned with the required computational accuracy. Reconfigurable circuits adapt their configuration to the current computational needs, optimizing energy efficiency by alternating between high-precision and lower-precision modes as necessary. This combination offers a tailored balance between energy conservation and computational accuracy for various applications.

Designing approximation circuits with the aforementioned features while adhering to quality constraints significantly extends the design cycle. This complexity arises as designers must ensure that circuits not only meet functional and optimal performance criteria but also operate within predefined error margins. To enhance the design of approximation circuits with effective error management, it's essential to employ advanced methods such as Approximate Logic Synthesis (ALS). ALS is geared towards meeting diverse accuracy requirements while also addressing power and delay variances. Integrating ALS, especially AHLS, and automated design exploration tools along with appropriate analytical or semi-analytical error models underscores the necessity for designing quality-configurable circuits that adjust to different accuracy levels in real time. It's also important to extend approximation beyond traditional gate-based designs to include more complex functional units. Though initial research efforts, such as those by Lee [508] and Alan [509] and their colleagues, have started to address these challenges through proposals for approximate high-level synthesis in custom hardware circuit design and runtime accuracy-configurable circuits, respectively, this area is still in its nascent stages. This trend towards customized ML models requires new Auto-ML tools and co-design strategies that integrate algorithmic and hardware considerations for optimal use of approximate computing in advanced ML settings.

C. ADAPTIVE ERROR REDUCTION IN RECONFIGURABLE APPROXIMATE CIRCUIT

The current challenge is that each application requires a specific characteristic of approximations to mention the accuracy within acceptable level. Different approximations have varying effects on the performance and accuracy of application. Intuitively, there is no one-size-fits-all solution. Therefore, the future direction is to design on universal design with adapting itself to reduce error as possible as. Addressing the challenge of reducing errors in input data reconfigurable approximate circuits, particularly with an approximate adder, involves a dynamic and adaptive approach. In such circuits, the configuration of each full adder can be adjusted based on the input data and the carry signal. This adaptability allows the circuit to modify how it performs approximations in real-time, optimizing for accuracy in critical computations while still benefiting from the efficiency of approximation in less critical areas.

For instance, if the approximate adder detects that the input data or the carry signal leads to a potentially significant error, it can reconfigure itself to reduce or eliminate the approximation for that specific calculation. This self-adjusting capability ensures that the circuit maintains a balance between the desired efficiency of approximation techniques and the need for accuracy in the output, particularly for computations where precision is crucial. By dynamically adjusting the level of approximation based on the input data characteristics and the computational context, such reconfigurable approximate circuits can effectively minimize errors while still leveraging the benefits of approximate computing.

In recent years, much of the focus in approximation computing was on a single-layer approach, limiting approximation to specific modules. However, researchers are now aiming to maximize the advantages of approximate computing by integrating various techniques across different design levels, including hardware or software or both, for a given application. This approach, known as cross-layer codesign, represents a significant and ongoing challenge in the field. For example, explores approximation strategies in printed circuits for machine learning, enhancing efficiency and reducing complexity.

XIV. PERSPECTIVES ON FUTURE DIRECTIONS

In recent years, the field of approximate computing (AC) has witnessed significant advancements, positioning it as a potential mainstream computing approach in future systems. One primary reason for this shift is the diminishing returns on performance improvement through the scaling of CMOS technology. Additionally, the diversity of modern architectures, ranging from high-performance computing (HPC) to embedded systems like the Internet of Things (IoT) and autonomous vehicles, necessitates a balance between efficiency in terms of memory, performance, power consumption, and the quality of final outcomes. However, approximate computing is one of the most promising techniques for many future applications, especially those related to human perception [15], [16]. Recent trends indicate its increasing adoption in various domains, including AI-based applications and services, supported by industry leaders like Google and IBM. Major corporations such as IBM, Google, Intel, and ARM are actively engaged in pioneering research and the development of commercial offerings that incorporate approximate computing strategies. For example, Google's Tensor Processing Units (TPUs), which employ an approximate computing technique known as reduced precision to lower energy usage [17]. Google also employs approximate computing strategies in its data centers to optimize energy usage without compromising the quality of service. Another example is that IBM has developed an AI accelerator chip capable of achieving high performance (in TOPS) by integrating multiple and multi-level approximate techniques [13].

Due to these significant advancements achieved by AxC, this motivated other communities in electronic design automation (EDA) and software engineering to develop tools and methodologies to facilitate approximate computing designs. Therefore, developing specialized hardware architectures optimized for AxC, coupled with corresponding software tools and programming models, will be crucial for realizing its full potential. Effective error resilience techniques and error estimation Frameworks to manage and mitigate errors introduced by approximation are essential for ensuring the reliability and robustness of AxC systems. To fully realize the potential of AxC, it is crucial to investigate its application across multiple layers of the computing stack, including hardware, architecture, software, and algorithms. Most of the current research concentrates on error-tolerant applications, but we believe the next research area is to demonstrate the effectiveness of these AxC techniques in safety-critical applications. Therefore, a comprehensive approach to implementing AxC across different layers can unlock new efficiencies and capabilities in modern computing systems.

One of the recent future directions is to build systems employing dynamic, adaptive approximation techniques that can adjust the level of approximation based on application requirements, input data characteristics, and available resources. This ensures optimal trade-offs between accuracy and performance. AxC is well-suited for machine learning and AI applications, where small losses in accuracy can be tolerated in exchange for significant performance gains. Research will focus on developing approximate algorithms and hardware accelerators tailored for these applications. AxC is expected to find applications in various emerging fields, such as IoT, edge computing, and embedded systems, where energy efficiency and real-time performance are critical.

Recent advancements in neuromorphic computing have addressed the power and latency issues of traditional digital systems. The researchers attempt to create more efficient and intelligent computer systems to mimic the human brain by constructing sophisticated hardware architectures and developing new theories and brain-inspired algorithms. Brain-inspired computing faces several significant challenges, holds promising future directions, and directly relates to emerging non-volatile memory (eNVM). eNVM is attractive for implementing the synapses in the neural network [461]. Processing-In-Memory (PIM) is the most attractive architecture used in designing brain-inspired computing models. The brain-inspired computing model is based on the so-called Spiking Neural Networks (SNNs). Recent research highlights the potential of hybrid neural networks (HNNs) in various applications. The emerging trend of designing hybrid neural networks (HNNs) by combining spiking and artificial neural networks leverages the strengths of both. Therefore, Zhao et al. [510] proposed a framework using hybrid units (HUs) to link and integrate multiple neural network structures, especially the integration of spiking neural networks (SNNs) within HNNs. Overall, the future of brain-inspired computing lies in continuing to refine these hybrid models and exploring new materials and architectures to bridge the gap between biological and artificial neural systems. By integrating approximate computing techniques, HNNs can achieve better performance and energy efficiency, making them more viable for large-scale, real-time applications. This opportunity must be exploited by researchers and designers to align with the broader goal of creating scalable and sustainable AI systems that can handle increasingly complex tasks with minimal resources.

We know that one of the primary concerns for communities is ensuring the security and privacy of data, especially when the data is processed using AC techniques. Future crucial research areas focus on developing secure and privacy-preserving AC methodologies. The potential of approximate computing extends beyond traditional AI and signal processing applications. Emerging areas such as hardware security, cryptocurrency mining, and lattice-based post-quantum cryptography are poised to benefit from the efficiency gains offered by approximate computing. These applications require significant computational resources and can tolerate a degree of error, making them ideal candidates for approximate computing techniques. We also believe that one of the challenges that faces approximate computing is the lack of a systematic and theoretical foundation for AC, including formal models for error analysis, performance optimization, and algorithm design. Therefore, establishing a strong theoretical foundation will guide future research and development in this field.

While challenges remain, the ongoing research and development in approximate computing are paving the way for its widespread adoption. By leveraging approximate computing techniques, future systems can achieve significant improvements in energy efficiency and performance, particularly in error-tolerant applications. The growing demand for approximate computing will necessitate diverse contributions from various stakeholders, including hardware designers, system developers, test engineers, and researchers. Collaborative efforts across these disciplines will be essential to advance approximate computing as a mainstream paradigm. This interdisciplinary approach will help address the challenges associated with approximate computing, such as error management, reliability, and user acceptance. As the field continues to evolve, we expect to see more innovative applications and a growing integration of approximate computing into mainstream computing paradigms.

XV. CONCLUSION

In this paper, we explored the state-of-the-art of approximate computing, focusing on its application in data, software, hardware, and architecture, and highlighting its benefits across various fields. It reviews recent progress and challenges in approximate computing, with a detailed examination of its significant impact, particularly in machine learning and IoT. The survey emphasizes the transformative potential of approximate computing in these areas and aims to enrich the research community, offering a valuable reference for researchers. We explored and discussed the state-of-the-art data level of approximation. We focused on the data sampling algorithms used in various frameworks to improve the efficiency and speed of processing large datasets. We discussed programming models and software frameworks (e.g., ApproxHadoop), which are used for processing large datasets across clusters of computers or on the cloud or another example, like BlinkDB, which is specifically designed for approximate queries on large datasets. We review the state-of-the-art data structures, which are not less important than the others because they offer efficiency in data storage and computation. For example, Boom filters are widely used in IOT and wearable electronics, where battery life is a major concern. In this review paper, we expanded our focus on approximations beyond the data level. We performed an extensive analysis of optimizing the code using approximate computing techniques and discussed and categorized the most important types of approximate programming languages. Regarding the architecture level, we discussed the state-of-the-art different approximate memories and emphasized significant innovation in the approximate Processing-In-Memory and Content-Addressable Memory (CAM). Expanding our focus on approximations beyond just memories to explore the last innovate works on processors, especially in the AI domain. At the circuit level, we presented and discussed the state-of-the-art of all arithmetic units, elementary and activation functions, and emphasized in our discussion on approximate logic synthesis.

Regarding the application level, we focused on the emerging IOT, DL/ML, and data mining applications and discussed software, hardware, cross-layer, and end-to-end approximations. We highlighted the traditional use of approximate computing techniques focusing on a single subsystem. The core argument is that to realize the full benefits of approximate computing, we need to move beyond these siloed approaches and focus on a full-system approach through applying approximations strategically across different system layers. There are great advantages to using cross-layer and end-to-end approximations (full-system approach): they can lead to significant improvements in speed, energy consumption, and overall optimization; they can control holistic errors through understanding the propagation of errors throughout the entire system, which enables better management of overall accuracy; and they can tailor the solutions where A full-system view allows for custom-designed approximations matching the specific tolerance and performance requirements of individual applications. We reported well-established libraries and benchmark suites for evaluating the quality-of-service of approximate designs. We presented some open-source tools and logic syntheses. We intended to discuss the security concerns of using approximate computing. Despite advances in approximate computing, there's a critical need for continued innovation to unlock its full potential in complex system designs. Our survey concludes with a discussion on these challenges and future research directions. Establishing standardized benchmarks and error metrics for approximate computing. This will enable researchers and designers to compare different approaches and help users select the most appropriate solution for their use case.

REFERENCES

[1] A. G. M. Strollo and D. Esposito, "Approximate computing in the nanoscale era," in *Proc. Int. Conf. IC Design Technol. (ICICDT)*, Jun. 2018, pp. 21–24, doi: 10.1109/ICICDT.2018.8399746.

[2] A. Najafi, "Systematic design of low-power processing elements using stochastic and approximate computing techniques," Ph.D. thesis, Dept. Phys. Electron. Eng., Univ. Bremen, 2021, doi: 10.26092/elib/460.

[3] H. Esmaeilzadeh, E. Blem, R. St. Amant, K. Sankaralingam, and D. Burger, "Dark silicon and the end of multicore scaling," in *Proc. 38th Annu. Int. Symp. Comput. Archit. (ISCA)*, Jun. 2011, pp. 365–376.

[4] N. Hardavellas, M. Ferdman, B. Falsafi, and A. Ailamaki, "Toward dark silicon in servers," *IEEE Micro*, vol. 31, no. 4, pp. 6–15, Jul. 2011, doi: 10.1109/MM.2011.77.

[5] M. Shafique, S. Garg, J. Henkel, and D. Marculescu, "The EDA challenges in the dark silicon era," in *Proc. 51st ACM/EDAC/IEEE Design Autom. Conf. (DAC)*, Jun. 2014, pp. 1–6, doi: 10.1145/2593069.2593229.

[6] J. L. Hennessy and D. A. Patterson, "A new golden age for computer architecture," *Commun. ACM*, vol. 62, no. 2, pp. 48–60, Jan. 2019, doi: 10.1145/3282307.

[7] K. Rupp. (2024). *Karlrupp/Microprocessor-Trend-Data*. Accessed: Sep. 2, 2024. [Online]. Available: https://github.com/karlrupp/microprocessor-trend-data

[8] S. Dutt, S. Nandi, and G. Trivedi, "A comparative survey of approximate adders," in *Proc. 26th Int. Conf. Radioelektronika (RADIOELEKTRONIKA)*, Apr. 2016, pp. 61–65, doi: 10.1109/RADIOELEK.2016.7477392.

[9] S. Venkataramani, S. T. Chakradhar, K. Roy, and A. Raghunathan, "Approximate computing and the quest for computing efficiency," in *Proc. 52nd ACM/EDAC/IEEE Design Autom. Conf. (DAC)*, Jun. 2015, pp. 1–6, doi: 10.1145/2744769.2744904.

[10] M. D. Hill and M. R. Marty, "Amdahl's law in the multicore era," *Computer*, vol. 41, no. 7, pp. 33–38, Jul. 2008, doi: 10.1109/mc.2008.209.

[11] M. Shafique and S. Garg, "Computing in the dark silicon era: Current trends and research challenges," *IEEE Des. Test. IEEE Des. Test. Comput.*, vol. 34, no. 2, pp. 8–23, Apr. 2017, doi: 10.1109/MDAT.2016.2633408.

[12] S. Liu, K. Pattabiraman, T. Moscibroda, and B. G. Zorn, "Flikker: Saving DRAM refresh-power through critical data partitioning," *ACM SIGPLAN Notices*, vol. 46, no. 3, pp. 213–224, Mar. 2011, doi: 10.1145/1961296.1950391.

[13] W. Liu, F. Lombardi, and M. Shulte, "A retrospective and prospective view of approximate computing [point of view]," *Proc. IEEE*, vol. 108, no. 3, pp. 394–399, Mar. 2020, doi: 10.1109/JPROC.2020.2975695.

[14] V. K. Chippa, S. T. Chakradhar, K. Roy, and A. Raghunathan, "Analysis and characterization of inherent application resilience for approximate computing," in *Proc. 50th ACM/EDAC/IEEE Design Autom. Conf. (DAC)*. New York, NY, USA: ACM, May 2013, pp. 1–9, doi: 10.1145/2463209.2488873.

[15] A. Dalloo, "Enhance the segmentation principle in approximate computing," in *Proc. Int. Conf. Circuits Syst. Digit. Enterprise Technol. (ICCSDET)*, Dec. 2018, pp. 1–7, doi: 10.1109/ICCSDET.2018.8821112.

[16] A. Dalloo, A. Najafi, and A. Garcia-Ortiz, "Systematic design of an approximate adder: The optimized lower part constant-OR adder," *IEEE Trans. Very Large Scale Integr. (VLSI) Syst.*, vol. 26, no. 8, pp. 1595–1599, Aug. 2018, doi: 10.1109/TVLSI.2018.2822278.

[17] N. P. Jouppi et al., "In-datacenter performance analysis of a tensor processing unit," in *Proc. ACM/IEEE 44th Annu. Int. Symp. Comput. Archit. (ISCA)*. New York, NY, USA: ACM, Jun. 2017, pp. 1–12, doi: 10.1145/3079856.3080246.

[18] N. Enright Jerger and J. San Miguel, "Approximate computing," *IEEE Micro*, vol. 38, no. 4, pp. 8–10, Jul. 2018, doi: 10.1109/MM.2018.043191120.

[19] Q. Xu, T. Mytkowicz, and N. S. Kim, "Approximate computing: A survey," *IEEE Des. Test. Comput.*, vol. 33, no. 1, pp. 8–22, Feb. 2016, doi: 10.1109/MDAT.2015.2505723.

[20] D. Mohapatra, V. K. Chippa, A. Raghunathan, and K. Roy, "Design of voltage-scalable meta-functions for approximate computing," in *Proc. Design, Autom. Test Eur.*, Mar. 2011, pp. 1–6, doi: 10.1109/DATE.2011.5763154.

[21] A. Rahimi, A. Ghofrani, K.-T. Cheng, L. Benini, and R. K. Gupta, "Approximate associative memristive memory for energy-efficient GPUs," in *Proc. Design, Autom. Test Eur. Conf. Exhib. (DATE)*, Mar. 2015, pp. 1497–1502, doi: 10.7873/DATE.2015.0579.

[22] G. Rodrigues, F. Lima Kastensmidt, and A. Bosio, "Survey on approximate computing and its intrinsic fault tolerance," *Electronics*, vol. 9, no. 4, p. 557, Mar. 2020, doi: 10.3390/electronics9040557.

[23] T. Alan, A. Gerstlauer, and J. Henkel, "Cross-layer approximate hardware synthesis for runtime configurable accuracy," *IEEE Trans. Very Large Scale Integr. (VLSI) Syst.*, vol. 29, no. 6, pp. 1231–1243, Apr. 2021, doi: 10.1109/TVLSI.2021.3068312.

[24] Z. Ebrahimi, D. Klar, M. A. Ekhtiyar, and A. Kumar, "Plasticine: A cross-layer approximation methodology for multi-kernel applications through minimally biased, high-throughput, and energy-efficient SIMD soft multiplier-divider," *ACM Trans. Design Autom. Electron. Syst.*, vol. 27, no. 2, pp. 1–33, Nov. 2021, doi: 10.1145/3486616.

[25] M. A. Hanif and M. Shafique, "A cross-layer approach towards developing efficient embedded deep learning systems," *Microprocessors Microsyst.*, vol. 88, Feb. 2022, Art. no. 103609, doi: 10.1016/j.micpro.2020.103609.

[26] S. Mittal, "A survey of techniques for approximate computing," *ACM Comput. Surv.*, vol. 48, no. 4, pp. 1–33, Mar. 2016, doi: 10.1145/2893356.

[27] H. Jiang, C. Liu, L. Liu, F. Lombardi, and J. Han, "A review, classification, and comparative evaluation of approximate arithmetic circuits," *ACM J. Emerg. Technol. Comput. Syst.*, vol. 13, no. 4, pp. 1–34, Aug. 2017, doi: 10.1145/3094124.

[28] F. Betzel, K. Khatamifard, H. Suresh, D. J. Lilja, J. Sartori, and U. Karpuzcu, "Approximate communication: Techniques for reducing communication bottlenecks in large-scale parallel systems," *ACM Comput. Surv.*, vol. 51, no. 1, pp. 1–32, Jan. 2018, doi: 10.1145/3145812.

[29] G. Zervakis, H. Saadat, H. Amrouch, A. Gerstlauer, S. Parameswaran, and J. Henkel, "Approximate computing for ML: State-of-the-art, challenges and visions," in *Proc. 26th Asia South Pacific Design Autom. Conf. (ASP-DAC)*, Jan. 2021, pp. 189–196.

[30] J. Henkel, H. Li, A. Raghunathan, M. B. Tahoori, S. Venkataramani, X. Yang, and G. Zervakis, "Approximate computing and the efficient machine learning expedition," in *Proc. IEEE/ACM Int. Conf. Comput. Aided Design (ICCAD)*. New York, NY, USA: ACM, Oct. 2022, pp. 1–9.

[31] J. Lee, L. Mukhanov, A. S. Molahosseini, U. Minhas, Y. Hua, J. M. del Rincon, K. Dichev, C.-H. Hong, and H. Vandierendonck, "Resource-efficient convolutional networks: A survey on model-, arithmetic-, and implementation-level techniques," *ACM Comput. Surveys*, vol. 55, no. 13s, pp. 1–36, Dec. 2023, doi: 10.1145/3587095.

[32] H.-H. Que, Y. Jin, T. Wang, M.-K. Liu, X.-H. Yang, and F. Qiao, "A survey of approximate computing: From arithmetic units design to high-level applications," *J. Comput. Sci. Technol.*, vol. 38, no. 2, pp. 251–272, Apr. 2023, doi: 10.1007/s11390-023-2537-y.

[33] H. J. Damsgaard, A. Ometov, and J. Nurmi, "Approximation opportunities in edge computing hardware: A systematic literature review," *ACM Comput. Surv.*, vol. 55, no. 12, pp. 1–49, Mar. 2023, doi: 10.1145/3572772.

[34] V. Leon, M. Abdullah Hanif, G. Armeniakos, X. Jiao, M. Shafique, K. Pekmestzi, and D. Soudris, ''Approximate computing survey, Part I: Terminology and software & hardware approximation techniques,'' 2023, *arXiv:2307.11124*.

[35] V. Leon, M. Abdullah Hanif, G. Armeniakos, X. Jiao, M. Shafique, K. Pekmestzi, and D. Soudris, ''Approximate computing survey, Part II: Application-specific & architectural approximation techniques and applications,'' 2023, *arXiv:2307.11128*.

[36] K. K. Pandey and D. Shukla, ''Stratified sampling-based data reduction and categorization model for big data mining,'' in *Communication and Intelligent Systems* (Lecture Notes in Networks and Systems), J. C. Bansal, M. K. Gupta, H. Sharma, and B. Agarwal, Eds., Singapore: Springer, 2020, pp. 107–122, doi: 10.1007/978-981-15-3325-9_9.

