A Survey on Design Space Exploration Approaches for Approximate Computing Systems

electronics

MDPI

Review

A Survey on Design Space Exploration Approaches for Approximate Computing Systems

Sepide Saeedi [1,*], Ali Piri [2], Bastien Deveautour [3], Ian O'Connor [2], Alberto Bosio [2], Alessandro Savino [1] and Stefano Di Carlo [1]

[1] Department of Control and Computer Engineering, Politecnico di Torino, 10138 Turin, Italy; alessandro.savino@polito.it (A.S.); stefano.dicarlo@polito.it (S.D.C.)
[2] ECL, INSA Lyon, CNRS, UCBL, CPE Lyon, INL, UMR5270, 69130 Ecully, France; ali.piri@ec-lyon.fr (A.P.); ian.oconnor@ec-lyon.fr (I.O.); alberto.bosio@ec-lyon.fr (A.B.)
[3] CNRS, IETR, UMR6164, Nantes Université, 35042 Nantes, France; bastien.deveautour@univ-nantes.fr
* Correspondence: sepide.saeedi@polito.it

Abstract: Approximate Computing (AxC) has emerged as a promising paradigm to enhance performance and energy efficiency by allowing a controlled trade-off between accuracy and resource consumption. It is extensively adopted across various abstraction levels, from software to architecture and circuit levels, employing diverse methodologies. The primary objective of AxC is to reduce energy consumption for executing error-resilient applications, accepting controlled and inherently acceptable output quality degradation. However, harnessing AxC poses several challenges, including identifying segments within a design amenable to approximation and selecting suitable AxC techniques to fulfill accuracy and performance criteria. This survey provides a comprehensive review of recent methodologies proposed for performing Design Space Exploration (DSE) to find the most suitable AxC techniques, focusing on both hardware and software implementations. DSE is a crucial design process where system designs are modeled, evaluated, and optimized for various extra-functional system behaviors such as performance, power consumption, energy efficiency, and accuracy. A systematic literature review was conducted to identify papers that ascribe their DSE algorithms, excluding those relying on exhaustive search methods. This survey aims to detail the state-of-the-art DSE methodologies that efficiently select AxC techniques, offering insights into their applicability across different hardware platforms and use-case domains. For this purpose, papers were categorized based on the type of search algorithm used, with Machine Learning (ML) and Evolutionary Algorithms (EAs) being the predominant approaches. Further categorization is based on the target hardware, including Field-Programmable Gate Arrays (FPGAs), Application-Specific Integrated Circuits (ASICs), general-purpose Central Processing Units (CPUs), and Graphics Processing Units (GPUs). A notable observation was that most studies targeted image processing applications due to their tolerance for accuracy loss. By providing an overview of techniques and methods outlined in existing literature pertaining to the DSE of AxC designs, this survey elucidates the current trends and challenges in optimizing approximate designs.

Keywords: approximatecomputing; design space exploration; circuit design; high-level synthesis; multi-objective optimization

Citation: Saeedi, S.; Piri, A.; Deveautour, B.; O'Connor, I.; Bosio, A.; Savino, A.; Di Carlo, S. A Survey on Design Space Exploration Approaches for Approximate Computing Systems. *Electronics* **2024**, *13*, 4442. https://doi.org/10.3390/electronics13224442

Academic Editor: Arkaitz Zubiaga

Received: 2 October 2024
Revised: 28 October 2024
Accepted: 5 November 2024
Published: 13 November 2024

1. Introduction

As large-scale application domains like scientific computing, social media, and financial analytics continue to expand, the computational and storage requirements of modern systems have surpassed the available resources. In the upcoming decade, it is anticipated that the amount of data managed by global data centers will increase by fifty times, while the number of processors will only grow by a factor of ten [1]. This indicates that the demand for performance will soon outstrip resource allocations.

Furthermore, Information and Communication Technology (ICT) devices and services currently contribute significantly to the world's overall energy consumption, with projections indicating that their energy demand will rise to nearly 21% by 2030 [2]. Consequently, it becomes evident that relying solely on over-provisioning resources will not suffice to address the impending challenges facing the computing industry. These constraints on both computational resources and energy consumption create a growing urgency for new approaches to improve efficiency while maintaining acceptable levels of performance.

In recent decades, significant technological advancements and increasing computational demands have driven a remarkable reduction in the size of integrated circuits and computing systems. This downscaling of CMOS technology has resulted in several key benefits, such as enhanced computational performance, improved energy efficiency, and the ability to increase the number of cores per chip. Smaller transistors allow for faster switching speeds, enabling higher clock frequencies, which translates to quicker data processing and more powerful computing systems. Additionally, as transistors shrink, the power required to switch them can be reduced, leading to lower overall energy consumption, which is crucial for mobile and battery-operated devices.