[37] T. D. Nguyen, M.-H. Shih, D. Srivastava, S. Tirthapura, and B. Xu, ''Stratified random sampling from streaming and stored data,'' *Distrib. Parallel Databases*, vol. 39, no. 3, pp. 665–710, Sep. 2021, doi: 10.1007/s10619-020-07315-w.

[38] J. Zhang, H. Chen, D. Yu, Y. Pei, and Y. Deng, ''Cluster-preserving sampling algorithm for large-scale graphs,'' *Sci. China Inf. Sci.*, vol. 66, no. 1, Nov. 2022, Art. no. 112103, doi: 10.1007/s11432-021-3370-4.

[39] S. Shankar and A. G. Parameswaran, ''Towards observability for production machine learning pipelines,'' *Proc. VLDB Endowment*, vol. 15, no. 13, pp. 4015–4022, Sep. 2022, doi: 10.14778/3565838.3565853.

[40] B. G. Galuzzi, L. Milazzo, and C. Damiani, ''Best practices in flux sampling of constrained-based models,'' in *Machine Learning, Optimization, and Data Science* (Lecture Notes in Computer Science), G. Nicosia, V. Ojha, E. La Malfa, G. La Malfa, P. Pardalos, G. Di Fatta, G. Giuffrida, and R. Umeton, Eds., Cham, Switzerland: Springer, 2023, pp. 234–248, doi: 10.1007/978-3-031-25891-6_18.

[41] N. Sobhani and S. J. Delany, ''Identity term sampling for measuring gender bias in training data,'' in *Artificial Intelligence and Cognitive Science* (Communications in Computer and Information Science). Cham, Switzerland: Springer, 2023, pp. 226–238, doi: 10.1007/978-3-031-26438-2_18.

[42] H. Wu, H. Xu, X. Tian, W. Zhang, and C. Lu, ''Multistage sampling and optimization for forest volume inventory based on spatial autocorrelation analysis,'' *Forests*, vol. 14, no. 2, p. 250, Jan. 2023, doi: 10.3390/f14020250.

[43] B. Zhang, Y. Du, H. Huang, Y.-E. Sun, G. Gao, X. Wang, and S. Chen, ''Multi-layer adaptive sampling for per-flow spread measurement,'' in *Algorithms and Architectures for Parallel Processing* (Lecture Notes in Computer Science), Y. Lai, T. Wang, M. Jiang, G. Xu, W. Liang, and A. Castiglione, Eds., Cham, Switzerland: Springer, 2022, pp. 743–758, doi: 10.1007/978-3-030-95384-3_46.

[44] S. Moshtaghi Largani and S. Lee, ''Efficient sampling for big provenance,'' in *Proc. Companion ACM Web Conf.* New York, NY, USA: ACM, Apr. 2023, pp. 1508–1511, doi: 10.1145/3543873.3587556.

[45] V. Sanca and A. Ailamaki, ''Sampling-based AQP in modern analytical engines,'' in *Data Management on New Hardware*. New York, NY, USA: ACM, Jun. 2022, pp. 1–8, doi: 10.1145/3533737.3535095.

[46] E. A. Deiana, V. St-Amour, P. A. Dinda, N. Hardavellas, and S. Campanoni, ''Unconventional parallelization of nondeterministic applications,'' in *Proc. 23rd Int. Conf. Architectural Support Program. Lang. Operating Syst.* New York, NY, USA: ACM, Mar. 2018, pp. 432–447, doi: 10.1145/3173162.3173181.

[47] N. Laptev, K. Zeng, and C. Zaniolo, ''Early accurate results for advanced analytics on MapReduce,'' 2012, *arXiv:1207.0142*.

[48] I. Goiri, R. Bianchini, S. Nagarakatte, and T. D. Nguyen, ''ApproxHadoop: Bringing approximations to MapReduce frameworks,'' *ACM SIGPLAN Notices*, vol. 50, no. 4, pp. 383–397, Mar. 2015, doi: 10.1145/2775054.2694351.

[49] G. Hu, D. Zhang, S. Rigo, and T. D. Nguyen, ''Approximation with error bounds in spark,'' 2018, *arXiv:1812.01823*.

[50] D. L. Quoc, R. Chen, P. Bhatotia, C. Fetzer, V. Hilt, and T. Strufe, ''StreamApprox: Approximate computing for stream analytics,'' in *Proc. 18th ACM/IFIP/USENIX Middleware Conf.*, Dec. 2017, pp. 185–197, doi: 10.1145/3135974.3135989.

[51] Z. Wen, D. L. Quoc, P. Bhatotia, R. Chen, and M. Lee, ''ApproxIoT: Approximate analytics for edge computing,'' in *Proc. IEEE 38th Int. Conf. Distrib. Comput. Syst. (ICDCS)*, Jul. 2018, pp. 411–421, doi: 10.1109/ICDCS.2018.00048.

[52] Y. Park, J. Qing, X. Shen, and B. Mozafari, ''BlinkML: Efficient maximum likelihood estimation with probabilistic guarantees,'' in *Proc. Int. Conf. Manage. Data*, Jun. 2019, pp. 1135–1152, doi: 10.1145/3299869.3300077.

[53] M. R. Anderson and M. Cafarella, ''Input selection for fast feature engineering,'' in *Proc. IEEE 32nd Int. Conf. Data Eng. (ICDE)*, May 2016, pp. 577–588, doi: 10.1109/ICDE.2016.7498272.

[54] Z. Carmichael, H. F. Langroudi, C. Khazanov, J. Lillie, J. L. Gustafson, and D. Kudithipudi, ''Performance-efficiency trade-off of low-precision numerical formats in deep neural networks,'' 2019, *arXiv:1903.10584*.

[55] S. Cherubin and G. Agosta, ''Tools for reduced precision computation: A survey,'' *ACM Comput. Surv.*, vol. 53, no. 2, pp. 1–35, Apr. 2020, doi: 10.1145/3381039.

[56] A. Gholami, S. Kim, Z. Dong, Z. Yao, M. W. Mahoney, and K. Keutzer, ''A survey of quantization methods for efficient neural network inference,'' 2021, *arXiv:2103.13630*.

[57] T. Zebin, P. J. Scully, N. Peek, A. J. Casson, and K. B. Ozanyan, ''Design and implementation of a convolutional neural network on an edge computing smartphone for human activity recognition,'' *IEEE Access*, vol. 7, pp. 133509–133520, 2019, doi: 10.1109/ACCESS.2019.2941836.

[58] *Quantization Aware Training | TensorFlow Model Optimization*. Accessed: Mar. 15, 2023. [Online]. Available: https://www.tensorflow.org/model_optimization/guide/quantization/training

[59] P.-E. Novac, G. Boukli Hacene, A. Pegatoquet, B. Miramond, and V. Gripon, ''Quantization and deployment of deep neural networks on microcontrollers,'' *Sensors*, vol. 21, no. 9, p. 2984, Apr. 2021, doi: 10.3390/s21092984.

[60] J. Zhai, B. Li, S. Lv, and Q. Zhou, ''FPGA-based vehicle detection and tracking accelerator,'' *Sensors*, vol. 23, no. 4, p. 2208, Feb. 2023, doi: 10.3390/s23042208.

[61] P. Colangelo, N. Nasiri, E. Nurvitadhi, A. Mishra, M. Margala, and K. Nealis, ''Exploration of low numeric precision deep learning inference using Intel FPGAs,'' in *Proc. IEEE 26th Annu. Int. Symp. Field-Program. Custom Comput. Mach. (FCCM)*, Apr. 2018, pp. 73–80, doi: 10.1109/FCCM.2018.00020.

[62] G. Dai and J. Fan, ''An industrial-grade solution for crop disease image detection tasks,'' *Frontiers Plant Sci.*, vol. 13, pp. 1–12, Jun. 2022. [Online]. Available: https://www.frontiersin.org/articles/10.3389/fpls.2022.921057

[63] M. M. Farag, ''A self-contained STFT CNN for ECG classification and arrhythmia detection at the edge,'' *IEEE Access*, vol. 10, pp. 94469–94486, 2022, doi: 10.1109/ACCESS.2022.3204703.

[64] D. Costa, M. Costa, and S. Pinto, ''Train me if you can: Decentralized learning on the deep edge,'' *Appl. Sci.*, vol. 12, no. 9, p. 4653, May 2022, doi: 10.3390/app12094653.

[65] P. Micikevicius, S. Narang, J. Alben, G. Diamos, E. Elsen, D. Garcia, B. Ginsburg, M. Houston, O. Kuchaiev, G. Venkatesh, and H. Wu, ''Mixed precision training,'' 2017, *arXiv:1710.03740*.

[66] *Train With Mixed Precision*. Accessed: Jun. 8, 2023. [Online]. Available: https://docs.nvidia.com/deeplearning/performance/mixed-precision-training/index.html

[67] S. Kang, K. Choi, and Y. Park, ''PreScaler: An efficient system-aware precision scaling framework on heterogeneous systems,'' in *Proc. 18th ACM/IEEE Int. Symp. Code Gener. Optim.*, vol. 48. New York, NY, USA: ACM, Feb. 2020, pp. 280–292, doi: 10.1145/3368826.3377917.

[68] S. Yesil, I. Akturk, and U. R. Karpuzcu, ''Toward dynamic precision scaling,'' *IEEE Micro*, vol. 38, no. 4, pp. 30–39, Jul. 2018, doi: 10.1109/MM.2018.043191123.

[69] W.-F. Chiang, M. Baranowski, I. Briggs, A. Solovyev, G. Gopalakrishnan, and Z. Rakamarić, ''Rigorous floating-point mixed-precision tuning,'' *ACM SIGPLAN Notices*, vol. 52, no. 1, pp. 300–315, Jan. 2017, doi: 10.1145/3093333.3009846.

[70] P. V. Kotipalli, R. Singh, P. Wood, I. Laguna, and S. Bagchi, ''AMPT-GA: Automatic mixed precision floating point tuning for GPU applications,'' in *Proc. ACM Int. Conf. Supercomputing*. New York, NY, USA: ACM, Jun. 2019, pp. 160–170, doi: 10.1145/3330345.3330360.

[71] H. Guo and C. Rubio-González, ''Exploiting community structure for floating-point precision tuning,'' in *Proc. 27th ACM SIGSOFT Int. Symp. Softw. Test. Anal.* New York, NY, USA: ACM, Jul. 2018, pp. 333–343, doi: 10.1145/3213846.3213862.

[72] S. Garg, J. Lou, A. Jain, Z. Guo, B. J. Shastri, and M. Nahmias, ''Dynamic precision analog computing for neural networks,'' *IEEE J. Sel. Topics Quantum Electron.*, vol. 29, no. 2, pp. 1–12, Mar. 2023, doi: 10.1109/JSTQE.2022.3218019.

[73] G. Giamougiannis, A. Tsakyridis, M. Moralis-Pegios, C. Pappas, M. Kirtas, N. Passalis, D. Lazovsky, A. Tefas, and N. Pleros, ''Analog nanophotonic computing going practical: Silicon photonic deep learning engines for tiled optical matrix multiplication with dynamic precision,'' *Nanophotonics*, vol. 12, no. 5, pp. 963–973, Mar. 2023, doi: 10.1515/nanoph-2022-0423.

[74] W. Fornaciari, G. Agosta, D. Cattaneo, L. Denisov, A. Galimberti, G. Magnani, and D. Zoni, ''Hardware and software support for mixed precision computing: A roadmap for embedded and HPC systems,'' in *Proc. Design, Autom. Test Eur. Conf. Exhib. (DATE)*, Apr. 2023, pp. 1–6, doi: 10.23919/date56975.2023.10137092.

[75] S. Yamagiwa, W. Yang, and K. Wada, ''Adaptive lossless image data compression method inferring data entropy by applying deep neural network,'' *Electronics*, vol. 11, no. 4, p. 504, Feb. 2022, doi: 10.3390/electronics11040504.

[76] H. M. Yasin and S. Y. Ameen, ''Review and evaluation of end-to-end video compression with deep-learning,'' in *Proc. Int. Conf. Mod. Trends Inf. Commun. Technol. Ind. (MTICTI)*, Dec. 2021, pp. 1–8, doi: 10.1109/MTICTI53925.2021.9664790.

[77] C. Ma, D. Liu, X. Peng, L. Li, and F. Wu, ''Convolutional neural network-based arithmetic coding for HEVC intra-predicted residues,'' *IEEE Trans. Circuits Syst. Video Technol.*, vol. 30, no. 7, pp. 1901–1916, Jul. 2020, doi: 10.1109/TCSVT.2019.2927027.

[78] S. Wiedemann, H. Kirchhoffer, S. Matlage, P. Haase, A. Marban, T. Marinc, D. Neumann, A. Osman, D. Marpe, H. Schwarz, T. Wiegand, and W. Samek, ''DeepCABAC: Context-adaptive binary arithmetic coding for deep neural network compression,'' 2019, *arXiv:1905.08318*.

[79] Y. He, J. Lin, Z. Liu, H. Wang, L.-J. Li, and S. Han, ''AMC: AutoML for model compression and acceleration on mobile devices,'' in *Computer Vision—ECCV 2018* (Lecture Notes in Computer Science), V. Ferrari, M. Hebert, C. Sminchisescu, and Y. Weiss, Eds., Cham, Switzerland: Springer, 2018, pp. 815–832, doi: 10.1007/978-3-030-01234-2_48.

[80] D. Dai, C. Dong, S. Xu, Q. Yan, Z. Li, C. Zhang, and N. Luo, ''Ms RED: A novel multi-scale residual encoding and decoding network for skin lesion segmentation,'' *Med. Image Anal.*, vol. 75, Jan. 2022, Art. no. 102293, doi: 10.1016/j.media.2021.102293.

[81] D. G. Cortés, E. Onieva, I. P. López, L. Trinchera, and J. Wu, ''Autoencoder-enhanced clustering: A dimensionality reduction approach to financial time series,'' *IEEE Access*, vol. 12, pp. 16999–17009, 2024, doi: 10.1109/ACCESS.2024.3359413.

[82] Z. Duan, M. Lu, J. Ma, Y. Huang, Z. Ma, and F. Zhu, ''QARV: Quantization-aware ResNet VAE for lossy image compression,'' *IEEE Trans. Pattern Anal. Mach. Intell.*, vol. 46, no. 1, pp. 436–450, Jan. 2024, doi: 10.1109/TPAMI.2023.3322904.

[83] H. Zhang, Z. Hu, C. Luo, W. Zuo, and M. Wang, ''Semantic image inpainting with progressive generative networks,'' in *Proc. 26th ACM Int. Conf. Multimedia*. New York, NY, USA: ACM, Oct. 2018, pp. 1939–1947, doi: 10.1145/3240508.3240625.

[84] Y. Yang, K. Zheng, B. Wu, Y. Yang, and X. Wang, ''Network intrusion detection based on supervised adversarial variational auto-encoder with regularization,'' *IEEE Access*, vol. 8, pp. 42169–42184, 2020, doi: 10.1109/ACCESS.2020.2977007.

[85] S. Wiedemann, H. Kirchhoffer, S. Matlage, P. Haase, A. Marban, T. Marinc, D. Neumann, T. Nguyen, H. Schwarz, T. Wiegand, D. Marpe, and W. Samek, ''DeepCABAC: A universal compression algorithm for deep neural networks,'' *IEEE J. Sel. Topics Signal Process.*, vol. 14, no. 4, pp. 700–714, May 2020, doi: 10.1109/JSTSP.2020.2969554.

[86] X. Wang, Z. Liu, Y. Gao, X. Zheng, X. Chen, and C. Wu, ''Near-optimal data structure for approximate range emptiness problem in information-centric Internet of Things,'' *IEEE Access*, vol. 7, pp. 21857–21869, 2019, doi: 10.1109/ACCESS.2019.2897154.

[87] P. H. Chia, D. Desfontaines, I. M. Perera, D. Simmons-Marengo, C. Li, W.-Y. Day, Q. Wang, and M. Guevara, ''KHyperLogLog: Estimating reidentifiability and joinability of large data at scale,'' in *Proc. IEEE Symp. Secur. Privacy (SP)*, May 2019, doi: 10.1109/SP.2019.00046.

[88] X. Yang, A. Vernitski, and L. Carrea, ''An approximate dynamic programming approach for improving accuracy of lossy data compression by Bloom filters,'' *Eur. J. Oper. Res.*, vol. 252, no. 3, pp. 985–994, Aug. 2016, doi: 10.1016/j.ejor.2016.01.042.

[89] R. Patgiri, A. Biswas, and S. Nayak, ''DeepBF: Malicious URL detection using learned Bloom filter and evolutionary deep learning,'' 2021, *arXiv:2103.12544*.

[90] B. H. Bloom, ''Space/time trade-offs in hash coding with allowable errors,'' *Commun. ACM*, vol. 13, no. 7, pp. 422–426, Jul. 1970, doi: 10.1145/362686.362692.

[91] S. Z. Kiss, É. Hosszu, J. Tapolcai, L. Rónyai, and O. Rottenstreich, ''Bloom filter with a false positive free zone,'' *IEEE Trans. Netw. Service Manage.*, vol. 18, no. 2, pp. 2334–2349, Jun. 2021, doi: 10.1109/TNSM.2021.3059075.

[92] Y. Wu, J. He, S. Yan, J. Wu, T. Yang, O. Ruas, G. Zhang, and B. Cui, ''Elastic Bloom filter: Deletable and expandable filter using elastic fingerprints,'' *IEEE Trans. Comput.*, vol. 71, no. 4, pp. 984–991, Apr. 2022, doi: 10.1109/TC.2021.3067713.

[93] F. G. Gebretsadik, S. Nayak, and R. Patgiri, ''EBF: An enhanced Bloom filter for intrusion detection in IoT,'' *J. Big Data*, vol. 10, no. 1, p. 102, Jun. 2023, doi: 10.1186/s40537-023-00790-9.

[94] H. A. Seymen and M. E. Yalçın, ''Design and implementation of a lightweight Bloom filter accelerator for IoT applications,'' in *Proc. 14th Int. Conf. Electr. Electron. Eng. (ELECO)*, Nov. 2023, pp. 1–5, doi: 10.1109/eleco60389.2023.10415987.

[95] L. Luo, D. Guo, R. T. B. Ma, O. Rottenstreich, and X. Luo, ''Optimizing Bloom filter: Challenges, solutions, and comparisons,'' 2018, *arXiv:1804.04777*.

[96] A. Singh, S. Garg, R. Kaur, S. Batra, N. Kumar, and A. Y. Zomaya, ''Probabilistic data structures for big data analytics: A comprehensive review,'' *Knowl.-Based Syst.*, vol. 188, Jan. 2020, Art. no. 104987, doi: 10.1016/j.knosys.2019.104987.

[97] P. Reviriego, P. Junsangsri, S. Liu, and F. Lombardi, ''Error-tolerant data sketches using approximate nanoscale memories and voltage scaling,'' *IEEE Trans. Nanotechnol.*, vol. 21, pp. 16–22, 2022, doi: 10.1109/TNANO.2021.3139394.

[98] F. Deng and D. Rafiei, ''New estimation algorithms for streaming data: Count-min can do more,'' Webdocs.Cs.Ualberta.Ca, Univ. Alberta, Edmonton, AB, Canada, 2007. [Online]. Available: http://www.cs.ualberta.ca/~fandeng/paper/cmm.pdf

[99] G. Pitel and G. Fouquier, ''Count-min-log sketch: Approximately counting with approximate counters,'' 2015, *arXiv:1502.04885*.

[100] Z. Wei, Y. Tian, W. Chen, L. Gu, and X. Zhang, ''DUNE: Improving accuracy for sketch-INT network measurement systems,'' 2022, *arXiv:2212.04816*.

[101] T. Yang, J. Jiang, P. Liu, Q. Huang, J. Gong, Y. Zhou, R. Miao, X. Li, and S. Uhlig, ''Elastic sketch: Adaptive and fast network-wide measurements,'' in *Proc. Conf. ACM Special Interest Group Data Commun.* New York, NY, USA: ACM, Aug. 2018, pp. 561–575, doi: 10.1145/3230543.3230544.

[102] K. Zhao, J. Wang, H. Qi, X. Xie, X. Zhou, and K. Li, ''HBL-sketch: A new three-tier sketch for accurate network measurement,'' in *Algorithms and Architectures for Parallel Processing* (Lecture Notes in Computer Science), S. Wen, A. Zomaya, and L. T. Yang, Eds., Cham, Switzerland: Springer, 2020, pp. 48–59, doi: 10.1007/978-3-030-38991-8_4.

[103] T. Yang, S. Gao, Z. Sun, Y. Wang, Y. Shen, and X. Li, ''Diamond sketch: Accurate per-flow measurement for big streaming data,'' *IEEE Trans. Parallel Distrib. Syst.*, vol. 30, no. 12, pp. 2650–2662, Dec. 2019, doi: 10.1109/TPDS.2019.2923772.

[104] J. Zhu, J. Jin, Z. Gao, and P. Reviriego, ''Single event transient tolerant count min sketches,'' *Microelectron. Rel.*, vol. 129, Feb. 2022, Art. no. 114486, doi: 10.1016/j.microrel.2022.114486.

[105] G. Cormode and S. Muthukrishnan, ''An improved data stream summary: The count-min sketch and its applications,'' *J. Algorithms*, vol. 55, no. 1, pp. 58–75, Apr. 2005, doi: 10.1016/j.jalgor.2003.12.001.

[106] A. Ebrahim, ''High-level design optimizations for implementing data stream sketch frequency estimators on FPGAs,'' *Electronics*, vol. 11, no. 15, p. 2399, Jul. 2022, doi: 10.3390/electronics11152399.

[107] A. Khan and S. Yan, ''Composite hashing for data stream sketches,'' 2018, *arXiv:1808.06800*.

[108] N. Seleznev, S. Kumar, and C. B. Bruss, ''Double-hashing algorithm for frequency estimation in data streams,'' 2022, *arXiv:2204.00650*.

[109] P. Tyagi, M. C. Malta, and A. Dutta, ''Hashing for cleaner reverse engineered queries for the entity comparison problem in RDF graphs,'' in *Proc. IEEE/WIC/ACM Int. Joint Conf. Web Intell. Intell. Agent Technol. (WI-IAT)*, Dec. 2020, pp. 177–186, doi: 10.1109/WIIAT50758.2020.00028.

[110] X. Zhu, G. Wu, H. Zhang, S. Wang, and B. Ma, ''Dynamic count-min sketch for analytical queries over continuous data streams,'' in *Proc. IEEE 25th Int. Conf. High Perform. Comput. (HiPC)*, Dec. 2018, pp. 225–234, doi: 10.1109/HIPC.2018.00033.

[111] P. Jia, P. Wang, J. Zhao, J. Tao, Y. Yuan, and X. Guan, ''Erasable virtual HyperLogLog for approximating cumulative distribution over data streams,'' *IEEE Trans. Knowl. Data Eng.*, vol. 34, no. 11, pp. 5336–5350, Nov. 2022. [Online]. Available: https://ieeexplore.ieee.org/document/9328544

[112] K. G. Paterson and M. Raynal, ''HyperLogLog: Exponentially bad in adversarial settings,'' in *Proc. IEEE 7th Eur. Symp. Secur. Privacy*, Jun. 2022, pp. 154–170, doi: 10.1109/EuroSP53844.2022.00018.

[113] S. Heule, M. Nunkesser, and A. Hall, ''HyperLogLog in practice: Algorithmic engineering of a state of the art cardinality estimation algorithm,'' in *Proc. 16th Int. Conf. Extending Database Technol.* New York, NY, USA: ACM, Mar. 2013, pp. 683–692, doi: 10.1145/2452376.2452456.

[114] M. Karppa and R. Pagh, ''HyperLogLogLog: Cardinality estimation with one log more,'' 2022, *arXiv:2205.11327*.

[115] Q. Xiao, S. Chen, Y. Zhou, and J. Luo, ''Estimating cardinality for arbitrarily large data stream with improved memory efficiency,'' *IEEE/ACM Trans. Netw.*, vol. 28, no. 2, pp. 433–446, Apr. 2020, doi: 10.1109/TNET.2020.2970860.

[116] J. Xu, ''Cardinalities estimation under sliding time window by sharing HyperLogLog counter,'' 2018, *arXiv:1810.13132*.

[117] O. Ertl, ''New cardinality estimation algorithms for HyperLogLog sketches,'' 2017, *arXiv:1702.01284*.

[118] W. Li, Y. Zhang, Y. Sun, W. Wang, M. Li, W. Zhang, and X. Lin, ''Approximate nearest neighbor search on high dimensional data—Experiments, analyses, and improvement,'' *IEEE Trans. Knowl. Data Eng.*, vol. 32, no. 8, pp. 1475–1488, Aug. 2020, doi: 10.1109/TKDE.2019.2909204.

[119] O. Ertl, ''SetSketch: Filling the gap between MinHash and HyperLogLog,'' *Proc. VLDB Endowment*, vol. 14, no. 11, pp. 2244–2257, Jul. 2021, doi: 10.14778/3476249.3476276.

[120] Y. William Yu and G. M. Weber, ''HyperMinHash: MinHash in LogLog space,'' 2017, *arXiv:1710.08436*.

[121] T. Dunning, ''The t-digest: Efficient estimates of distributions,'' *Softw. Impacts*, vol. 7, Feb. 2021, Art. no. 100049, doi: 10.1016/j.simpa.2020.100049.

[122] B. W. Ford. (2022). *An Instruction Profiling Based Framework to Promote Software Portability*. Accessed: Mar. 6, 2023. [Online]. Available: https://digital.library.txstate.edu/handle/10877/15757

[123] A. Mercat, J. Bonnot, M. Pelcat, W. Hamidouche, and D. Menard, ''Exploiting computation skip to reduce energy consumption by approximate computing, an HEVC encoder case study,'' in *Proc. Design, Autom. Test Eur. Conf. Exhib. (DATE)*. Lausanne, Switzerland: IEEE, Mar. 2017, pp. 494–499, doi: 10.23919/DATE.2017.7927039.

[124] Y. Lin, C. Sakr, Y. Kim, and N. Shanbhag, ''PredictiveNet: An energy-efficient convolutional neural network via zero prediction,'' in *Proc. IEEE Int. Symp. Circuits Syst. (ISCAS)*, May 2017, pp. 1–4, doi: 10.1109/ISCAS.2017.8050797.

[125] E. Eskandarnia, H. M. Al-Ammal, and R. Ksantini, ''An embedded deep-clustering-based load profiling framework,'' *Sustain. Cities Soc.*, vol. 78, Mar. 2022, Art. no. 103618, doi: 10.1016/j.scs.2021.103618.

[126] S. Li, S. Park, and S. Mahlke, ''Sculptor: Flexible approximation with selective dynamic loop perforation,'' in *Proc. Int. Conf. Supercomputing*, vol. 156. New York, NY, USA: ACM, Jun. 2018, pp. 341–351, doi: 10.1145/3205289.3205317.

[127] D. Maier and B. Juurlink, ''Model-based loop perforation,'' in *Proc. Eur. Conf. Parallel Process.*, Lisbon, Portugal. Berlin, Germany: Springer, Aug. 2021, pp. 549–554, doi: 10.1007/978-3-031-06156-1_48.

[128] H. Omar, M. Ahmad, and O. Khan, ''GraphTuner: An input dependence aware loop perforation scheme for efficient execution of approximated graph algorithms,'' in *Proc. IEEE Int. Conf. Comput. Design (ICCD)*, Nov. 2017, pp. 201–208, doi: 10.1109/ICCD.2017.38.

[129] O. Kislal and M. T. Kandemir, ''Data access skipping for recursive partitioning methods,'' *Comput. Lang., Syst. Struct.*, vol. 53, pp. 143–162, Sep. 2018, doi: 10.1016/j.cl.2018.03.003.

[130] V. Y. Raparti and S. Pasricha, ''Approximate NoC and memory controller architectures for GPGPU accelerators,'' *IEEE Trans. Parallel Distrib. Syst.*, vol. 31, no. 5, pp. 25–39, May 2020, doi: 10.1109/TPDS.2019.2958344.

[131] M. Karakoy, O. Kislal, X. Tang, M. T. Kandemir, and M. Arunachalam, ''Architecture-aware approximate computing,'' *Proc. ACM Meas. Anal. Comput. Syst.*, vol. 3, no. 2, pp. 1–24, Jun. 2019, doi: 10.1145/3341617.3326153.

[132] S. Leroux, P. Molchanov, P. Simoens, B. Dhoedt, T. Breuel, and J. Kautz, ''IamNN: Iterative and adaptive mobile neural network for efficient image classification,'' 2018, *arXiv:1804.10123*.

[133] J. Yang, Y. Bhalgat, S. Chang, F. Porikli, and N. Kwak, ''Dynamic iterative refinement for efficient 3D hand pose estimation,'' 2021, *arXiv:2111.06500*.

[134] Y. Yoo, D. Han, and S. Yun, ''EXTD: Extremely tiny face detector via iterative filter reuse,'' 2019, *arXiv:1906.06579*.

[135] W. Jin, J. Wohlwend, R. Barzilay, and T. Jaakkola, ''Iterative refinement graph neural network for antibody sequence-structure co-design,'' 2021, *arXiv:2110.04624*.

[136] Y. Tian, Y. Zhang, and H. Zhang, ''Recent advances in stochastic gradient descent in deep learning,'' *Mathematics*, vol. 11, no. 3, p. 682, Jan. 2023, doi: 10.3390/math11030682.

[137] C. Shieh, S. Ofner, and C. B. Draucker, ''Reasons for and associated characteristics with early study termination: Analysis of ClinicalTrials.Gov data on pregnancy topics,'' *Nursing Outlook*, vol. 70, no. 2, pp. 271–279, Mar. 2022, doi: 10.1016/j.outlook.2021.12.006.

[138] M. Mahsereci, L. Balles, C. Lassner, and P. Hennig, ''Early stopping without a validation set,'' 2017, *arXiv:1703.09580*.

[139] M. V. Ferro, Y. D. Mosquera, F. J. R. Pena, and V. M. D. Bilbao, ''Early stopping by correlating online indicators in neural networks,'' *Neural Netw.*, vol. 159, pp. 109–124, Feb. 2023, doi: 10.1016/j.neunet.2022.11.035.

[140] Y. Bai, E. Yang, B. Han, Y. Yang, J. Li, Y. Mao, G. Niu, and T. Liu, ''Understanding and improving early stopping for learning with noisy labels,'' 2021, *arXiv:2106.15853*.

[141] Y.-W. Chen, C. Wang, A. Saied, and R. Zhuang, ''ACE: Adaptive constraint-aware early stopping in hyperparameter optimization,'' 2022, *arXiv:2208.02922*.

[142] Y. Matsubara, M. Levorato, and F. Restuccia, ''Split computing and early exiting for deep learning applications: Survey and research challenges,'' *ACM Comput. Surv.*, vol. 55, no. 5, pp. 1–30, Dec. 2022, doi: 10.1145/3527155.

[143] S. Paguada, L. Batina, I. Buhan, and I. Armendariz, ''Being patient and persistent: Optimizing an early stopping strategy for deep learning in profiled attacks,'' 2021, *arXiv:2111.14416*.

[144] E. Cetinic, T. Lipic, and S. Grgic, ''Fine-tuning convolutional neural networks for fine art classification,'' *Expert Syst. Appl.*, vol. 114, pp. 107–118, Dec. 2018, doi: 10.1016/j.eswa.2018.07.026.

[145] E. Lattanzi, C. Contoli, and V. Freschi, ''Do we need early exit networks in human activity recognition?'' *Eng. Appl. Artif. Intell.*, vol. 121, May 2023, Art. no. 106035, doi: 10.1016/j.engappai.2023.106035.

[146] C. Yang and X. Ma, ''Improving stability of fine-tuning pretrained language models via component-wise gradient norm clipping,'' 2022, *arXiv:2210.10325*.

[147] F. Liu, X. Huang, Y. Chen, and J. A. K. Suykens, ''Random features for kernel approximation: A survey on algorithms, theory, and beyond,'' 2020, *arXiv:2004.11154*.

[148] A. De Marchi, A. Dreves, M. Gerdts, S. Gottschalk, and S. Rogovs, ''A function approximation approach for parametric optimization,'' *J. Optim. Theory Appl.*, vol. 196, no. 1, pp. 56–77, Jan. 2023, doi: 10.1007/s10957-022-02138-4.

[149] D. Dũng and V. K. Nguyen, ''Deep ReLU neural networks in high-dimensional approximation,'' *Neural Netw.*, vol. 142, pp. 619–635, Oct. 2021, doi: 10.1016/j.neunet.2021.07.027.

[150] Z. Zainuddin and O. Pauline, ''Function approximation using artificial neural networks,'' *WSEAS Trans. Math.*, vol. 7, no. 6, pp. 333–338, Jun. 2008.

[151] T. De Ryck, S. Lanthaler, and S. Mishra, ''On the approximation of functions by tanh neural networks,'' *Neural Netw.*, vol. 143, pp. 732–750, Nov. 2021, doi: 10.1016/j.neunet.2021.08.015.

[152] S. S. Sawant, M. Wiedmann, S. Göb, N. Holzer, E. W. Lang, and T. Götz, ''Compression of deep convolutional neural network using additional importance-weight-based filter pruning approach,'' *Appl. Sci.*, vol. 12, no. 21, p. 11184, Nov. 2022, doi: 10.3390/app122111184.

[153] C.-T. Huang, J.-C. Chen, and J.-L. Wu, ''Learning sparse neural networks through mixture-distributed regularization,'' in *Proc. IEEE/CVF Conf. Comput. Vis. Pattern Recognit. Workshops (CVPRW)*, Jun. 2020, pp. 2968–2977, doi: 10.1109/CVPRW50498.2020.00355.

[154] C. Louizos, M. Welling, and D. P. Kingma, "Learning sparse neural networks through L_0 regularization," 2017, *arXiv:1712.01312*.

[155] J. Luo, Y. Gan, C.-M. Vong, C.-M. Wong, and C. Chen, "Scalable and memory-efficient sparse learning for classification with approximate Bayesian regularization priors," *Neurocomputing*, vol. 457, pp. 106–116, Oct. 2021, doi: 10.1016/j.neucom.2021.06.025.

[156] Z. Izzo, M. A. Smart, K. Chaudhuri, and J. Zou, "Approximate data deletion from machine learning models," in *Proc. 24th Int. Conf. Artif. Intell. Statist.*, 2011, pp. 2008–2016. Accessed: Jun. 30, 2023.

[157] S. Park, J. Lee, S. Mo, and J. Shin, "Lookahead: A far-sighted alternative of magnitude-based pruning," 2020, *arXiv:2002.04809*.

[158] N. Lee, T. Ajanthan, and P. H. S. Torr, "SNIP: Single-shot network pruning based on connection sensitivity," 2018, *arXiv:1810.02340*.

[159] X. Xiao, Z. Wang, and S. Rajasekaran, "AutoPrune: Automatic network pruning by regularizing auxiliary parameters," in *Proc. Adv. Neural Inf. Process. Syst.*, Red Hook, NY, USA: Curran Associates, Inc., 2019, pp. 1–17. Accessed: Mar. 19, 2023. [Online]. Available: https://proceedings.neurips.cc/paper/2019/hash/4efc9e02abdab6b61 66251918570a307-Abstract.html

[160] Z. Huang and N. Wang, "Data-driven sparse structure selection for deep neural networks," 2017, *arXiv:1707.01213*.

[161] S. Yu, Z. Yao, A. Gholami, Z. Dong, S. Kim, M. W. Mahoney, and K. Keutzer, "Hessian-aware pruning and optimal neural implant," 2021, *arXiv:2101.08940*.

[162] Y. Fu, C. Liu, D. Li, Z. Zhong, X. Sun, J. Zeng, and Y. Yao, "Exploring structural sparsity of deep networks via inverse scale spaces," *IEEE Trans. Pattern Anal. Mach. Intell.*, vol. 45, no. 2, pp. 1749–1765, Feb. 2023, doi: 10.1109/TPAMI.2022.3168881.

[163] N. J. Kim and H. Kim, "AGT: Channel pruning using adaptive gradient training for accelerating convolutional neural networks," in *Proc. Int. Conf. Electron., Inf., Commun. (ICEIC)*, Feb. 2023, pp. 1–3, doi: 10.1109/ICEIC57457.2023.10049943.

[164] S. Gao, P. Dong, Z. Pan, and X. You, "Lightweight deep learning based channel estimation for extremely large-scale massive MIMO systems," *IEEE Trans. Veh. Technol.*, vol. 73, no. 7, pp. 10750–10754, Jul. 2024, doi: 10.1109/TVT.2024.3364510.

[165] Y. He, P. Liu, Z. Wang, Z. Hu, and Y. Yang, "Filter pruning via geometric median for deep convolutional neural networks acceleration," in *Proc. IEEE/CVF Conf. Comput. Vis. Pattern Recognit. (CVPR)*, Jun. 2019, doi: 10.1109/CVPR.2019.00447.

[166] C. Zhao, B. Ni, J. Zhang, Q. Zhao, W. Zhang, and Q. Tian, "Variational convolutional neural network pruning," in *Proc. IEEE/CVF Conf. Comput. Vis. Pattern Recognit. (CVPR)*, Jun. 2019, pp. 2775–2784, doi: 10.1109/CVPR.2019.00289.

[167] M. Sabih, F. Hannig, and J. Teich, "DyFiP: Explainable AI-based dynamic filter pruning of convolutional neural networks," in *Proc. 2nd Eur. Workshop Mach. Learn. Syst.* New York, NY, USA: ACM, Apr. 2022, pp. 109–115, doi: 10.1145/3517207.3526982.

[168] J.-H. Luo, J. Wu, and W. Lin, "ThiNet: A filter level pruning method for deep neural network compression," 2017, *arXiv:1707.06342*.

[169] K.-L. Du, M. N. S. Swamy, Z.-Q. Wang, and W. H. Mow, "Matrix factorization techniques in machine learning, signal processing, and statistics," *Mathematics*, vol. 11, no. 12, p. 2674, Jun. 2023, doi: 10.3390/math11122674.

[170] M. Sabih, A. Mishra, F. Hannig, and J. Teich, "MOSP: Multi-objective sensitivity pruning of deep neural networks," in *Proc. IEEE 13th Int. Green Sustain. Comput. Conf. (IGSC)*, Oct. 2022, pp. 1–8, doi: 10.1109/IGSC55832.2022.9969374.

[171] T. Hoefler, D. Alistarh, T. Ben-Nun, N. Dryden, and A. Peste, "Sparsity in deep learning: Pruning and growth for efficient inference and training in neural networks," *J. Mach. Learn. Res.*, vol. 22, no. 1, pp. 241:10882–241:11005, Jan. 2021.

[172] J. Choquette and W. Gandhi, "NVIDIA A100 GPU: Performance & innovation for GPU computing," in *Proc. IEEE Hot Chips 32 Symp. (HCS)*, Aug. 2020, pp. 1–43, doi: 10.1109/HCS49909.2020.9220622.

[173] L. Lu, J. Xie, R. Huang, J. Zhang, W. Lin, and Y. Liang, "An efficient hardware accelerator for sparse convolutional neural networks on FPGAs," in *Proc. IEEE 27th Annu. Int. Symp. Field-Program. Custom Comput. Mach. (FCCM)*. San Diego, CA, USA: IEEE, Apr. 2019, pp. 17–25, doi: 10.1109/FCCM.2019.00013.

[174] A. Tragoudaras, P. Stoikos, K. Fanaras, A. Tziouvaras, G. Floros, G. Dimitriou, K. Kolomvatsos, and G. Stamoulis, "Design space exploration of a sparse MobileNetV2 using high-level synthesis and sparse matrix techniques on FPGAs," *Sensors*, vol. 22, no. 12, p. 4318, Jun. 2022, doi: 10.3390/s22124318.

[175] P. Pinto and J. M. P. Cardoso, "A methodology and framework for software memoization of functions," in *Proc. 18th ACM Int. Conf. Comput. Frontiers*, vol. 13. New York, NY, USA: ACM, May 2021, pp. 93–101, doi: 10.1145/3457388.3458668.

[176] I. Brumar, M. Casas, M. Moreto, M. Valero, and G. S. Sohi, "ATM: Approximate task memoization in the runtime system," in *Proc. IEEE Int. Parallel Distrib. Process. Symp. (IPDPS)*, May 2017, pp. 1140–1150, doi: 10.1109/IPDPS.2017.49.

[177] A. Suresh, E. Rohou, and A. Seznec, "Compile-time function memoization," in *Proc. 26th Int. Conf. Compiler Construct.* New York, NY, USA: ACM, Feb. 2017, pp. 45–54, doi: 10.1145/3033019.3033024.

[178] G. Zhang and D. Sanchez, "Leveraging hardware caches for memoization," *IEEE Comput. Archit. Lett.*, vol. 17, no. 1, pp. 59–63, Jan. 2018, doi: 10.1109/LCA.2017.2762308.

[179] G. Tziantzioulis, N. Hardavellas, and S. Campanoni, "Temporal approximate function memoization," *IEEE Micro*, vol. 38, no. 4, pp. 60–70, Jul. 2018, doi: 10.1109/MM.2018.043191126.

[180] P. Arundhati, S. K. Jena, and S. K. Pani, "Approximate function memoization," *Concurrency Comput., Pract. Exper.*, vol. 34, no. 23, p. e7204, Oct. 2022, doi: 10.1002/cpe.7204.

[181] Z. Liu, A. Yazdanbakhsh, D. K. Wang, H. Esmaeilzadeh, and N. S. Kim, "AxMemo: Hardware-compiler co-design for approximate code memoization," in *Proc. ACM/IEEE 46th Annu. Int. Symp. Comput. Archit. (ISCA)*. New York, NY, USA: ACM, Jun. 2019, pp. 685–697.

[182] S. Bubeck, R. Eldan, Y. T. Lee, and D. Mikulincer, "Network size and weights size for memorization with two-layers neural networks," in *Proc. 34th Int. Conf. Neural Inf. Process. Syst.* Red Hook, NY, USA: Curran Associates Inc., 2020, pp. 4977–4986.

[183] A. M. Kassem, "Mitigating approximate memorization in language models via dissimilarity learned policy," 2023, *arXiv:2305.01550*.

[184] G. Kyriakides and K. Margaritis, "An introduction to neural architecture search for convolutional networks," 2020, *arXiv:2005.11074*.

[185] J. Fabrício Filho, I. Felzmann, and L. Wanner, "SmartApprox: Learning-based configuration of approximate memories for energy-efficient execution," *Sustain. Comput., Informat. Syst.*, vol. 34, Apr. 2022, Art. no. 100701, doi: 10.1016/j.suscom.2022.100701.