However, CMOS downscaling is not without its drawbacks. As transistors continue to shrink, the benefits of reduced supply voltage become less significant, and the leakage current (unwanted current that flows even when the transistor is off) becomes more pronounced, leading to higher static power consumption. Moreover, the exponential increase in power consumption due to higher clock frequencies has introduced thermal challenges, as more energy is dissipated as heat, which can damage the chip and reduce its lifespan. The combination of these factors means that the traditional benefits of CMOS scaling are diminishing, and the ability to further increase the number of cores per chip is constrained by power and thermal limits. Consequently, as CMOS technology reaches its scaling limits, it becomes imperative to explore alternative approaches, such as new materials, 3D stacking, or novel architectures, to continue improving computing efficiency without exacerbating these power and thermal issues [3].

In addition to the trends mentioned above, the nature of the tasks fueling the demand for computing has evolved across the computing spectrum, spanning from mobile devices to the cloud. Within data centers and the cloud, the impetus for computing stems from the necessity to efficiently manage, organize, search, and derive conclusions from vast datasets. In contrast, the predominant computing demand for mobile and embedded devices arises from the desire for more immersive media experiences and more natural, intelligent interactions with users and the surrounding environment. Although computational errors are generally undesirable, a common thread runs through this spectrum: these applications are not primarily concerned with computing precise numerical outputs. Instead, "correctness" is defined as generating results that are sufficiently accurate to deliver an acceptable user experience [4].

These applications inherently possess a resilience towards errors, meaning they can produce satisfactory outputs even when some of their computations are carried out in an approximate manner [5]. For instance, in search and recommendation systems, there is not always a single definitive or "golden" result; instead, multiple answers falling within a specific range are considered acceptable. Additionally, iterative applications processing extensive data sets may terminate convergence prematurely or employ heuristics [6]. In many Machine Learning (ML) applications, even if a golden result exists, the most advanced algorithms may not be able to achieve it. Consequently, users often have to settle for reasonably inaccurate but still adequate results. Furthermore, applications such as multimedia, wireless communication, speech recognition, and data mining exhibit a degree of error tolerance. Human perceptual limitations signify that such errors may not significantly affect image, audio, and video processing applications. Another example pertains to applications dealing with noisy input data (e.g., image and sensor data processing, and speech recognition). The noise in the input naturally leads to imprecise results, and approximations have a similar impact. In simpler terms, applications that can handle

noisy inputs also possess the capability to withstand approximations [7–9]. Finally, some applications utilize computational patterns like aggregation or iterative refinement, which can mitigate or compensate for the effects of approximations.

By intentionally introducing controlled approximations, Approximate Computing (AxC) leverages the inherent resilience of these applications to improve energy efficiency and performance while aligning well with the evolving demands of diverse application domains. AxC is an encouraging approach to enhance computing efficiency.

The concept of AxC encompasses a wide array of techniques that capitalize on the inherent error resilience of applications, ultimately leading to improved efficiency across all computing stack layers, ranging from the fundamental transistor-level design to software implementations. These techniques can have varying impacts on both the hardware and the output quality. AxC capitalizes on the existence of data and algorithms that can tolerate errors, as well as the limitations in the perception of end-users. It strategically balances accuracy against the potential for performance improvements or energy savings. In essence, it takes advantage of the gap between the level of accuracy that computer systems can provide and the level of accuracy required by the specific application or the end-users. This required accuracy is typically much lower than what the computer systems can deliver. The selective relaxation of accuracy allows for considerable gains in key parameters like power and performance, particularly within applications where exact correctness is secondary to operational efficiency.

Leveraging AxC involves addressing a few aspects and challenges. The first challenge is identifying the segments within the targeted software or hardware component that can be candidates for approximation. Identifying segments of code or data that can be approximated may necessitate a comprehensive understanding of the application on behalf of the designer.

The second challenge is implementing the AxC technique to introduce approximations. On the one hand, there is a limit to the accuracy degradation that can be introduced so the output remains acceptable. On the other hand, the level of accuracy degradation and the performance improvements or energy savings vary depending on the selected AxC technique. Hence, available AxC techniques should be evaluated and compared with find the most suitable AxC technique tailored for a target application or design.

The next challenge is choosing the suitable error measurement criteria, often tailored to the particular application, and executing the actual error assessment process to ensure that the output adheres to the predefined quality standards [5]. The error assessment usually involves simulating the precise and approximate versions of applications. However, alternative methods like Bayesian inference [10,11] or ML-based approaches [12] have been put forth in the scientific literature.

A Design Space Exploration (DSE) can be performed to address all the previously mentioned challenges. The goal of performing a DSE is to determine the most optimal approximate configurations from those generated by applying a given set of approximation techniques to the design. Hence, the DSE approaches can help systematically evaluate different approximate designs to choose the most suitable AxC techniques and, consequently, the best configurations for any given combination of AxC techniques. Early DSE approaches either combine multiple design objectives into a single-objective optimization problem or optimize a solitary parameter while keeping the remaining variables constant. More recent research, as seen in published works, has tackled circuit design issues by considering a Multi-objective Optimization Problem (MOP) to seek out Pareto-optimal approximate circuit configurations [13]. Regrettably, these approaches predominantly concentrated on simple systems, specifically arithmetic components like adders and multipliers, as they form the foundational components for more intricate designs [14].