[186] T. Huang, W. Dong, F. Wu, X. Li, and G. Shi, "Uncertainty-driven knowledge distillation for language model compression," *IEEE/ACM Trans. Audio, Speech, Language Process.*, vol. 31, pp. 2850–2858, Jun. 2023, doi: 10.1109/TASLP.2023.3289303.

[187] X. Liu, Z. Shi, Z. Wu, J. Chen, and G. Zhai, "GridDehazeNet+: An enhanced multi-scale network with intra-task knowledge transfer for single image dehazing," *IEEE Trans. Intell. Transp. Syst.*, vol. 24, no. 1, pp. 870–884, Jan. 2023, doi: 10.1109/TITS.2022.3210455.

[188] W. Tang, M. S. Shakeel, Z. Chen, H. Wan, and W. Kang, "Target-category agnostic knowledge distillation with frequency domain supervision," *IEEE Trans. Ind. Informat.*, vol. 19, no. 7, pp. 8462–8471, Jul. 2023, doi: 10.1109/TII.2022.3218635.

[189] C.-S. Lin and Y. F. Wang, "Describe, spot and explain: Interpretable representation learning for discriminative visual reasoning," *IEEE Trans. Image Process.*, vol. 32, pp. 2481–2492, 2023, doi: 10.1109/TIP.2023.3268001.

[190] Y. Zhao and N.-M. Cheung, "FS-BAN: Born-again networks for domain generalization few-shot classification," *IEEE Trans. Image Process.*, vol. 32, pp. 2252–2266, 2023, doi: 10.1109/TIP.2023.3266172.

[191] J. Li, X. Chen, P. Zheng, Q. Wang, and Z. Yu, "Deep generative knowledge distillation by likelihood finetuning," *IEEE Access*, vol. 11, pp. 46441–46453, 2023, doi: 10.1109/ACCESS.2023.3273952.

[192] J. Chen, X. Qu, J. Li, J. Wang, J. Wan, and J. Xiao, "Detecting out-of-distribution examples via class-conditional impressions reappearing," in *Proc. IEEE Int. Conf. Acoust., Speech Signal Process. (ICASSP)*, Jun. 2023, pp. 1–5, doi: 10.1109/ICASSP49357.2023.10095909.

[193] Y. Wang, Z. Ge, Z. Chen, X. Liu, C. Ma, Y. Sun, and L. Qi, "Explicit and implicit knowledge distillation via unlabeled data," in *Proc. IEEE Int. Conf. Acoust., Speech Signal Process. (ICASSP)*, Jun. 2023, pp. 1–5, doi: 10.1109/ICASSP49357.2023.10095175.

[194] L. Yu, T. Hua, W. Yang, P. Ye, and Q. Liao, "CDHD: Contrastive dreamer for hint distillation," in *Proc. ICASSP - IEEE Int. Conf. Acoust., Speech Signal Process. (ICASSP)*, Jun. 2023, pp. 1–5, doi: 10.1109/ICASSP49357.2023.10096829.

[195] H. Yin, P. Molchanov, J. M. Alvarez, Z. Li, A. Mallya, D. Hoiem, N. K. Jha, and J. Kautz, "Dreaming to distill: Data-free knowledge transfer via DeepInversion," in *Proc. IEEE/CVF Conf. Comput. Vis. Pattern Recognit. (CVPR)*, Jun. 2020, pp. 8712–8721, doi: 10.1109/CVPR42600.2020.00874.

[196] Z. Altuntaş, S. Arslan, and B. Boz, "Approximate execution and grouping of critical sections for performance-accuracy tradeoff," *Concurrency Comput., Pract. Exper.*, vol. 35, no. 24, p. e7614, Nov. 2023, doi: 10.1002/cpe.7614.

[197] S. K. Khatamifard, I. Akturk, and U. R. Karpuzcu, "On approximate speculative lock elision," *IEEE Trans. Multi-Scale Comput. Syst.*, vol. 4, no. 2, pp. 141–151, Apr. 2018, doi: 10.1109/TMSCS.2017.2773488.

[198] L. Carpentieri and B. Cosenza, "Towards a SYCL API for approximate computing," in *Proc. Int. Workshop OpenCL*, vol. 48. New York, NY, USA: ACM, Apr. 2023, pp. 1–2, doi: 10.1145/3585341.3585374.

[199] K. Lee, R. Bhattacharya, J. Dass, V. N. S. P. Sakuru, and R. N. Mahapatra, "A relaxed synchronization approach for solving parallel quadratic programming problems with guaranteed convergence," in *Proc. IEEE Int. Parallel Distrib. Process. Symp. (IPDPS)*, May 2016, pp. 182–191, doi: 10.1109/IPDPS.2016.66.

[200] G. Stitt and D. Campbell, "PANDORA: An architecture-independent parallelizing approximation-discovery framework," *ACM Trans. Embedded Comput. Syst.*, vol. 19, no. 5, pp. 1–17, Nov. 2020, doi: 10.1145/3391899.

[201] A. L. C. Bueno, N. D. L. R. Rodriguez, and E. D. Sotelino, "Adaptive relaxed synchronization through the use of supervised learning methods," *Future Gener. Comput. Syst.*, vol. 106, pp. 260–269, May 2020, doi: 10.1016/j.future.2019.12.051.

[202] A. Sampson, A. Baixo, B. Ransford, T. Moreau, J. Yip, L. Ceze, and M. Oskin, "ACCEPT: A programmer-guided compiler framework for practical approximate computing," Dept. Comput. Sci. Eng., Univ. Washington, Washington, DC, USA, Tech. Rep. UW-CSE-15-01, 2015.

[203] J. Ansel, Y. L. Wong, C. Chan, M. Olszewski, A. Edelman, and S. Amarasinghe, "Language and compiler support for auto-tuning variable-accuracy algorithms," in *Proc. Int. Symp. Code Gener. Optim. (CGO)*, Apr. 2011, pp. 85–96, doi: 10.1109/CGO.2011.5764677.

[204] M. Carbin, S. Misailovic, and M. C. Rinard, "Verifying quantitative reliability for programs that execute on unreliable hardware," *Commun. ACM*, vol. 59, no. 8, pp. 83–91, Jul. 2016, doi: 10.1145/2958738.

[205] A. Sampson, W. Dietl, E. Fortuna, D. Gnanapragasam, L. Ceze, and D. Grossman, "EnerJ: Approximate data types for safe and general low-power computation," in *Proc. 32nd ACM SIGPLAN Conf. Program. Lang. Design Implement.* New York, NY, USA: ACM, Jun. 2011, pp. 164–174, doi: 10.1145/1993498.1993518.

[206] J. Park, H. Esmaeilzadeh, X. Zhang, M. Naik, and W. Harris, "FlexJava: Language support for safe and modular approximate programming," in *Proc. 10th Joint Meeting Found. Softw. Eng.*, vol. 2008. New York, NY, USA: ACM, Aug. 2015, pp. 745–757, doi: 10.1145/2786805.2786807.

[207] M. Nguyen, R. Perera, M. Wang, and N. Wu, "Modular probabilistic models via algebraic effects," *Proc. ACM Program. Lang.*, vol. 6, pp. 381–410, Aug. 2022, doi: 10.1145/3547635.

[208] A. McCallum, K. Schultz, and S. Singh, "FACTORIE: Probabilistic programming via imperatively defined factor graphs," in *Proc. Advances in Neural Information Processing Systems*. Red Hook, NY, USA: Curran Associates, 2009, pp. 1–12. Accessed: Jul. 2, 2023. [Online]. Available: https://proceedings.neurips.cc/paper_files/paper/2009/hash/847cc55b7032108eee6dd897f3bca8a5-Abstract.html

[209] V. Mansinghka, D. Selsam, and Y. Perov, "Venture: A higher-order probabilistic programming platform with programmable inference," 2014, *arXiv:1404.0099*.

[210] A. Todeschini, F. Caron, M. Fuentes, P. Legrand, and P. Del Moral, "Biips: Software for Bayesian inference with interacting particle systems," 2014, *arXiv:1412.3779*.

[211] B. Carpenter, A. Gelman, M. D. Hoffman, D. Lee, B. Goodrich, M. Betancourt, M. Brubaker, J. Guo, P. Li, and A. Riddell, "Stan: A probabilistic programming language," *J. Stat. Softw.*, vol. 76, no. 1, p. 1, 2017, doi: 10.18637/jss.v076.i01.

[212] M. I. Gorinova, A. D. Gordon, and C. Sutton, "Probabilistic programming with densities in SlicStan: Efficient, flexible, and deterministic," *Proc. ACM Program. Lang.*, vol. 3, pp. 1–30, Jan. 2019, doi: 10.1145/3290348.

[213] M. Cusumano-Towner and V. K. Mansinghka, "A design proposal for gen: Probabilistic programming with fast custom inference via code generation," in *Proc. 2nd ACM SIGPLAN Int. Workshop Mach. Learn. Program. Lang.*, vol. 16. New York, NY, USA: ACM, Jun. 2018, pp. 52–57, doi: 10.1145/3211346.3211350.

[214] O. Kiselyov, "Probabilistic programming language and its incremental evaluation," in *Programming Languages and Systems* (Lecture Notes in Computer Science), A. Igarashi, Ed., Cham, Switzerland: Springer, 2016, pp. 357–376, doi: 10.1007/978-3-319-47958-3_19.

[215] J. Ai, N. S. Arora, N. Dong, B. Gokkaya, T. Jiang, A. Kubendran, A. Kumar, M. Tingley, and N. Torabi, "HackPPL: A universal probabilistic programming language," in *Proc. 3rd ACM SIGPLAN Int. Workshop Mach. Learn. Program. Lang.*, vol. 43. New York, NY, USA: ACM, Jun. 2019, pp. 20–28, doi: 10.1145/3315508.3329974.

[216] D. Tolpin, J.-W. van de Meent, H. Yang, and F. Wood, "Design and implementation of probabilistic programming language anglican," in *Proc. 28th Symp. Implement. Appl. Funct. Program. Lang.*, vol. 48. New York, NY, USA: ACM, Aug. 2016, pp. 1–12, doi: 10.1145/3064899.3064910.

[217] D. Tolpin, "Deployable probabilistic programming," in *Proc. ACM SIGPLAN Int. Symp. New Ideas, New Paradigms, Reflections Program. Softw.*, vol. 18. New York, NY, USA: ACM, Oct. 2019, pp. 1–16, doi: 10.1145/3359591.3359727.

[218] K. Joshi, V. Fernando, and S. Misailovic, "Aloe: Verifying reliability of approximate programs in the presence of recovery mechanisms," in *Proc. 18th ACM/IEEE Int. Symp. Code Gener. Optim.* New York, NY, USA: ACM, Feb. 2020, pp. 56–67, doi: 10.1145/3368826.3377924.

[219] E. Bingham, J. P. Chen, M. Jankowiak, F. Obermeyer, N. Pradhan, T. Karaletsos, R. Singh, P. Szerlip, P. Horsfall, and N. D. Goodman, "Pyro: Deep universal probabilistic programming," 2018, *arXiv:1810.09538*.

[220] A. Ścibior, O. Kammar, M. Vákár, S. Staton, H. Yang, Y. Cai, K. Ostermann, S. K. Moss, C. Heunen, and Z. Ghahramani, "Denotational validation of higher-order Bayesian inference," *Proc. ACM Program. Lang.*, vol. 2, pp. 1–29, Dec. 2017, doi: 10.1145/3158148.

[221] A. K. Lew, M. F. Cusumano-Towner, B. Sherman, M. Carbin, and V. K. Mansinghka, "Trace types and denotational semantics for sound programmable inference in probabilistic languages," *Proc. ACM Program. Lang.*, vol. 4, pp. 1–32, Dec. 2019, doi: 10.1145/3371087.

[222] S. Dylus, J. Christiansen, and F. Teegen, "Probabilistic functional logic programming," in *Practical Aspects of Declarative Languages* (Lecture Notes in Computer Science), K. Hamlen, and N. Leone, Eds., Cham, Switzerland: Springer, 2018, pp. 3–19, doi: 10.1007/978-3-319-73305-0_1.

[223] *FACTORIE: Home*. Accessed: Jul. 2, 2023. [Online]. Available: http://factorie.cs.umass.edu/

[224] M. Biel and M. Johansson, "Efficient stochastic programming in Julia," *INFORMS J. Comput.*, vol. 34, no. 4, pp. 1885–1902, Jul. 2022, doi: 10.1287/ijoc.2022.1158.

[225] A. Ścibior, O. Kammar, and Z. Ghahramani, "Functional programming for modular Bayesian inference," *Proc. ACM Program. Lang.*, vol. 2, pp. 1–29, Jul. 2018, doi: 10.1145/3236778.

[226] A. Ścibior, Z. Ghahramani, and A. D. Gordon, "Practical probabilistic programming with monads," in *Proc. ACM SIGPLAN Symp. Haskell*. New York, NY, USA: ACM, Aug. 2015, pp. 165–176, doi: 10.1145/2804302.2804317.

[227] C. Grazian and Y. Fan, "A review of approximate Bayesian computation methods via density estimation: Inference for simulator-models," *WIREs Comput. Statist.*, vol. 12, no. 4, p. e1486, Jul. 2020, doi: 10.1002/wics.1486.

[228] M. Barbareschi, S. Barone, N. Mazzocca, and A. Moriconi, "Design space exploration tools," in *Approximate Computing Techniques: From Component- to Application-Level*. Cham, Switzerland: Springer, 2022, pp. 215–259, doi: 10.1007/978-3-030-94705-7_8.

[229] S. Misailovic, "Accuracy-aware compilers," in *Approximate Computing Techniques: From Component- to Application-Level*.Cham, Switzerland: Springer, 2022, pp. 177–214, doi: 10.1007/978-3-030-94705-7_7.

[230] D. Gadioli, E. Vitali, G. Palermo, and C. Silvano, ''MARGOt: A dynamic autotuning framework for self-aware approximate computing,'' *IEEE Trans. Comput.*, vol. 68, no. 5, pp. 713–728, May 2019, doi: 10.1109/TC.2018.2883597.

[231] W.-C. Lee, Y. Liu, P. Liu, S. Ma, H. Choi, X. Zhang, and R. Gupta, ''White-box program tuning,'' in *Proc. IEEE/ACM Int. Symp. Code Gener. Optim. (CGO)*, Feb. 2019, pp. 122–135, doi: 10.1109/CGO.2019.8661177.

[232] T. T. Jost, Y. Durand, C. Fabre, A. Cohen, and F. Pérrot, ''Seamless compiler integration of variable precision floating-point arithmetic,'' in *Proc. IEEE/ACM Int. Symp. Code Gener. Optim. (CGO)*, Feb. 2021, pp. 65–76, doi: 10.1109/CGO51591.2021.9370331.

[233] H. Sharif, Y. Zhao, M. Kotsifakou, A. Kothari, B. Schreiber, E. Wang, Y. Sarita, N. Zhao, K. Joshi, V. S. Adve, S. Misailovic, and S. Adve, ''ApproxTuner: A compiler and runtime system for adaptive approximations,'' in *Proc. 26th ACM SIGPLAN Symp. Princ. Pract. Parallel Program.*, vol. 216. New York, NY, USA: ACM, Feb. 2021, pp. 262–277, doi: 10.1145/3437801.3446108.

[234] L. Liu, S. Isaacman, and U. Kremer, ''An adaptive application framework with customizable quality metrics,'' *ACM Trans. Design Autom. Electron. Syst.*, vol. 27, no. 2, pp. 1–33, Nov. 2021, doi: 10.1145/3477428.

[235] R. Venkatagiri, K. Ahmed, A. Mahmoud, S. Misailovic, D. Marinov, C. W. Fletcher, and S. V. Adve, ''Gem5-approxilyzer: An open-source tool for application-level soft error analysis,'' in *Proc. 49th Annu. IEEE/IFIP Int. Conf. Dependable Syst. Netw. (DSN)*, Jun. 2019, pp. 214–221, doi: 10.1109/DSN.2019.00033.

[236] D. Danopoulos, G. Zervakis, K. Siozios, D. Soudris, and J. Henkel, ''AdaPT: Fast emulation of approximate DNN accelerators in PyTorch,'' *IEEE Trans. Comput.-Aided Design Integr. Circuits Syst.*, vol. 42, no. 6, pp. 2074–2078, Jun. 2023, doi: 10.1109/TCAD.2022.3212645.

[237] M. Schramm, S. Bhowmik, and K. Rothermel, ''Flexible application-aware approximation for modern distributed graph processing frameworks,'' in *Proc. 5th ACM SIGMOD Joint Int. Workshop Graph Data Manage. Experiences Syst. (GRADES) Netw. Data Analytics (NDA)*, vol. 12. New York, NY, USA: ACM, Jun. 2022, pp. 1–10, doi: 10.1145/3534540.3534693.

[238] S. De, S. Mohamed, D. Goswami, and H. Corporaal, ''Approximation-aware design of an image-based control system,'' *IEEE Access*, vol. 8, pp. 174568–174586, 2020, doi: 10.1109/ACCESS.2020.3023047.

[239] M. A. Johnston and V. Vassiliadis, ''Towards an approximation-aware computational workflow framework for accelerating large-scale discovery tasks: Invited paper,'' in *Proc. Workshop Adv. Tools, Program. Lang., PLatforms Implementing Evaluating Algorithms Distrib. Syst.*, vol. 11. New York, NY, USA: ACM, Jul. 2022, pp. 7–14, doi: 10.1145/3524053.3542746.

[240] M. A. Hanif, R. Hafiz, and M. Shafique, ''Error resilience analysis for systematically employing approximate computing in convolutional neural networks,'' in *Proc. Design, Autom. Test Eur. Conf. Exhib. (DATE)*, Mar. 2018, pp. 913–916, doi: 10.23919/DATE.2018.8342139.

[241] K. Parasyris, J. Diffenderfer, H. Menon, I. Laguna, J. Vanover, R. Vogt, and D. Osei-Kuffuor, ''Approximate computing through the lens of uncertainty quantification,'' in *Proc. Int. Conf. for High Perform. Comput., Netw., Storage Anal.*, Nov. 2022, pp. 1–14, doi: 10.1109/SC41404.2022.00072.

[242] A. Bernstein, A. Dudeja, and Z. Langley, ''A framework for dynamic matching in weighted graphs,'' in *Proc. 53rd Annu. ACM SIGACT Symp. Theory Comput.*, Jun. 2021, pp. 668–681, doi: 10.1145/3406325.3451113.

[243] X. Fang, N. Han, G. Zhou, S. Teng, Y. Xu, and S. Xie, ''Dynamic double classifiers approximation for cross-domain recognition,'' *IEEE Trans. Cybern.*, vol. 52, no. 4, pp. 2618–2629, Apr. 2022, doi: 10.1109/TCYB.2020.3004398.

[244] M. Gao and G. Qu, ''Estimate and recompute: A novel paradigm for approximate computing on data flow graphs,'' *IEEE Trans. Comput.-Aided Design Integr. Circuits Syst.*, vol. 39, no. 2, pp. 335–345, Feb. 2020, doi: 10.1109/TCAD.2018.2889662.

[245] Y. Wang, J. Dong, Y. Liu, C. Wang, and G. Qu, ''RMLIM: A runtime machine learning based identification model for approximate computing on data flow graphs,'' *IEEE Trans. Sustain. Comput.*, vol. 7, no. 1, pp. 201–210, Jan. 2022, doi: 10.1109/TSUSC.2021.3074292.

[246] M. Soni, A. Pal, and J. S. Miguel, ''As-is approximate computing,'' *ACM Trans. Archit. Code Optim.*, vol. 20, no. 1, pp. 1–26, Nov. 2022, doi: 10.1145/3559761.

[247] B. Nongpoh, R. Ray, S. Dutta, and A. Banerjee, ''AutoSense: A framework for automated sensitivity analysis of program data,'' *IEEE Trans. Softw. Eng.*, vol. 43, no. 12, pp. 1110–1124, Dec. 2017, doi: 10.1109/TSE.2017.2654251.

[248] P. Roy, R. Ray, C. Wang, and W. F. Wong, ''ASAC: Automatic sensitivity analysis for approximate computing,'' *ACM SIGPLAN Notices*, vol. 49, no. 5, pp. 95–104, Jun. 2014, doi: 10.1145/2666357.2597812.

[249] K. Joshi, V. Fernando, and S. Misailovic, ''Statistical algorithmic profiling for randomized approximate programs,'' in *Proc. IEEE/ACM 41st Int. Conf. Softw. Eng. (ICSE)*. Piscataway, NJ, USA: IEEE Press, May 2019, pp. 608–618, doi: 10.1109/ICSE.2019.00071.

[250] C. Gonzalez, H. Liu, M. Noh, E. Karl, T. Toifl, and S. Hsu, ''f5: Enabling new system architectures with 2.5D, 3D, and chiplets,'' in *IEEE Int. Solid-State Circuits Conf. (ISSCC) Dig. Tech. Papers*, vol. 64, Feb. 2021, pp. 529–532, doi: 10.1109/ISSCC42613.2021.9365834.

[251] A. Zeitak and A. Morrison, ''Cuckoo trie: Exploiting memory-level parallelism for efficient DRAM indexing,'' in *Proc. ACM SIGOPS 28th Symp. Operating Syst. Princ.* New York, NY, USA: ACM, Oct. 2021, pp. 147–162, doi: 10.1145/3477132.3483551.

[252] R. Kumar, M. Alipour, and D. Black-Schaffer, ''Freeway: Maximizing MLP for slice-out-of-order execution,'' in *Proc. IEEE Int. Symp. High Perform. Comput. Archit. (HPCA)*, Feb. 2019, pp. 558–569, doi: 10.1109/HPCA.2019.00009.

[253] K. Dimple, S. Guglani, A. Dasgupta, R. Sharma, S. Roy, and B. K. Kaushik, ''Modified knowledge-based neural networks using control variates for the fast uncertainty quantification of on-chip MWCNT interconnects,'' *IEEE Trans. Electromagn. Compat.*, vol. 65, no. 4, pp. 1232–1246, Aug. 2023, doi: 10.1109/TEMC.2023.3279695.

[254] H. Liu, L. Zheng, Y. Huang, C. Liu, X. Ye, J. Yuan, X. Liao, H. Jin, and J. Xue, ''Accelerating personalized recommendation with cross-level near-memory processing,'' in *Proc. 50th Annu. Int. Symp. Comput. Archit.*, vol. 42. New York, NY, USA: ACM, Jun. 2023, pp. 1–13, doi: 10.1145/3579371.3589101.

[255] J. Song, X. Tang, H. Luo, H. Zhang, X. Qiao, Z. Sun, X. Yang, Y. Wang, R. Wang, and R. Huang, ''A calibration-free 15-level/cell eDRAM computing-in-memory macro with 3T1C current-programmed dynamic-cascoded MLC achieving 233-to-304-TOPS/W 4b MAC,'' in *Proc. IEEE Custom Integr. Circuits Conf. (CICC)*, Apr. 2023, pp. 1–2, doi: 10.1109/CICC57935.2023.10121207.

[256] A. Ranjan, A. Raha, V. Raghunathan, and A. Raghunathan, ''Approximate memory compression,'' *IEEE Trans. Very Large Scale Integr. (VLSI) Syst.*, vol. 28, no. 4, pp. 980–991, Apr. 2020, doi: 10.1109/TVLSI.2020.2970041.

[257] G. Singh, L. Chelini, S. Corda, A. J. Awan, S. Stuijk, R. Jordans, H. Corporaal, and A.-J. Boonstra, ''Near-memory computing: Past, present, and future,'' *Microprocessors Microsyst.*, vol. 71, Nov. 2019, Art. no. 102868, doi: 10.1016/j.micpro.2019.102868.

[258] B. W. Denkinger, F. Ponzina, S. S. Basu, A. Bonetti, S. Balási, M. Ruggiero, M. Peón-Quirós, D. Rossi, A. Burg, and D. Atienza, ''Impact of memory voltage scaling on accuracy and resilience of deep learning based edge devices,'' *IEEE Des. Test. IEEE Design Test. Comput.*, vol. 37, no. 2, pp. 84–92, Apr. 2020, doi: 10.1109/MDAT.2019.2947282.

[259] A. Raha, S. Sutar, H. Jayakumar, and V. Raghunathan, ''Quality configurable approximate DRAM,'' *IEEE Trans. Comput.*, vol. 66, no. 7, pp. 1172–1187, Jul. 2017, doi: 10.1109/TC.2016.2640296.

[260] A. Ranjan, A. Raha, V. Raghunathan, and A. Raghunathan, ''Approximate memory compression for energy-efficiency,'' in *Proc. IEEE/ACM Int. Symp. Low Power Electron. Design (ISLPED)*, Jul. 2017, pp. 1–6, doi: 10.1109/ISLPED.2017.8009173.

[261] R. V. W. Putra, M. A. Hanif, and M. Shafique, ''An off-chip memory access optimization for embedded deep learning systems,'' in *Embedded Machine Learning for Cyber-Physical, IoT, and Edge Computing: Hardware Architectures*, S. Pasricha and M. Shafique, Eds., Cham, Switzerland: Springer, 2024, pp. 175–198, doi: 10.1007/978-3-031-19568-6_6.

[262] R. V. W. Putra, M. A. Hanif, and M. Shafique, ''EnforceSNN: Enabling resilient and energy-efficient spiking neural network inference considering approximate DRAMs for embedded systems,'' *Frontiers Neurosci.*, vol. 16, Aug. 2022, Art. no. 937782.

[263] L. Orosa, Y. Wang, I. Puddu, M. Sadrosadati, K. Razavi, J. Gómez-Luna, H. Hassan, N. Mansouri-Ghiasi, A. Tavakkol, M. Patel, J. Kim, V. Seshadri, U. Kang, S. Ghose, R. Azevedo, and O. Mutlu, "Dataplant: Enhancing system security with low-cost in-DRAM value generation primitives," 2019, *arXiv:1902.07344*.

[264] R. V. Wicaksana Putra, M. Abdullah Hanif, and M. Shafique, "DRMap: A generic DRAM data mapping policy for energy-efficient processing of convolutional neural networks," in *Proc. 57th ACM/IEEE Design Autom. Conf. (DAC)*, Jul. 2020, pp. 1–6, doi: 10.1109/DAC18072.2020.9218672.

[265] G. Stazi, A. Mastrandrea, M. Olivieri, and F. Menichelli, "Quality aware selective ECC for approximate DRAM," in *Applications in Electronics Pervading Industry, Environment and Society* (Lecture Notes in Electrical Engineering). Cham, Switzerland: Springer, 2020, pp. 109–116, doi: 10.1007/978-3-030-37277-4_13.

[266] N. Gupta, A. P. Shah, S. Khan, S. K. Vishvakarma, M. Waltl, and P. Girard, "Error-tolerant reconfigurable VDD 10T SRAM architecture for IoT applications," *Electronics*, vol. 10, no. 14, Jan. 2021, Art. no. 14, doi: 10.3390/electronics10141718.

[267] E. Russo, M. Palesi, D. Patti, H. Lahdhiri, S. Monteleone, G. Ascia, and V. Catania, "Combined application of approximate computing techniques in DNN hardware accelerators," in *Proc. IEEE Int. Parallel Distrib. Process. Symp. Workshops (IPDPSW)*, May 2022, pp. 16–23, doi: 10.1109/IPDPSW55747.2022.00013.

[268] D. T. Nguyen, N. H. Hung, H. Kim, and H.-J. Lee, "An approximate memory architecture for energy saving in deep learning applications," *IEEE Trans. Circuits Syst. I, Reg. Papers*, vol. 67, no. 5, pp. 1588–1601, May 2020, doi: 10.1109/TCSI.2019.2962516.

[269] A. Teman, G. Karakonstantis, R. Giterman, P. Meinerzhagen, and A. Burg, "Energy versus data integrity trade-offs in embedded high-density logic compatible dynamic memories," in *Proc. Design, Autom. Test Eur. Conf. Exhib. (DATE)*, Mar. 2015, pp. 489–494, doi: 10.7873/DATE.2015.0783.

[270] G. Stazi, L. Adani, A. Mastrandrea, M. Olivieri, and F. Menichelli, "Impact of approximate memory data allocation on a H.264 software video encoder," in *High Performance Computing* (Lecture Notes in Computer Science), R. Yokota, M. Weiland, J. Shalf, and S. Alam, Eds., Cham, Switzerland: Springer, 2018, pp. 545–553, doi: 10.1007/978-3-030-02465-9_38.

[271] F. Menichelli, G. Stazi, A. Mastrandrea, and M. Olivieri, "An emulator for approximate memory platforms based on QEmu," in *Applications in Electronics Pervading Industry, Environment and Society* (Lecture Notes in Electrical Engineering), A. De Gloria, Ed., Cham, Switzerland: Springer, 2018, pp. 153–159, doi: 10.1007/978-3-319-55071-8_20.

[272] M. Liu, H. Que, X. Yang, K. Zhang, Q. Yu, L. Yan, T. Wang, Y. Jin, and N. Zhou, "A selective bit dropping and encoding co-strategy in image processing for low-power design in DRAM and SRAM," *IEEE J. Emerg. Sel. Topics Circuits Syst.*, vol. 13, no. 1, pp. 48–57, Mar. 2023, doi: 10.1109/JETCAS.2023.3234402.

[273] Z. Shao, X. Chen, L. Du, L. Chen, Y. Du, W. Zhuang, H. Wei, C. Xie, and Z. Wang, "Memory-efficient CNN accelerator based on interlayer feature map compression," 2021, *arXiv:2110.06155*.

[274] A. Raha, S. Ghosh, D. Mohapatra, D. A. Mathaikutty, R. Sung, C. Brick, and V. Raghunathan, "Special session: Approximate TinyML systems: Full system approximations for extreme energy-efficiency in intelligent edge devices," in *Proc. IEEE 39th Int. Conf. Comput. Design (ICCD)*, Oct. 2021, pp. 13–16, doi: 10.1109/ICCD53106.2021.00015.

[275] A. Raha, S. Venkataramani, V. Raghunathan, and A. Raghunathan, "Energy-efficient reduce-and-rank using input-adaptive approximations," *IEEE Trans. Very Large Scale Integr. (VLSI) Syst.*, vol. 25, no. 2, pp. 462–475, Feb. 2017, doi: 10.1109/TVLSI.2016.2586379.

[276] D.-T. Nguyen, N.-M. Ho, M.-S. Le, W.-F. Wong, and I.-J. Chang, "ZEM: Zero-cycle bit-masking module for deep learning refreshless DRAM," *IEEE Access*, vol. 9, pp. 93723–93733, 2021, doi: 10.1109/ACCESS.2021.3088893.

[277] S. Pal, S. Bose, W.-H. Ki, and A. Islam, "Characterization of half-select free write assist 9T SRAM cell," *IEEE Trans. Electron Devices*, vol. 66, no. 11, pp. 4745–4752, Nov. 2019, doi: 10.1109/TED.2019.2942493.

[278] M. Imani and T. S. Rosing, "Approximate CPU and GPU design using emerging memory technologies," in *Approximate Circuits: Methodologies and CAD*, S. Reda and M. Shafique, Eds., Cham, Switzerland: Springer, 2019, pp. 383–398, doi: 10.1007/978-3-319-99322-5_19.

[279] A. M. H. Monazzah, M. Shoushtari, S. G. Miremadi, A. M. Rahmani, and N. Dutt, "QuARK: Quality-configurable approximate STT-MRAM cache by fine-grained tuning of reliability-energy knobs," in *Proc. IEEE/ACM Int. Symp. Low Power Electron. Design (ISLPED)*. Milwaukee, WI, USA: Quality, Jul. 2017, pp. 1–6, doi: 10.1109/ISLPED.2017.8009198.

[280] A. M. Hosseini Monazzah, A. M. Rahmani, A. Miele, and N. Dutt, "CAST: Content-aware STT-MRAM cache write management for different levels of approximation," *IEEE Trans. Comput.-Aided Design Integr. Circuits Syst.*, vol. 39, no. 12, pp. 4385–4398, Dec. 2020, doi: 10.1109/TCAD.2020.2986320.

[281] K. Desnos, M. Pelcat, J.-F. Nezan, and S. Aridhi, "On memory reuse between inputs and outputs of dataflow actors," *ACM Trans. Embedded Comput. Syst.*, vol. 15, no. 2, pp. 1–25, Feb. 2016, doi: 10.1145/2871744.

[282] S. Minakova and T. Stefanov, "Buffer sizes reduction for memory-efficient CNN inference on mobile and embedded devices," in *Proc. 23rd Euromicro Conf. Digit. Syst. Design (DSD)*, Aug. 2020, pp. 133–140, doi: 10.1109/DSD51259.2020.00031.

[283] S. Minakova and T. Stefanov, "Memory-throughput trade-off for CNN-based applications at the edge," *ACM Trans. Design Autom. Electron. Syst.*, vol. 28, no. 1, pp. 1–26, Dec. 2022, doi: 10.1145/3527457.

[284] D. Blalock, J. J. G. Ortiz, J. Frankle, and J. Guttag, "What is the state of neural network pruning?" 2020, *arXiv:2003.03033*.

[285] Y. Cheng, D. Wang, P. Zhou, and T. Zhang, "A survey of model compression and acceleration for deep neural networks," 2017, *arXiv:1710.09282*.

[286] H. Miomandre, J.-F. Nezan, D. Menard, A. Campbell, A. Griffin, S. Hall, and A. Ensor, "Approximate buffers for reducing memory requirements: Case study on SKA," in *Proc. IEEE Workshop Signal Process. Syst. (SiPS)*, Oct. 2020, pp. 1–6, doi: 10.1109/SiPS50750.2020.9195262.

[287] H. Miomandre, J.-F. Nezan, and D. Ménard, "Design space exploration for memory-oriented approximate computing techniques," in *Proc. IEEE 33rd Int. Conf. Appl.-specific Syst., Architectures Processors (ASAP)*, Jul. 2022, pp. 122–125, doi: 10.1109/ASAP54787.2022.00028.

[288] C. Gao, X. Xin, Y. Lu, Y. Zhang, J. Yang, and J. Shu, "ParaBit: Processing parallel bitwise operations in NAND flash memory based SSDs," in *Proc. 54th Annu. IEEE/ACM Int. Symp. Microarchitecture*. New York, NY, USA: ACM, Oct. 2021, pp. 59–70, doi: 10.1145/3466752.3480078.

[289] J. Choi, H.-J. Lee, and C. E. Rhee, "ADC-PIM: Accelerating convolution on the GPU via in-memory approximate data comparison," *IEEE J. Emerg. Sel. Topics Circuits Syst.*, vol. 12, no. 2, pp. 458–471, Jun. 2022, doi: 10.1109/JETCAS.2022.3167391.

[290] G. H. Lee, S. Hwang, J. Yu, and H. Kim, "Architecture and process integration overview of 3D NAND flash technologies," *Appl. Sci.*, vol. 11, no. 15, p. 6703, Jul. 2021, doi: 10.3390/app11156703.

[291] S.-J. Byun, D.-G. Kim, K.-D. Park, Y.-J. Choi, P. Kumar, I. Ali, D.-G. Kim, J.-M. Yoo, H.-K. Huh, Y.-J. Jung, S.-K. Kim, Y.-G. Pu, and K.-Y. Lee, "A low-power analog processor-in-memory-based convolutional neural network for biosensor applications," *Sensors*, vol. 22, no. 12, p. 4555, Jun. 2022, doi: 10.3390/s22124555.

[292] H. Jin, C. Liu, H. Liu, R. Luo, J. Xu, F. Mao, and X. Liao, "ReHy: A ReRAM-based digital/analog hybrid PIM architecture for accelerating CNN training," *IEEE Trans. Parallel Distrib. Syst.*, vol. 33, no. 11, pp. 2872–2884, Nov. 2022, doi: 10.1109/TPDS.2021.3138087.

[293] N. Hajinazar, G. F. Oliveira, S. Gregorio, J. D. Ferreira, N. M. Ghiasi, M. Patel, M. Alser, S. Ghose, J. Gómez-Luna, and O. Mutlu, "SIMDRAM: A framework for bit-serial SIMD processing using DRAM," in *Proc. 26th ACM Int. Conf. Architectural Support Program. Lang. Operating Syst.* New York, NY, USA: ACM, Apr. 2021, pp. 329–345, doi: 10.1145/3445814.3446749.

[294] H. Zhang, Y. Shu, Q. Deng, H. Sun, W. Zhao, and Y. Ha, "WDVR-RAM: A 0.25–1.2 V, 2.6–76 POPS/W charge-domain in-memory-computing binarized CNN accelerator for dynamic AIoT workloads," *IEEE Trans. Circuits Syst. I, Reg. Papers*, vol. 70, no. 10, pp. 3964–3977, Oct. 2023, doi: 10.1109/tcsi.2023.3294296.

[295] A. Ehrmann, T. Blachowicz, G. Ehrmann, and T. Grethe, "Recent developments in phase-change memory," *Appl. Res.*, vol. 1, no. 4, Dec. 2022, Art. no. e202200024, doi: 10.1002/appl.202200024.

[296] E. Garzón, L. Yavits, A. Teman, and M. Lanuzza, "Approximate content-addressable memories: A review," *Chips*, vol. 2, no. 2, pp. 70–82, Mar. 2023, doi: 10.3390/chips2020005.

[297] Y. Fu and Y. Wu, "CARAM: A content-aware hybrid PCM/DRAM main memory system framework," in *Proc. IFIP Int. Conf. Netw. Parallel Comput.*, Zhengzhou, China. Heidelberg, Germany: Springer, Sep. 2020, pp. 243–248, doi: 10.1007/978-3-030-79478-1_21.

[298] T. V. Mahendra, S. W. Hussain, S. Mishra, and A. Dandapat, "Energy-efficient precharge-free ternary content addressable memory (TCAM) for high search rate applications," *IEEE Trans. Circuits Syst. I, Reg. Papers*, vol. 67, no. 7, pp. 2345–2357, Jul. 2020, doi: 10.1109/TCSI.2020.2978295.

[299] H. Zhan, C. Wang, H. Cui, X. Liu, F. Liu, and X. Cheng, "High-speed and energy-efficient single-port content addressable memory to achieve dual-port operation," in *Proc. Design, Autom. Test Eur. Conf. Exhib. (DATE)*, vol. 56, Apr. 2023, pp. 1–6, doi: 10.23919/date56975.2023.10137206.

[300] G. Stazi, A. Mastrandrea, M. Olivieri, and F. Menichelli, "Full system emulation of approximate memory platforms with AppropinQuo," *J. Low Power Electron.*, vol. 15, no. 1, pp. 30–39, Mar. 2019, doi: 10.1166/jolpe.2019.1595.

[301] M. Yayla, Z. Valipour Dehnoo, M. Masoudinejad, and J.-J. Chen, "TREAM: A tool for evaluating error resilience of tree-based models using approximate memory," in *Embedded Computer Systems: Architectures, Modeling, and Simulation* (Lecture Notes in Computer Science), A. Orailoglu, M. Reichenbach, and M. Jung, Eds., Cham, Switzerland: Springer, 2022, pp. 61–73, doi: 10.1007/978-3-031-15074-6_4.

[302] R. Yarmand, M. Kamal, A. Afzali-Kusha, and M. Pedram, "DART: A framework for determining approximation levels in an approximable memory hierarchy," *IEEE Trans. Very Large Scale Integr. (VLSI) Syst.*, vol. 28, no. 1, pp. 273–286, Jan. 2020, doi: 10.1109/TVLSI.2019.2935832.

[303] F. Ferdaus, B. M. S. B. Talukder, and M. T. Rahman, "Approximate MRAM: High-performance and power-efficient computing with MRAM chips for error-tolerant applications," *IEEE Trans. Comput.*, vol. 72, no. 3, pp. 668–681, Mar. 2023, doi: 10.1109/TC.2022.3174584.

[304] K. Kim, S.-J. Jang, J. Park, E. Lee, and S.-S. Lee, "Lightweight and energy-efficient deep learning accelerator for real-time object detection on edge devices," *Sensors*, vol. 23, no. 3, p. 1185, Jan. 2023, doi: 10.3390/s23031185.

[305] J. Bonnot, A. Mercat, E. Nogues, and D. Ménard, "Approximate computing at the algorithmic level," in *Approximate Computing Techniques: From Component- to Application-Level*. Cham, Switzerland: Springer, 2022, pp. 109–142, doi: 10.1007/978-3-030-94705-7_5.

[306] J. Murray, P. Wettin, P. P. Pande, and B. Shirazi, "Dynamic voltage and frequency scaling," in *Sustainable Wireless Network-on-Chip Architectures*, J. Murray, P. Wettin, P. P. Pande, and B. Shirazi, Eds., Boston, MA, USA: Morgan Kaufmann, 2016, pp. 79–105, doi: 10.1016/B978-0-12-803625-9.00014-5.

[307] H. Ali, U. U. Tariq, J. Hardy, X. Zhai, L. Lu, Y. Zheng, F. Bensaali, A. Amira, K. Fatema, and N. Antonopoulos, "A survey on system level energy optimisation for MPSoCs in IoT and consumer electronics," *Comput. Sci. Rev.*, vol. 41, Aug. 2021, Art. no. 100416, doi: 10.1016/j.cosrev.2021.100416.

[308] A. Garg and P. Kulkarni, "Dynamic memory management for GPU-based training of deep neural networks," in *Proc. IEEE Int. Parallel Distrib. Process. Symp. (IPDPS)*, May 2019, pp. 200–209, doi: 10.1109/IPDPS.2019.00030.

[309] S. Pandey, L. Siddhu, and P. R. Panda, "NeuroCool: Dynamic thermal management of 3D DRAM for deep neural networks through customized prefetching," *ACM Trans. Design Autom. Electron. Syst.*, vol. 29, no. 1, pp. 1–35, Jan. 2024, doi: 10.1145/3630012.

[310] S. Li, Z. Zhang, R. Mao, J. Xiao, L. Chang, and J. Zhou, "A fast and energy-efficient SNN processor with adaptive clock/event-driven computation scheme and online learning," *IEEE Trans. Circuits Syst. I, Reg. Papers*, vol. 68, no. 4, pp. 1543–1552, Apr. 2021, doi: 10.1109/TCSI.2021.3052885.

[311] R. N. Tadros and P. A. Beerel, "A robust and self-adaptive clocking technique for SFQ circuits," *IEEE Trans. Appl. Supercond.*, vol. 28, no. 7, pp. 1–11, Oct. 2018, doi: 10.1109/TASC.2018.2856836.

[312] S. Vangal, S. Paul, S. Hsu, A. Agarwal, R. Krishnamurthy, J. Tschanz, and V. De, "Near-threshold voltage design techniques for heterogenous manycore system-on-chips," *J. Low Power Electron. Appl.*, vol. 10, no. 2, p. 16, May 2020, doi: 10.3390/jlpea10020016.

[313] J. Baik, J. Lee, and K. Kang, "Task migration and scheduler for mixed-criticality systems," *Sensors*, vol. 22, no. 5, p. 1926, Mar. 2022, doi: 10.3390/s22051926.

[314] D. D'Agostino, I. Merelli, M. Aldinucci, and D. Cesini, "Hardware and software solutions for energy-efficient computing in scientific programming," *Sci. Program.*, vol. 2021, pp. 1–9, Jun. 2021, doi: 10.1155/2021/5514284.

[315] A. S. Baroughi, S. Huemer, H. S. Shahhoseini, and N. TaheriNejad, "AxE: An approximate-exact multi-processor system-on-chip platform," in *Proc. 25th Euromicro Conf. Digit. Syst. Design (DSD)*, Maspalomas, Spain, Aug. 2022, pp. 60–66, doi: 10.1109/DSD57027.2022.00018.

[316] A. Aponte-Moreno, F. Restrepo-Calle, and C. Pedraza, "A low-cost fault tolerance method for ARM and RISC-V microprocessor-based systems using temporal redundancy and approximate computing through simplified iterations," *J. Integr. Circuits Syst.*, vol. 16, no. 3, pp. 1–14, Apr. 2022, doi: 10.29292/jics.v16i3.539.

[317] İ. Taştan, M. Karaca, and A. Yurdakul, "Approximate CPU design for IoT end-devices with learning capabilities," *Electronics*, vol. 9, no. 1, p. 125, Jan. 2020, doi: 10.3390/electronics9010125.

[318] I. M. A. Jawarneh, P. Bellavista, A. Corradi, L. Foschini, and R. Montanari, "SpatialSSJP: QoS-aware adaptive approximate stream-static spatial join processor," *IEEE Trans. Parallel Distrib. Syst.*, vol. 35, no. 1, pp. 73–88, Jan. 2024, doi: 10.1109/TPDS.2023.3330669.

[319] M. E. Elbtity, P. S. Chandarana, B. Reidy, J. K. Eshraghian, and R. Zand, "APTPU: Approximate computing based tensor processing unit," *IEEE Trans. Circuits Syst. I, Reg. Papers*, vol. 69, no. 12, pp. 5135–5146, Dec. 2022, doi: 10.1109/TCSI.2022.3206262.

[320] A. Siddique, K. Basu, and K. A. Hoque, "Exploring fault-energy trade-offs in approximate DNN hardware accelerators," in *Proc. 22nd Int. Symp. Quality Electron. Design (ISQED)*, Apr. 2021, pp. 343–348, doi: 10.1109/ISQED51717.2021.9424345.

[321] M. H. Ahmadilivani, M. Barbareschi, S. Barone, A. Bosio, M. Daneshtalab, S. D. Torca, G. Gavarini, M. Jenihhin, J. Raik, A. Ruospo, E. Sanchez, and M. Taheri, "Special session: Approximation and fault resiliency of DNN accelerators," in *Proc. IEEE 41st VLSI Test Symp. (VTS)*, Apr. 2023, pp. 1–10, doi: 10.1109/VTS56346.2023.10140043.

[322] J. N. Mitchell, "Computer multiplication and division using binary logarithms," *IRE Trans. Electron. Comput.*, vols. EC–11, no. 4, pp. 512–517, Aug. 1962, doi: 10.1109/TEC.1962.5219391.

[323] A. B. Kahng and S. Kang, "Accuracy-configurable adder for approximate arithmetic designs," in *Proc. DAC Design Autom. Conf.*, Jun. 2012, pp. 820–825, doi: 10.1145/2228360.2228509.

[324] M. Shafique, W. Ahmad, R. Hafiz, and J. Henkel, "A low latency generic accuracy configurable adder," in *Proc. 52nd ACM/EDAC/IEEE Design Autom. Conf. (DAC)*, Jun. 2015, pp. 1–6, doi: 10.1145/2744769.2744778.

[325] H. R. Mahdiani, A. Ahmadi, S. M. Fakhraie, and C. Lucas, "Bio-inspired imprecise computational blocks for efficient VLSI implementation of soft-computing applications," *IEEE Trans. Circuits Syst. I, Reg. Papers*, vol. 57, no. 4, pp. 850–862, Apr. 2010, doi: 10.1109/TCSI.2009.2027626.

[326] V. Gupta, D. Mohapatra, A. Raghunathan, and K. Roy, "Low-power digital signal processing using approximate adders," *IEEE Trans. Comput.-Aided Design Integr. Circuits Syst.*, vol. 32, no. 1, pp. 124–137, Jan. 2013, doi: 10.1109/TCAD.2012.2217962.

[327] K. Du, P. Varman, and K. Mohanram, "High performance reliable variable latency carry select addition," in *Proc. Design, Autom. Test Eur. Conf. Exhib. (DATE)*, Mar. 2012, pp. 1257–1262, doi: 10.1109/DATE.2012.6176685.

[328] S. Singh and Y. B. Shukla, "Low power carry select adder using FinFET technology," in *Proc. 6th Int. Conf. Devices, Circuits Syst. (ICDCS)*, Apr. 2022, pp. 152–155, doi: 10.1109/ICDCS54290.2022.9780840.

[329] C. R. N. Praneeth, Ch. U. Kumari, T. Padma, and N. A. Vignesh, "Low-Energy-Consumption design: 16 bit block based carry speculative approximate adder," in *Proc. IEEE 3rd Global Conf. for Advancement Technol. (GCAT)*, Oct. 2022, pp. 1–4, doi: 10.1109/GCAT55367.2022.9972197.

[330] H. Ghabeli, A. S. Molahosseini, A. A. E. Zarandi, and L. Sousa, "Variable latency carry speculative adders with input-based dynamic configuration," *Comput. Electr. Eng.*, vol. 93, Jul. 2021, Art. no. 107247, doi: 10.1016/j.compeleceng.2021.107247.

[331] A. Najafi, M. Weißbrich, G. Payá-Vayá, and A. Garcia-Ortiz, "Coherent design of hybrid approximate adders: Unified design framework and metrics," *IEEE J. Emerg. Sel. Topics Circuits Syst.*, vol. 8, no. 4, pp. 736–745, Dec. 2018, doi: 10.1109/JETCAS.2018.2833284.

[332] G. Giustolisi and G. Palumbo, "Hybrid full adders: Optimized design, critical review and comparison in the energy-delay space," *Electronics*, vol. 11, no. 19, p. 3220, Oct. 2022, doi: 10.3390/electronics11193220.

[333] H.-R. basireddy, K. Challa, and T. Nikoubin, "Hybrid logical effort for hybrid logic style full adders in multistage structures," *IEEE Trans. Very Large Scale Integr. (VLSI) Syst.*, vol. 27, no. 5, pp. 1138–1147, May 2019, doi: 10.1109/TVLSI.2018.2889833.

[334] A. Nandal and M. Kumar, "Design and implementation of CMOS full adder circuit with ECRL and sleepy keeper technique," in *Proc. Int. Conf. Adv. Comput., Commun. Control Netw. (ICACCCN)*, Oct. 2018, pp. 733–738, doi: 10.1109/ICACCCN.2018.8748336.

[335] M. Agarwal, N. Agrawal, and Md. A. Alam, "A new design of low power high speed hybrid CMOS full adder," in *Proc. Int. Conf. Signal Process. Integr. Netw. (SPIN)*, Feb. 2014, pp. 448–452, doi: 10.1109/SPIN.2014.6776995.

[336] A. M. Hassani, M. Rezaalipour, and M. Dehyadegari, "A novel ultra low power accuracy configurable adder at transistor level," in *Proc. 8th Int. Conf. Comput. Knowl. Eng. (ICCKE)*, Oct. 2018, pp. 165–170, doi: 10.1109/ICCKE.2018.8566643.

[337] M. Kumar, S. K. Arya, and S. Pandey, "Single bit full adder design using 8 transistors with novel 3 transistors XNOR gate," 2012, *arXiv:1201.1966*.

[338] P. Kumar and R. K. Sharma, "Low voltage high performance hybrid full adder," *Eng. Sci. Technol., Int. J.*, vol. 19, no. 1, pp. 559–565, Mar. 2016, doi: 10.1016/j.jestch.2015.10.001.

[339] M. Hasan, Md. J. Hossein, M. Hossain, H. U. Zaman, and S. Islam, "Design of a scalable low-power 1-Bit hybrid full adder for fast computation," *IEEE Trans. Circuits Syst. II, Exp. Briefs*, vol. 67, no. 8, pp. 1464–1468, Aug. 2020, doi: 10.1109/TCSII.2019.2940558.

[340] J. Kandpal, A. Tomar, M. Agarwal, and K. K. Sharma, "High-speed hybrid-logic full adder using high-performance 10-T XOR–XNOR cell," *IEEE Trans. Very Large Scale Integr. (VLSI) Syst.*, vol. 28, no. 6, pp. 1413–1422, Jun. 2020, doi: 10.1109/TVLSI.2020.2983850.

[341] V. Gupta, D. Mohapatra, S. P. Park, A. Raghunathan, and K. Roy, "IMPACT: IMPrecise adders for low-power approximate computing," in *Proc. IEEE/ACM Int. Symp. Low Power Electron. Design*, Aug. 2011, pp. 409–414, doi: 10.1109/ISLPED.2011.5993675.

[342] H. Naseri and S. Timarchi, "Low-power and fast full adder by exploring new XOR and XNOR gates," *IEEE Trans. Very Large Scale Integr. (VLSI) Syst.*, vol. 26, no. 8, pp. 1481–1493, Aug. 2018, doi: 10.1109/TVLSI.2018.2820999.

[343] T. Nirmalraj, S. K. Pandiyan, R. K. Karan, R. Sivaraman, and R. Amirtharajan, "Design of low-power 10-transistor full adder using GDI technique for energy-efficient arithmetic applications," *Circuits, Syst., Signal Process.*, vol. 42, no. 6, pp. 3649–3667, Jan. 2023, doi: 10.1007/s00034-022-02287-x.

[344] A. Bhargav and P. Huynh, "Design and analysis of low-power and high speed approximate adders using CNFETs," *Sensors*, vol. 21, no. 24, p. 8203, Dec. 2021, doi: 10.3390/s21248203.

[345] J. Lee, H. Seo, H. Seok, and Y. Kim, "A novel approximate adder design using error reduced carry prediction and constant truncation," *IEEE Access*, vol. 9, pp. 119939–119953, 2021, doi: 10.1109/ACCESS.2021.3108443.

[346] K. Chen, W. Liu, J. Han, and F. Lombardi, "Profile-based output error compensation for approximate arithmetic circuits," *IEEE Trans. Circuits Syst. I, Reg. Papers*, vol. 67, no. 12, pp. 4707–4718, Dec. 2020, doi: 10.1109/TCSI.2020.2996567.

[347] P. Albicocco, G. C. Cardarilli, A. Nannarelli, M. Petricca, and M. Re, "Imprecise arithmetic for low power image processing," in *Proc. Conf. Rec. 46th Asilomar Conf. Signals, Syst. Comput. (ASILOMAR)*, Nov. 2012, pp. 983–987, doi: 10.1109/ACSSC.2012.6489164.

[348] P. Balasubramanian and D. L. Maskell, "Hardware optimized and error reduced approximate adder," *Electronics*, vol. 8, no. 11, p. 1212, Oct. 2019, doi: 10.3390/electronics8111212.

[349] P. Balasubramanian, R. Nayar, D. L. Maskell, and N. E. Mastorakis, "An approximate adder with a near-normal error distribution: Design, error analysis and practical application," *IEEE Access*, vol. 9, pp. 4518–4530, 2021, doi: 10.1109/ACCESS.2020.3047651.

[350] H. Seo, Y. S. Yang, and Y. Kim, "Design and analysis of an approximate adder with hybrid error reduction," *Electronics*, vol. 9, no. 3, p. 471, Mar. 2020, doi: 10.3390/electronics9030471.

[351] M. M. A. D. Rosa, G. Paim, P. U. L. D. Costa, E. A. C. D. Costa, R. I. Soares, and S. Bampi, "AxPPA: Approximate parallel prefix adders," *IEEE Trans. Very Large Scale Integr. (VLSI) Syst.*, vol. 31, no. 1, pp. 17–28, Jan. 2023, doi: 10.1109/TVLSI.2022.3218021.

[352] R. Roy, J. Raiman, N. Kant, I. Elkin, R. Kirby, M. Siu, S. Oberman, S. Godil, and B. Catanzaro, "PrefixRL: Optimization of parallel prefix circuits using deep reinforcement learning," in *Proc. 58th ACM/IEEE Design Autom. Conf. (DAC)*. Piscataway, NJ, USA: IEEE Press, Dec. 2021, pp. 853–858, doi: 10.1109/DAC18074.2021.9586094.

[353] P. Balasubramanian, R. Nayar, and D. L. Maskell, "Gate-level static approximate adders: A comparative analysis," *Electronics*, vol. 10, no. 23, p. 2917, Nov. 2021, doi: 10.3390/electronics10232917.

[354] S. Xu and B. C. Schafer, "Exposing approximate computing optimizations at different levels: From behavioral to gate-level," *IEEE Trans. Very Large Scale Integr. (VLSI) Syst.*, vol. 25, no. 11, pp. 3077–3088, Nov. 2017, doi: 10.1109/TVLSI.2017.2735299.

[355] V. Camus, M. Cacciotti, J. Schlachter, and C. Enz, "Design of approximate circuits by fabrication of false timing paths: The carry cut-back adder," *IEEE J. Emerg. Sel. Topics Circuits Syst.*, vol. 8, no. 4, pp. 746–757, Dec. 2018, doi: 10.1109/JETCAS.2018.2851749.

[356] M. Pashaeifar, M. Kamal, A. Afzali-Kusha, and M. Pedram, "Approximate reverse carry propagate adder for energy-efficient DSP applications," *IEEE Trans. Very Large Scale Integr. (VLSI) Syst.*, vol. 26, no. 11, pp. 2530–2541, Nov. 2018, doi: 10.1109/TVLSI.2018.2859939.

[357] D. Esposito, A. G. M. Strollo, E. Napoli, D. De Caro, and N. Petra, "Approximate multipliers based on new approximate compressors," *IEEE Trans. Circuits Syst. I, Reg. Papers*, vol. 65, no. 12, pp. 4169–4182, Dec. 2018, doi: 10.1109/TCSI.2018.2839266.

[358] F. Sabetzadeh, M. H. Moaiyeri, and M. Ahmadinejad, "An ultra-efficient approximate multiplier with error compensation for error-resilient applications," *IEEE Trans. Circuits Syst. II, Exp. Briefs*, vol. 70, no. 2, pp. 776–780, Feb. 2023, doi: 10.1109/TCSII.2022.3215065.

[359] F. Farshchi, M. S. Abrishami, and S. M. Fakhraie, "New approximate multiplier for low power digital signal processing," in *Proc. 17th CSI Int. Symp. Comput. Archit. Digit. Syst. (CADS)*, Oct. 2013, pp. 25–30, doi: 10.1109/CADS.2013.6714233.

[360] A. S. Roy, H. Agrawal, and A. S. Dhar, "ACBAM-accuracy-configurable sign inclusive broken array booth multiplier design," *IEEE Trans. Emerg. Topics Comput.*, vol. 10, no. 4, pp. 2072–2078, Oct. 2022, doi: 10.1109/TETC.2021.3107509.

[361] Z. Wang, G. A. Jullien, and W. C. Miller, "A new design technique for column compression multipliers," *IEEE Trans. Comput.*, vol. 44, no. 8, pp. 962–970, Aug. 1995, doi: 10.1109/12.403712.

[362] P. J. Edavoor, S. Raveendran, and A. D. Rahulkar, "Approximate multiplier design using novel dual-stage 4:2 compressors," *IEEE Access*, vol. 8, pp. 48337–48351, 2020, doi: 10.1109/ACCESS.2020.2978773.

[363] R. Dornelles, G. Paim, B. Silveira, M. Fonseca, E. Costa, and S. Bampi, "A power-efficient 4-2 adder compressor topology," in *Proc. 15th IEEE Int. New Circuits Syst. Conf. (NEWCAS)*, Jun. 2017, pp. 281–284, doi: 10.1109/NEWCAS.2017.8010160.

[364] W.-C. Yeh and C.-W. Jen, "High-speed booth encoded parallel multiplier design," *IEEE Trans. Comput.*, vol. 49, no. 7, pp. 692–701, Jul. 2000, doi: 10.1109/12.863039.

[365] Y. Zhu, W. Liu, P. Yin, T. Cao, J. Han, and F. Lombardi, "Design, evaluation and application of approximate-truncated booth multipliers," *IET Circuits, Devices Syst.*, vol. 14, no. 8, pp. 1305–1317, Nov. 2020, doi: 10.1049/iet-cds.2019.0398.

[366] M. H. Haider, H. Zhang, and S.-B. Ko, "Decoder reduction approximation scheme for booth multipliers," *IEEE Trans. Comput.*, vol. 73, no. 3, pp. 735–746, Mar. 2024, doi: 10.1109/tc.2023.3343093.

[367] P. Kulkarni, P. Gupta, and M. Ercegovac, "Trading accuracy for power with an underdesigned multiplier architecture," in *Proc. 24th Int. Conf. VLSI Design*, Jan. 2011, pp. 346–351, doi: 10.1109/VLSID.2011.51.

[368] H. Waris, C. Wang, W. Liu, J. Han, and F. Lombardi, "Hybrid partial product-based high-performance approximate recursive multipliers," *IEEE Trans. Emerg. Topics Comput.*, vol. 10, no. 1, pp. 507–513, Jan. 2022, doi: 10.1109/TETC.2020.3013977.

[369] S. Rehman, W. El-Harouni, M. Shafique, A. Kumar, J. Henkel, and J. Henkel, "Architectural-space exploration of approximate multipliers," in *Proc. IEEE/ACM Int. Conf. Comput.-Aided Design (ICCAD)*, Nov. 2016, pp. 1–8, doi: 10.1145/2966986.2967005.

[370] S. Venkatachalam and S.-B. Ko, ''Design of power and area efficient approximate multipliers,'' *IEEE Trans. Very Large Scale Integr. (VLSI) Syst.*, vol. 25, no. 5, pp. 1782–1786, May 2017, doi: 10.1109/TVLSI.2016.2643639.

[371] S. Ullah, S. S. Sahoo, N. Ahmed, D. Chaudhury, and A. Kumar, ''*AppAxO*: Designing application-specific approximate operators for FPGA-based embedded systems,'' *ACM Trans. Embedded Comput. Syst.*, vol. 21, no. 3, pp. 1–31, May 2022, doi: 10.1145/3513262.

[372] T. Zhang, Z. Niu, and J. Han, ''A brief review of logarithmic multiplier designs,'' in *Proc. IEEE 23rd Latin Amer. Test Symp. (LATS)*, Sep. 2022, pp. 1–4, doi: 10.1109/LATS57337.2022.9936921.

[373] Y. Wu, C. Chen, W. Xiao, X. Wang, C. Wen, J. Han, X. Yin, W. Qian, and C. Zhuo, ''A survey on approximate multiplier designs for energy efficiency: From algorithms to circuits,'' *ACM Trans. Design Autom. Electron. Syst.*, vol. 29, no. 1, pp. 1–37, Jan. 2024, doi: 10.1145/3610291.

[374] R. Pilipovic, P. Bulic, and U. Lotric, ''A two-stage operand trimming approximate logarithmic multiplier,'' *IEEE Trans. Circuits Syst. I, Reg. Papers*, vol. 68, no. 6, pp. 2535–2545, Jun. 2021, doi: 10.1109/TCSI.2021.3069168.

[375] R. Makimoto, T. Imagawa, and H. Ochi, ''Approximate logarithmic multipliers using half compensation with two line segments,'' in *Proc. IEEE 36th Int. Syst.-Chip Conf. (SOCC)*, Sep. 2023, pp. 1–6, doi: 10.1109/socc58585.2023.10256796.

[376] S. Yu, M. Tasnim, and S. X.-D. Tan, ''HEALM: Hardware-efficient approximate logarithmic multiplier with reduced error,'' in *Proc. 27th Asia South Pacific Design Autom. Conf. (ASP-DAC)*, Jan. 2022, pp. 37–42, doi: 10.1109/ASP-DAC52403.2022.9712543.

[377] M. S. Ansari, H. Jiang, B. F. Cockburn, and J. Han, ''Low-power approximate multipliers using encoded partial products and approximate compressors,'' *IEEE J. Emerg. Sel. Topics Circuits Syst.*, vol. 8, no. 3, pp. 404–416, Sep. 2018, doi: 10.1109/JETCAS.2018.2832204.

[378] P. Choudhary, L. Bhargava, M. Fujita, and V. Singh, ''LUT-based arithmetic circuit approximation with formal guarantee on worst case relative error,'' in *Proc. IEEE 24th Latin Amer. Test Symp. (LATS)*, Mar. 2023, pp. 1–2, doi: 10.1109/LATS58125.2023.10154494.

[379] U. S. Patankar, M. E. Flores, and A. Koel, ''Novel data dependent divider circuit block implementation for complex division and area critical applications,'' *Sci. Rep.*, vol. 13, no. 1, Feb. 2023, Art. no. 1, doi: 10.1038/s41598-023-28343-3.

[380] T. Deepa, P. Kaumudi, K. Sonali, and P. Saraswat, ''Design and implementation of approximate divider for error-resilient image processing applications,'' in *Proc. 2nd Int. Conf. Electr., Electron., Inf. Commun. Technol. (ICEEICT)*, Apr. 2023, pp. 1–5, doi: 10.1109/ICEEICT56924.2023.10157050.

[381] D. Piso, J. A. Pineiro, and J. D. Bruguera, ''Analysis of the impact of different methods for division/square root computation in the performance of a superscalar microprocessor,'' in *Proc. Euromicro Symp. Digit. Syst. Design. Archit., Methods Tools*, vol. 39, Oct. 2002, pp. 218–225, doi: 10.1109/DSD.2002.1115372.

[382] J. Oelund and S. Kim, ''ILAFD: Accuracy-configurable floating-point divider using an approximate reciprocal and an iterative logarithmic multiplier,'' in *Proc. Great Lakes Symp. VLSI*. New York, NY, USA: ACM, Jun. 2023, pp. 639–644, doi: 10.1145/3583781.3590262.

[383] S. Vahdat, M. Kamal, A. Afzali-Kusha, M. Pedram, and Z. Navabi, ''TruncApp: A truncation-based approximate divider for energy efficient DSP applications,'' in *Proc. Design, Autom. Test Eur. Conf. Exhib. (DATE)*, Mar. 2017, pp. 1635–1638, doi: 10.23919/DATE.2017.7927254.

[384] A. Shriram, A. Tiwari, U. A. Kumar, B. R. T. Karri, S. Veeramachaneni, and S. E. Ahmed, ''Power efficient approximate divider architecture for error resilient applications,'' in *Proc. IEEE 6th Conf. Inf. Commun. Technol. (CICT)*, Nov. 2022, pp. 1–6, doi: 10.1109/CICT56698.2022.9997960.

[385] S. Behroozi, J. Li, J. Melchert, and Y. Kim, ''SAADI: A scalable accuracy approximate divider for dynamic energy-quality scaling,'' in *Proc. 24th Asia South Pacific Design Autom. Conf.* New York, NY, USA: ACM, Jan. 2019, pp. 481–486, doi: 10.1145/3287624.3287668.

[386] P. Malík, ''High throughput floating point exponential function implemented in FPGA,'' in *Proc. IEEE Comput. Soc. Annu. Symp. VLSI*, Jul. 2015, pp. 97–100, doi: 10.1109/ISVLSI.2015.61.

[387] O. I. Bureneva and O. U. Kaidanovich, ''FPGA-based hardware implementation of fixed-point division using Newton–Raphson method,'' in *Proc. IV Int. Conf. Neural Netw. Neurotechnologies (NeuroNT)*, Jun. 2023, pp. 45–47, doi: 10.1109/NeuroNT58640.2023.10175844.

[388] Z. Ebrahimi, M. Zaid, M. Wijtvliet, and A. Kumar, ''RAPID: Approximate pipelined soft multipliers and dividers for high throughput and energy efficiency,'' *IEEE Trans. Comput.-Aided Design Integr. Circuits Syst.*, vol. 42, no. 3, pp. 712–725, Mar. 2023, doi: 10.1109/TCAD.2022.3184928.

[389] H. Wang, K. Chen, B. Wu, C. Wang, W. Liu, and F. Lombardi, ''HEADiv: A high-accuracy energy-efficient approximate divider with error compensation,'' in *Proc. 17th ACM Int. Symp. Nanosc. Architectures*. New York, NY, USA: ACM, May 2023, pp. 1–6, doi: 10.1145/3565478.3572324.

[390] C. Jha and J. Mekie, ''Design of novel CMOS based inexact subtractors and dividers for approximate computing: An in-depth comparison with PTL based designs,'' in *Proc. 22nd Euromicro Conf. Digit. Syst. Design (DSD)*, Aug. 2019, pp. 174–181, doi: 10.1109/DSD.2019.00034.

[391] W. Liu, T. Xu, J. Li, C. Wang, P. Montuschi, and F. Lombardi, ''Design of unsigned approximate hybrid dividers based on restoring array and logarithmic dividers,'' *IEEE Trans. Emerg. Topics Comput.*, vol. 10, no. 1, pp. 339–350, Jan. 2022, doi: 10.1109/TETC.2020.3022290.

[392] Y. Wu, H. Jiang, Z. Ma, P. Gou, Y. Lu, J. Han, S. Yin, S. Wei, and L. Liu, ''An energy-efficient approximate divider based on logarithmic conversion and piecewise constant approximation,'' *IEEE Trans. Circuits Syst. I, Reg. Papers*, vol. 69, no. 7, pp. 2655–2668, Jul. 2022, doi: 10.1109/TCSI.2022.3167894.

[393] L. Chen, J. Han, W. Liu, P. Montuschi, and F. Lombardi, ''Design, evaluation and application of approximate high-radix dividers,'' *IEEE Trans. Multi-Scale Comput. Syst.*, vol. 4, no. 3, pp. 299–312, Jul. 2018, doi: 10.1109/TMSCS.2018.2817608.

[394] D. Hendrycks and K. Gimpel, ''Gaussian error linear units (GELUs),'' 2016, *arXiv:1606.08415*.

[395] *Apply Gaussian Error Linear Unit (GELU) Activation—MATLAB Gelu*. Accessed: Aug. 30, 2023. [Online]. Available: https://www.mathworks.com/help/deeplearning/ref/dlarray.gelu.html

[396] *Papers With Code—GELU Explained*. Accessed: Aug. 29, 2023. [Online]. Available: https://paperswithcode.com/method/gelu

[397] P.-T.-P. Tang, ''Table-driven implementation of the exponential function in IEEE floating-point arithmetic,'' *ACM Trans. Math. Softw.*, vol. 15, no. 2, pp. 144–157, Jun. 1989, doi: 10.1145/63522.214389.

[398] H. de Lassus Saint-Geniès, D. Defour, and G. Revy, ''Exact lookup tables for the evaluation of trigonometric and hyperbolic functions,'' *IEEE Trans. Comput.*, vol. 66, no. 12, pp. 2058–2071, Dec. 2017, doi: 10.1109/TC.2017.2703870.

[399] A. G. M. Strollo, D. De Caro, and N. Petra, ''Elementary functions hardware implementation using constrained piecewise-polynomial approximations,'' *IEEE Trans. Comput.*, vol. 60, no. 3, pp. 418–432, Mar. 2011, doi: 10.1109/TC.2010.127.

[400] P. Nilsson, A. U. R. Shaik, R. Gangarajaiah, and E. Hertz, ''Hardware implementation of the exponential function using Taylor series,'' in *Proc. NORCHIP*, Oct. 2014, pp. 1–4, doi: 10.1109/NORCHIP.2014.7004740.

[401] B. Xiong, Y. Sui, Z. Jia, S. Li, and Y. Chang, ''Utilize the shift- and-add architecture to approximate multiple nonlinear functions,'' *Int. J. Circuit Theory Appl.*, vol. 49, no. 7, pp. 2290–2297, Jul. 2021, doi: 10.1002/cta.2994.

[402] B. Lakshmi and A. S. Dhar, ''CORDIC architectures: A survey,'' *VLSI Design*, vol. 2010, pp. 1–19, Mar. 2010, doi: 10.1155/2010/794891.

[403] J. E. Volder, ''The CORDIC trigonometric computing technique,'' *IRE Trans. Electron. Comput.*, vol. EC-8, no. 3, pp. 330–334, Sep. 1959, doi: 10.1109/TEC.1959.5222693.

[404] A. M. Dalloo, A. J. Humaidi, A. K. A. Mhdawi, and H. Al-Raweshidy, ''Low-power and low-latency hardware implementation of approximate hyperbolic and exponential functions for embedded system applications,'' *IEEE Access*, vol. 12, pp. 24151–24163, 2024, doi: 10.1109/access.2024.3364361.

[405] Y. Liu and K. K. Parhi, ''Computing hyperbolic tangent and sigmoid functions using stochastic logic,'' in *Proc. 50th Asilomar Conf. Signals, Syst. Comput.*, Nov. 2016, pp. 1580–1585, doi: 10.1109/ACSSC.2016.7869645.

[406] K. K. Parhi and Y. Liu, "Computing arithmetic functions using stochastic logic by series expansion," *IEEE Trans. Emerg. Topics Comput.*, vol. 7, no. 1, pp. 44–59, Jan. 2019, doi: 10.1109/TETC.2016.2618750.

[407] L. Huai, P. Li, G. E. Sobelman, and D. J. Lilja, "Stochastic computing implementation of trigonometric and hyperbolic functions," in *Proc. IEEE 12th Int. Conf. ASIC (ASICON)*, Oct. 2017, pp. 553–556, doi: 10.1109/ASICON.2017.8252535.

[408] L. Chen, F. Lombardi, J. Han, and W. Liu, "A fully parallel approximate CORDIC design," in *Proc. IEEE/ACM Int. Symp. Nanosc. Architectures (NANOARCH)*, Jul. 2016, pp. 197–202, doi: 10.1145/2950067.2950076.

[409] S. Rai and R. Srivastava, "FPGA realization of scale-free CORDIC algorithm-based window functions," in *Recent Trends in Communication, Computing, and Electronics* (Lecture Notes in Electrical Engineering), A. Khare, U. S. Tiwary, I. K. Sethi, and N. Singh, Eds., Singapore: Springer, 2019, pp. 245–257, doi: 10.1007/978-981-13-2685-1_24.

[410] L. Chen, J. Han, W. Liu, and F. Lombardi, "Algorithm and design of a fully parallel approximate coordinate rotation digital computer (CORDIC)," *IEEE Trans. Multi-Scale Comput. Syst.*, vol. 3, no. 3, pp. 139–151, Jul. 2017, doi: 10.1109/TMSCS.2017.2696003.

[411] R. K. Yousif, I. A. Hashim, and B. H. Abd, "Low area FPGA implementation of hyperbolic tangent function," in *Proc. 6th Int. Conf. Eng. Technol. Appl. (IICETA)*, vol. 3, Jul. 2023, pp. 596–602, doi: 10.1109/iiceta57613.2023.10351345.

[412] F. Ortega-Zamorano, J. M. Jerez, G. Juárez, J. O. Pérez, and L. Franco, "High precision FPGA implementation of neural network activation functions," in *Proc. IEEE Symp. Intell. Embedded Syst. (IES)*, Dec. 2014, pp. 55–60, doi: 10.1109/INTELES.2014.7008986.

[413] S. Sorayassa, M. Ahmadi, S. Sorayassa, and M. Ahmadi, "A memory based approach for digital implementation of tanh using LUT and RALUT," *Comput. Sci. Inf. Technol.*, vol. 12, no. 22, Dec. 2022, Art. no. 22, doi: 10.5121/csit.2022.122204.

[414] F. de Dinechin and A. Tisserand, "Multipartite table methods," *IEEE Trans. Comput.*, vol. 54, no. 3, pp. 319–330, Mar. 2005, doi: 10.1109/TC.2005.54.

[415] A. Raha and V. Raghunathan, "Q LUT: Input-aware quantized table lookup for energy-efficient approximate accelerators," *ACM Trans. Embedded Comput. Syst.*, vol. 16, no. 5s, pp. 1–23, Sep. 2017, doi: 10.1145/3126531.

[416] Y. Xie, A. N. Joseph Raj, Z. Hu, S. Huang, Z. Fan, and M. Joler, "A twofold lookup table architecture for efficient approximation of activation functions," *IEEE Trans. Very Large Scale Integr. (VLSI) Syst.*, vol. 28, no. 12, pp. 2540–2550, Dec. 2020, doi: 10.1109/TVLSI.2020.3015391.

[417] Z. Hajduk and G. R. Dec, "Very high accuracy hyperbolic tangent function implementation in FPGAs," *IEEE Access*, vol. 11, pp. 23701–23713, 2023, doi: 10.1109/ACCESS.2023.3253668.

[418] R. Yousif, I. Hashim, and B. Abd, "Implementation of hyperbolic sine and cosine functions based on FPGA using different approaches," *Eng. Technol. J.*, vol. 41, no. 8, pp. 1–16, Aug. 2023, doi: 10.30684/etj.2023.139756.1440.

[419] T.-K. Luong, V.-T. Nguyen, A.-T. Nguyen, and E. Popovici, "Efficient architectures and implementation of arithmetic functions approximation based stochastic computing," in *Proc. IEEE 30th Int. Conf. Appl.-Specific Syst., Archit. Processors (ASAP)*, Jul. 2019, pp. 281–287, doi: 10.1109/ASAP.2019.00018.

[420] M. Osta, A. Ibrahim, and M. Valle, "FPGA implementation of approximate CORDIC circuits for energy efficient applications," in *Proc. 26th IEEE Int. Conf. Electron., Circuits Syst. (ICECS)*, Nov. 2019, pp. 127–128, doi: 10.1109/ICECS46596.2019.8964758.

[421] A. Changela, Y. Kumar, M. Woźniak, J. Shafi, and M. F. Ijaz, "Radix-4 CORDIC algorithm based low-latency and hardware efficient VLSI architecture for Nth root and Nth power computations," *Sci. Rep.*, vol. 13, no. 1, p. 20918, Nov. 2023, doi: 10.1038/s41598-023-47890-3.

[422] I. Scarabottolo, G. Ansaloni, G. A. Constantinides, L. Pozzi, and S. Reda, "Approximate logic synthesis: A survey," *Proc. IEEE*, vol. 108, no. 12, pp. 2195–2213, Dec. 2020, doi: 10.1109/JPROC.2020.3014430.

[423] G. Liu and Z. Zhang, "Statistically certified approximate logic synthesis," in *Proc. IEEE/ACM Int. Conf. Comput.-Aided Design (ICCAD)*, Nov. 2017, pp. 344–351, doi: 10.1109/ICCAD.2017.8203798.

[424] W. Zeng, A. Davoodi, and R. O. Topaloglu, "Sampling-based approximate logic synthesis: An explainable machine learning approach," in *Proc. IEEE/ACM Int. Conf. Comput. Aided Design (ICCAD)*, vol. 30. Piscataway, NJ, USA: IEEE Press, Nov. 2021, pp. 1–9, doi: 10.1109/ICCAD51958.2021.9643484.

[425] I. Scarabottolo, G. Ansaloni, and L. Pozzi, "Circuit carving: A methodology for the design of approximate hardware," in *Proc. Design, Autom. Test Eur. Conf. Exhib. (DATE)*, Mar. 2018, pp. 545–550, doi: 10.23919/DATE.2018.8342067.

[426] J. Castro-Godínez, H. Barrantes-García, M. Shafique, and J. Henkel, "AxLS: A framework for approximate logic synthesis based on netlist transformations," *IEEE Trans. Circuits Syst. II, Exp. Briefs*, vol. 68, no. 8, pp. 2845–2849, Aug. 2021, doi: 10.1109/TCSII.2021.3068757.

[427] L. Witschen, T. Wiersema, M. Artmann, and M. Platzner, "MUSCAT: MUS-based circuit approximation technique," in *Proc. Design, Autom. Test Eur. Conf. Exhib. (DATE)*, Mar. 2022, pp. 172–177, doi: 10.23919/DATE54114.2022.9774604.

[428] A. Chandrasekharan, M. Soeken, D. Große, and R. Drechsler, "Approximation-aware rewriting of AIGs for error tolerant applications," in *Proc. IEEE/ACM Int. Conf. Comput.-Aided Design (ICCAD)*, Nov. 2016, pp. 1–8, doi: 10.1145/2966986.2967003.

[429] V. Mrazek, R. Hrbacek, Z. Vasicek, and L. Sekanina, "EvoApprox8b: Library of approximate adders and multipliers for circuit design and benchmarking of approximation methods," in *Proc. Design, Autom. Test Eur. Conf. Exhib. (DATE)*, Mar. 2017, pp. 258–261, doi: 10.23919/DATE.2017.7926993.

[430] S. Hashemi, H. Tann, and S. Reda, "BLASYS: Approximate logic synthesis using Boolean matrix factorization," in *Proc. 55th ACM/ESDA/IEEE Design Autom. Conf. (DAC)*, Jun. 2018, pp. 1–6, doi: 10.1109/DAC.2018.8465702.

[431] M. Rezaalipour, M. Biasion, I. Scarabottolo, G. A. Constantinides, and L. Pozzi, "A parametrizable template for approximate logic synthesis," in *Proc. 53rd Annu. IEEE/IFIP Int. Conf. Dependable Syst. Netw. Workshops (DSN-W)*, Jun. 2023, pp. 175–178, doi: 10.1109/dsn-w58399.2023.00049.

[432] G. Ammes, W. L. Neto, P. Butzen, P.-E. Gaillardon, and R. P. Ribas, "A two-level approximate logic synthesis combining cube insertion and removal," *IEEE Trans. Comput.-Aided Design Integr. Circuits Syst.*, vol. 41, no. 11, pp. 5126–5130, Nov. 2022, doi: 10.1109/TCAD.2022.3143489.

[433] Y. Wu and W. Qian, "ALFANS: Multilevel approximate logic synthesis framework by approximate node simplification," *IEEE Trans. Comput.-Aided Design Integr. Circuits Syst.*, vol. 39, no. 7, pp. 1470–1483, Jul. 2020, doi: 10.1109/TCAD.2019.2915328.

[434] M. Barbareschi, S. Barone, N. Mazzocca, and A. Moriconi, "A catalog-based AIG-rewriting approach to the design of approximate components," *IEEE Trans. Emerg. Topics Comput.*, vol. 11, no. 1, pp. 70–81, Jan. 2023, doi: 10.1109/TETC.2022.3170502.

[435] C. Meng, W. Qian, and A. Mishchenko, "ALSRAC: Approximate logic synthesis by resubstitution with approximate care set," in *Proc. 57th ACM/IEEE Design Autom. Conf. (DAC)*, Jul. 2020, pp. 1–6, doi: 10.1109/DAC18072.2020.9218627.

[436] K. Nepal, S. Hashemi, H. Tann, R. I. Bahar, and S. Reda, "Automated high-level generation of low-power approximate computing circuits," *IEEE Trans. Emerg. Topics Comput.*, vol. 7, no. 1, pp. 18–30, Jan. 2019, doi: 10.1109/TETC.2016.2598283.

[437] M. T. Leipnitz and G. L. Nazar, "Constraint-aware multi-technique approximate high-level synthesis for FPGAs," *ACM Trans. Reconfigurable Technol. Syst.*, vol. 16, no. 4, pp. 1–28, Oct. 2023, doi: 10.1145/3624481.

[438] J. Castro-Godínez, J. Mateus-Vargas, M. Shafique, and J. Henkel, "AxHLS: Design space exploration and high-level synthesis of approximate accelerators using approximate functional units and analytical models," in *Proc. IEEE/ACM Int. Conf. Comput. Aided Design (ICCAD)*. New York, NY, USA: ACM, Nov. 2020, pp. 1–9.

[439] R. Ranjan, S. Ullah, S. S. Sahoo, and A. Kumar, ''SyFAxO-GeN: Synthesizing FPGA-based approximate operators with generative networks,'' in *Proc. 28th Asia South Pacific Design Autom. Conf. (ASP-DAC)*. New York, NY, USA: ACM, Jan. 2023, pp. 402–409, doi: 10.1145/3566097.3567891.

[440] Y. Wu, C. Shen, Y. Jia, and W. Qian, ''Approximate logic synthesis for FPGA by wire removal and local function change,'' in *Proc. 22nd Asia South Pacific Design Autom. Conf. (ASP-DAC)*, Jan. 2017, pp. 163–169, doi: 10.1109/ASPDAC.2017.7858314.

[441] G. Pasandi, S. Pratty, and J. Forsyth, ''AISYN: AI-driven reinforcement learning-based logic synthesis framework,'' 2023, *arXiv:2302.06415*.

[442] G. Pasandi, M. Peterson, M. Herrera, S. Nazarian, and M. Pedram, ''Deep-PowerX: A deep learning-based framework for low-power approximate logic synthesis,'' in *Proc. ACM/IEEE Int. Symp. Low Power Electron. Design*. New York, NY, USA: ACM, Aug. 2020, pp. 73–78, doi: 10.1145/3370748.3406555.

[443] C.-T. Lee, Y.-T. Li, Y.-C. Chen, and C.-Y. Wang, ''Approximate logic synthesis by genetic algorithm with an error rate guarantee,'' in *Proc. 28th Asia South Pacific Design Autom. Conf. (ASP-DAC)*. New York, NY, USA: ACM, Jan. 2023, pp. 146–151, doi: 10.1145/3566097.3567890.

[444] R. Kalkreuth, Z. Vašíč ek, J. Husa, D. Vermetten, F. Ye, and T. Bäck, ''Towards a general Boolean function benchmark suite,'' in *Proc. Companion Conf. Genetic Evol. Comput.*, vol. 2. New York, NY, USA: ACM, Jul. 2023, pp. 591–594, doi: 10.1145/3583133.3590685.

[445] L. Sekanina, Z. Vasicek, and V. Mrazek, ''Automated search-based functional approximation for digital circuits,'' in *Approximate Circuits: Methodologies and CAD*, S. Reda and M. Shafique, Eds., Cham, Switzerland: Springer, 2019, pp. 175–203, doi: 10.1007/978-3-319-99322-5_9.

[446] Z. Vasicek and L. Sekanina, ''Evolutionary approach to approximate digital circuits design,'' *IEEE Trans. Evol. Comput.*, vol. 19, no. 3, pp. 432–444, Jun. 2015, doi: 10.1109/TEVC.2014.2336175.

[447] S. Su, Y. Wu, and W. Qian, ''Efficient batch statistical error estimation for iterative multi-level approximate logic synthesis,'' in *Proc. 55th ACM/ESDA/IEEE Design Autom. Conf. (DAC)*, Jun. 2018, pp. 1–6, doi: 10.1109/DAC.2018.8465838.

[448] S. Su, C. Meng, F. Yang, X. Shen, L. Ni, W. Wu, Z. Wu, J. Zhao, and W. Qian, ''VECBEE: A versatile efficiency–accuracy configurable batch error estimation method for greedy approximate logic synthesis,'' *IEEE Trans. Comput.-Aided Design Integr. Circuits Syst.*, vol. 41, no. 11, pp. 5085–5099, Nov. 2022, doi: 10.1109/TCAD.2022.3149717.

[449] M. Rezaalipour, L. Ferretti, I. Scarabottolo, G. A. Constantinides, and L. Pozzi, ''Multi-metric SMT-based evaluation of worst-case-error for approximate circuits,'' in *Proc. 53rd Annu. IEEE/IFIP Int. Conf. Dependable Syst. Netw. Workshops (DSN-W)*, vol. 7, Jun. 2023, pp. 199–202, doi: 10.1109/dsn-w58399.2023.00055.

[450] Z. Vasicek, ''Formal methods for exact analysis of approximate circuits,'' *IEEE Access*, vol. 7, pp. 177309–177331, 2019, doi: 10.1109/ACCESS.2019.2958605.

[451] G. Ammes, P. F. Butzen, A. I. Reis, and R. Ribas, ''Two-level and multilevel approximate logic synthesis,'' *J. Integr. Circuits Syst.*, vol. 17, no. 3, pp. 1–14, Dec. 2022, doi: 10.29292/jics.v17i3.661.

[452] P. Choudhary, L. Bhargava, V. Singh, and A. Kumar Suhag, ''Approximate computing: Evolutionary methods for functional approximation of digital circuits,'' *Mater. Today, Proc.*, vol. 66, pp. 3487–3492, Jan. 2022, doi: 10.1016/j.matpr.2022.06.386.

[453] A. Raha and V. Raghunathan, ''Approximating beyond the processor: Exploring full-system energy-accuracy tradeoffs in a smart camera system,'' *IEEE Trans. Very Large Scale Integr. (VLSI) Syst.*, vol. 26, no. 12, pp. 2884–2897, Dec. 2018, doi: 10.1109/TVLSI.2018.2864269.

[454] S. Hashemi, H. Tann, F. Buttafuoco, and S. Reda, ''Approximate computing for biometric security systems: A case study on iris scanning,'' in *Proc. Design, Autom. Test Eur. Conf. Exhib. (DATE)*, Mar. 2018, pp. 319–324, doi: 10.23919/DATE.2018.8342029.

[455] B. Srinivas Prabakaran, V. Mrazek, Z. Vasicek, L. Sekanina, and M. Shafique, ''Xel-FPGAs: An end-to-end automated exploration framework for approximate accelerators in FPGA-based systems,'' 2023, *arXiv:2303.04734*.

[456] Y. Umuroglu, N. J. Fraser, G. Gambardella, M. Blott, P. Leong, M. Jahre, and K. Vissers, ''FINN: A framework for fast, scalable binarized neural network inference,'' in *Proc. ACM/SIGDA Int. Symp. Field-Program. Gate Arrays*. New York, NY, USA: ACM, Feb. 2017, pp. 65–74, doi: 10.1145/3020078.3021744.

[457] N. R. Shanbhag, N. Verma, Y. Kim, A. D. Patil, and L. R. Varshney, ''Shannon-inspired statistical computing for the nanoscale era,'' *Proc. IEEE*, vol. 107, no. 1, pp. 90–107, Jan. 2019, doi: 10.1109/JPROC.2018.2869867.

[458] H. Kim and N. R. Shanbhag, ''Enhancing the accuracy of 6T SRAM-based in-memory architecture via maximum likelihood detection,'' *IEEE Trans. Signal Process.*, vol. 72, pp. 2799–2811, 2024, doi: 10.1109/tsp.2024.3394656.

[459] A. Pantazi, B. Rajendran, O. Simeone, and E. Neftci, ''Editorial: Neuro-inspired computing for next-gen AI: Computing model, architectures and learning algorithms,'' *Frontiers Neurosci.*, vol. 16, Jul. 2022, Art. no. 974627, doi: 10.3389/fnins.2022.974627.

[460] A. Mehonic and A. J. Kenyon, ''Brain-inspired computing needs a master plan,'' *Nature*, vol. 604, no. 7905, pp. 255–260, Apr. 2022, doi: 10.1038/s41586-021-04362-w.

[461] S. Yu, ''Neuro-inspired computing with emerging nonvolatile memorys,'' *Proc. IEEE*, vol. 106, no. 2, pp. 260–285, Feb. 2018, doi: 10.1109/JPROC.2018.2790840.

[462] S. Sen, S. Venkataramani, and A. Raghunathan, ''Approximate computing for spiking neural networks,'' in *Proc. Design, Autom. Test Eur. Conf. Exhib. (DATE)*, Mar. 2017, pp. 193–198, doi: 10.23919/DATE.2017.7926981.

[463] A. R. Nasser, A. M. Hasan, A. J. Humaidi, A. Alkhayyat, L. Alzubaidi, M. A. Fadhel, J. Santamaría, and Y. Duan, ''IoT and cloud computing in health-care: A new wearable device and cloud-based deep learning algorithm for monitoring of diabetes,'' *Electronics*, vol. 10, no. 21, p. 2719, Nov. 2021, doi: 10.3390/electronics10212719.

[464] S. K. Ghosh, A. Raha, and V. Raghunathan, ''Energy-efficient approximate edge inference systems,'' *ACM Trans. Embedded Comput. Syst.*, vol. 22, no. 4, pp. 1–50, Jul. 2023, doi: 10.1145/3589766.

[465] M. Fabjanč ič, O. Machidon, H. Sharif, Y. Zhao, S. Misailović, and V. Pejović, ''Mobiprox: Supporting dynamic approximate computing on mobiles,'' 2023, *arXiv:2303.11291*.

[466] V. Pejović, ''Towards approximate mobile computing,'' *GetMobile, Mobile Comput. Commun.*, vol. 22, no. 4, pp. 9–12, May 2019, doi: 10.1145/3325867.3325871.

[467] W. B. Qaim, A. Ometov, A. Molinaro, I. Lener, C. Campolo, E. S. Lohan, and J. Nurmi, ''Towards energy efficiency in the Internet of Wearable Things: A systematic review,'' *IEEE Access*, vol. 8, pp. 175412–175435, 2020, doi: 10.1109/ACCESS.2020.3025270.

[468] A. Das, S. K. Ghosh, A. Raha, and V. Raghunathan, ''Toward energy-efficient collaborative inference using multisystem approximations,'' *IEEE Internet Things J.*, vol. 11, no. 10, pp. 17989–18004, May 2024, doi: 10.1109/JIOT.2024.3365306.

[469] S. Jain, S. Venkataramani, V. Srinivasan, J. Choi, P. Chuang, and L. Chang, ''Compensated-DNN: Energy efficient low-precision deep neural networks by compensating quantization errors,'' in *Proc. 55th ACM/ESDA/IEEE Design Autom. Conf. (DAC)*, Jun. 2018, pp. 1–6, doi: 10.1109/DAC.2018.8465893.

[470] S. S. Sarwar, S. Venkataramani, A. Ankit, A. Raghunathan, and K. Roy, ''Energy-efficient neural computing with approximate multipliers,'' *ACM J. Emerg. Technol. Comput. Syst.*, vol. 14, no. 2, pp. 1–23, Jul. 2018, doi: 10.1145/3097264.

[471] Z. Peng, X. Chen, C. Xu, N. Jing, X. Liang, C. Lu, and L. Jiang, ''AXNet: ApproXimate computing using an end-to-end trainable neural network,'' in *Proc. IEEE/ACM Int. Conf. Comput.-Aided Design (ICCAD)*. New York, NY, USA: ACM, Nov. 2018, pp. 1–8.

[472] N. Ashar, G. Raut, V. Trivedi, S. K. Vishvakarma, and A. Kumar, ''QuantMAC: Enhancing hardware performance in DNNs with quantize enabled multiply-accumulate unit,'' *IEEE Access*, vol. 12, pp. 43600–43614, 2024, doi: 10.1109/ACCESS.2024.3379906.

[473] X. Sui, Q. Lv, Y. Bai, B. Zhu, L. Zhi, Y. Yang, and Z. Tan, ''A hardware-friendly low-bit power-of-two quantization method for CNNs and its FPGA implementation,'' *Sensors*, vol. 22, no. 17, p. 6618, Sep. 2022, doi: 10.3390/s22176618.

[474] X. Sui, Q. Lv, L. Zhi, B. Zhu, Y. Yang, Y. Zhang, and Z. Tan, "A hardware-friendly high-precision CNN pruning method and its FPGA implementation," *Sensors*, vol. 23, no. 2, p. 824, Jan. 2023, doi: 10.3390/s23020824.

[475] M. P. Véstias, R. P. Duarte, J. T. de Sousa, and H. C. Neto, "Fast convolutional neural networks in low density FPGAs using zero-skipping and weight pruning," *Electronics*, vol. 8, no. 11, p. 1321, Nov. 2019, doi: 10.3390/electronics8111321.

[476] S. Jang, W. Liu, and Y. Cho, "Convolutional neural network model compression method for software—Hardware co-design," *Information*, vol. 13, no. 10, p. 451, Sep. 2022, doi: 10.3390/info13100451.

[477] M. Zhang, L. Li, H. Wang, Y. Liu, H. Qin, and W. Zhao, "Optimized compression for implementing convolutional neural networks on FPGA," *Electronics*, vol. 8, no. 3, p. 295, Mar. 2019, doi: 10.3390/electronics8030295.

[478] J. Bai, S. Sun, W. Zhao, and W. Kang, "CIMQ: A hardware-efficient quantization framework for computing-in-memory-based neural network accelerators," *IEEE Trans. Comput.-Aided Design Integr. Circuits Syst.*, vol. 43, no. 1, pp. 189–202, Jan. 2024, doi: 10.1109/tcad.2023.3298705.

[479] M. Hafezan and E. Atoofian, "Mixed-precision architecture for GPU tensor cores," in *Proc. IEEE Smart World Congr. (SWC)*, Aug. 2023, pp. 1–8, doi: 10.1109/swc57546.2023.10448789.

[480] S. Tabrizchi, A. Nezhadi, S. Angizi, and A. Roohi, "AppCiP: Energy-efficient approximate convolution-in-pixel scheme for neural network acceleration," *IEEE J. Emerg. Sel. Topics Circuits Syst.*, vol. 13, no. 1, pp. 225–236, Mar. 2023, doi: 10.1109/JETCAS.2023.3242167.

[481] S. A. K. Gharavi and S. Safari, "Performance improvement of processor through configurable approximate arithmetic units in multicore systems," *IEEE Access*, vol. 12, pp. 43907–43917, 2024, doi: 10.1109/ACCESS.2024.3380912.

[482] H. Younes, A. Ibrahim, M. Rizk, and M. Valle, "Algorithmic-level approximate tensorial SVM using high-level synthesis on FPGA," *Electronics*, vol. 10, no. 2, p. 205, Jan. 2021, doi: 10.3390/electronics10020205.

[483] I. D. Mienye and N. Jere, "A survey of decision trees: Concepts, algorithms, and applications," *IEEE Access*, vol. 12, pp. 86716–86727, 2024, doi: 10.1109/ACCESS.2024.3416838.

[484] K. K. Pandey and D. Shukla, "Stratification to improve systematic sampling for big data mining using approximate clustering," in *Machine Intelligence and Smart Systems*, S. Agrawal, K. Kumar Gupta, J. H. Chan, J. Agrawal, and M. Gupta, Eds., Singapore: Springer, 2021, pp. 337–351, doi: 10.1007/978-981-33-4893-6_30.

[485] J. Liu, W. Yinchai, F. Wei, Q. Han, Y. Tao, L. Zhao, X. Li, and H. Sun, "Secure cloud-aided approximate nearest neighbor search on high-dimensional data," *IEEE Access*, vol. 11, pp. 109027–109037, 2023, doi: 10.1109/ACCESS.2023.3321457.

[486] S. Khan, S. Singh, H. V. Simhadri, and J. Vedurada, "BANG: Billion-scale approximate nearest neighbor search using a single GPU," 2024, *arXiv:2401.11324*.

[487] D. Vanderkam, R. Schonberger, H. Rowley, and S. Kumar, "Nearest neighbor search in Google correlate," Google, Inc., Mountain View, CA, USA, Tech. Rep., 2013. [Online]. Available: http://research.google.com/pubs/archive/41694.pdf

[488] F. Regazzoni, C. Alippi, and I. Polian, "Security: The dark side of approximate computing?" in *Proc. IEEE/ACM Int. Conf. Comput.-Aided Design (ICCAD)*, Nov. 2018, pp. 1–6, doi: 10.1145/3240765.3243497.

[489] P. Yellu, L. Buell, M. Mark, M. A. Kinsy, D. Xu, and Q. Yu, "Security threat analyses and attack models for approximate computing systems: From hardware and micro-architecture perspectives," *ACM Trans. Design Autom. Electron. Syst.*, vol. 26, no. 4, pp. 1–31, Apr. 2021, doi: 10.1145/3442380.

[490] P. Yellu and Q. Yu, "Securing approximate computing systems via obfuscating approximate-precise boundary," *IEEE Trans. Comput.-Aided Design Integr. Circuits Syst.*, vol. 42, no. 1, pp. 27–40, Jan. 2023, doi: 10.1109/TCAD.2022.3168261.

[491] S. Ariful Islam, "On the (In)security of approximate computing synthesis," 2019, *arXiv:1912.01209*.

[492] D.-E.-S. Kundi, A. Khalid, S. Bian, C. Wang, M. O'Neill, and W. Liu, "AxRLWE: A multilevel approximate ring-LWE co-processor for lightweight IoT applications," *IEEE Internet Things J.*, vol. 9, no. 13, pp. 10492–10501, Jul. 2022, doi: 10.1109/JIOT.2021.3122276.

[493] B. Miller and R. Pozo. *Java SciMark 2.0*. Accessed: Mar. 13, 2024. [Online]. Available: https://math.nist.gov/scimark2/

[494] S. Che, M. Boyer, J. Meng, D. Tarjan, J. W. Sheaffer, S.-H. Lee, and K. Skadron, "Rodinia: A benchmark suite for heterogeneous computing," in *Proc. IEEE Int. Symp. Workload Characterization (IISWC)*, Oct. 2009, pp. 44–54, doi: 10.1109/IISWC.2009.5306797.

[495] A. Yazdanbakhsh, D. Mahajan, H. Esmaeilzadeh, and P. Lotfi-Kamran, "AxBench: A multiplatform benchmark suite for approximate computing," *IEEE Des. Test. IEEE Des. Test. Comput.*, vol. 34, no. 2, pp. 60–68, Apr. 2017, doi: 10.1109/MDAT.2016.2630270.

[496] S. Ullah, S. S. Murthy, and A. Kumar, "SMApproxLib: Library of FPGA-based approximate multipliers," in *Proc. 55th ACM/ESDA/IEEE Design Autom. Conf. (DAC)*, Jun. 2018, pp. 1–6, doi: 10.1109/DAC.2018.8465845.

[497] L. Witschen, M. Awais, H. Ghasemzadeh Mohammadi, T. Wiersema, and M. Platzner, "CIRCA: Towards a modular and extensible framework for approximate circuit generation," *Microelectron. Rel.*, vol. 99, pp. 277–290, Aug. 2019, doi: 10.1016/j.microrel.2019.04.003.

[498] M. V. Bordin, D. Griebler, G. Mencagli, C. F. R. Geyer, and L. G. L. Fernandes, "DSPBench: A suite of benchmark applications for distributed data stream processing systems," *IEEE Access*, vol. 8, pp. 222900–222917, 2020, doi: 10.1109/ACCESS.2020.3043948.

[499] M. Bakhshalipour, M. Likhachev, and P. B. Gibbons, "RTRBench: A benchmark suite for real-time robotics," in *Proc. IEEE Int. Symp. Perform. Anal. Syst. Softw. (ISPASS)*, May 2022, pp. 175–186, doi: 10.1109/ISPASS55109.2022.00024.

[500] H. C. Prashanth and M. Rao, "SOMALib: Library of exact and approximate activation functions for hardware-efficient neural network accelerators," in *Proc. IEEE 40th Int. Conf. Comput. Design (ICCD)*, Oct. 2022, pp. 746–753, doi: 10.1109/ICCD56317.2022.00114.

[501] M. Item, J. Gómez-Luna, Y. Guo, G. F. Oliveira, M. Sadrosadati, and O. Mutlu, "TransPimLib: A library for efficient transcendental functions on processing-in-memory systems," 2023, *arXiv:2304.01951*.

[502] *OpenBenchmarking.org—Cross-Platform, Open-Source Automated Benchmarking Platform*. Accessed: Mar. 13, 2024. [Online]. Available: https://openbenchmarking.org/

[503] *Papers With Code—The Latest in Machine Learning*. Accessed: Mar. 13, 2024. [Online]. Available: https://paperswithcode.com/

[504] Y. Guan, H. Liang, N. Xu, W. Wang, S. Shi, X. Chen, G. Sun, W. Zhang, and J. Cong, "FP-DNN: An automated framework for mapping deep neural networks onto FPGAs with RTL-HLS hybrid templates," in *Proc. IEEE 25th Annu. Int. Symp. Field-Program. Custom Comput. Mach. (FCCM)*, Apr. 2017, pp. 152–159, doi: 10.1109/FCCM.2017.25.

[505] Y. Ma, Y. Cao, S. Vrudhula, and J.-S. Seo, "An automatic RTL compiler for high-throughput FPGA implementation of diverse deep convolutional neural networks," in *Proc. 27th Int. Conf. Field Program. Log. Appl. (FPL)*, Sep. 2017, pp. 1–8, doi: 10.23919/FPL.2017.8056824.

[506] *FINN*. Accessed: May 15, 2024. [Online]. Available: https://xilinx.github.io/finn/

[507] M. T. Arafin and Z. Lu, "Security challenges of processing-in-memory systems," in *Proc. Great Lakes Symp. VLSI*, vol. 99. New York, NY, USA: ACM, Sep. 2020, pp. 229–234, doi: 10.1145/3386263.3411365.

[508] S. Lee and A. Gerstlauer, "Approximate high-level synthesis of custom hardware," in *Approximate Circuits: Methodologies and CAD*, S. Reda and M. Shafique, Eds., Cham, Switzerland: Springer, 2019, pp. 205–223, doi: 10.1007/978-3-319-99322-5_10.

[509] T. Alan, A. Gerstlauer, and J. Henkel, "Runtime accuracy-configurable approximate hardware synthesis using logic gating and relaxation," in *Proc. Design, Autom. Test Eur. Conf. Exhib. (DATE)*, Mar. 2020, pp. 1578–1581, doi: 10.23919/DATE48585.2020.9116272.

[510] R. Zhao, Z. Yang, H. Zheng, Y. Wu, F. Liu, Z. Wu, L. Li, F. Chen, S. Song, J. Zhu, W. Zhang, H. Huang, M. Xu, K. Sheng, Q. Yin, J. Pei, G. Li, Y. Zhang, M. Zhao, and L. Shi, "A framework for the general design and computation of hybrid neural networks," *Nature Commun.*, vol. 13, no. 1, p. 3427, Jun. 2022, doi: 10.1038/s41467-022-30964-7.

AYAD M. DALLOO received the B.Sc. and M.Sc. degrees in electronic and communication engineering. He is currently pursuing the Ph.D. degree with the Electrical Engineering Department, University of Technology, Iraq. He is also a Faculty Member of the Communication Engineering Department, University of Technology. His research interests include approximate computing and machine learning.

AMJAD JALEEL HUMAIDI received the B.Sc. and M.Sc. degrees in control engineering from the Al-Rasheed College of Engineering and Science, in 1992 and 1997, respectively, and the Ph.D. degree, in 2006, with specialization in control and automation. He is currently a Professor with the Engineering College, University of Technology, Iraq. His research interests include adaptive, nonlinear and intelligent control, optimization, and real-time image processing.

AMMAR K. AL MHDAWI received the Ph.D. degree in electronic and electrical engineering from Brunel University London. He is currently a Lecturer with the School of Engineering and Sustainable Development, De Montfort University, U.K. His postdoctoral research from Newcastle University, U.K. He is also a Freelance Consultant Engineer with more than 15 years of experience in control engineering and robotics. His research interests include control systems, (AUV, AGV, and UAV) robotic systems, intelligent and automatic control, and interconnected systems. Moreover, he is a Guest Editor of *Actuators* (MDPI) special session.

HAMED AL-RAWESHIDY (Senior Member, IEEE) is currently a renowned Professor in communications engineering with the University of Technology, Baghdad, and advanced qualifications with the University of Glasgow, U.K., and Strathclyde University, U.K. He has a rich career, including roles with the Space and Astronomy Research Centre, Iraq; PerkinElmer, USA; Carl Zeiss, Germany; British Telecom, U.K.; and various universities, such as Oxford University, Manchester Metropolitan University, and Kent University. Currently, he directs the Wireless Networks and Communications Centre and Postgraduate Studies in Electronic and Computer Engineering, Brunel University London. He has published over 370 articles and edited the first book *Radio Over Fiber Technologies for Mobile Communications Networks*. He is also a Consultant for global telecom companies and a Principal Investigator for significant research projects. His current research interests include advanced technologies in communications engineering, including 5G and 6G developments, quantum computing, AI, and the IoT applications.