While several surveys have comprehensively explored DSE methods across domains like embedded systems and general-purpose computing, they often overlook the distinct challenges and considerations imposed by AxC. In contrast, approximate designs introduce new dimensions in the DSE process, as they require balancing accuracy with efficiency

gains in energy and performance tailored to the error resilience of specific applications. This survey focuses on DSE methodologies uniquely suited to the AxC paradigm, where selecting the optimal trade-offs involves performance and resource efficiency and assessing acceptable error margins. By providing a dedicated review of DSE approaches applicable to approximate designs, this work tries to fill a critical gap, offering insights that are not addressed in existing surveys and thereby supporting the design of next-generation systems that meet stringent energy and performance demands.

This paper aims to cover different DSE approaches leveraged in comparing approximate versions of a target application or design. The structure of this paper is as follows: Firstly, Section 2 provides a background on AxC techniques and DSE approaches. Then, in Section 3, the search methodology to find related studies and categorize them is explained. In Section 4, DSE approaches to compare and choose suitable AxC techniques are reviewed and compared. Finally, a conclusion is provided in Section 5.

2. Background

AxC represents an emerging paradigm enabling the development of significantly energy-efficient computing systems, including diverse hardware accelerators designed for tasks like image filtering, video processing, and data mining. This approach leverages the inherent error resilience of many applications, allowing for a trade-off between accuracy and energy efficiency [15]. This trade-off can be accomplished through various means, spanning from transistor-level design to software implementations, each with distinct effects on hardware integrity and output quality.

The following sections provide an overview of how different AxC techniques can be classified, followed by a description of the DSE paradigm.

2.1. Classification of AxC Techniques

AxC techniques can be classified into three groups based on their implementation level: software, architecture, and hardware, with specific techniques applicable at multiple levels, as shown in Figure 1. For instance, memory read approximation can be achieved through pure software approaches or within memory control units. While Figure 1 highlights some commonly utilized approximation methods from the existing literature, it is essential to note that there exist numerous ways to approximate an application, and the precise definition of what constitutes an approximation is a subject of debate [9,16].

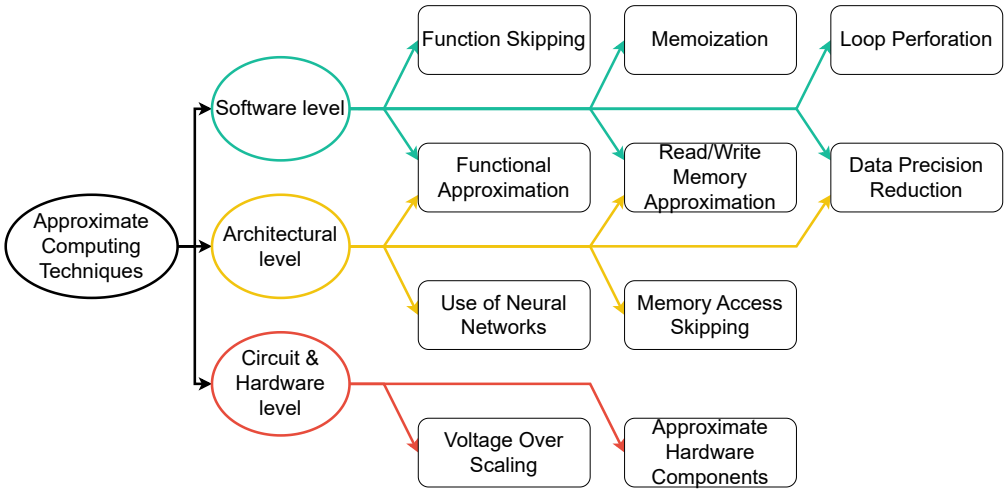

Figure 1. A classification of the AxC techniques.

2.1.1. Software-Level AxC Techniques

The loop-perforation technique serves as a notable example of software approximation, enabling the generation of valuable outcomes without executing every iteration of an iterative code [17]. Similarly, function-skipping involves bypassing specific code blocks

during runtime based on predefined conditions [18–20]. Another software approximation method involves reducing the bit width used for data representation, primarily impacting the memory footprint of the application. While reducing data precision can also affect the execution time and performance of the software, its impact relies on the hardware implementation of operations utilized by the application [21,22].

Memoization is a technique that optimizes performance and conserves energy by storing previously computed values for specific inputs. When the same input is re-encountered later, the stored value is reused, eliminating the need for redundant computation. This trade-off between computation and memory use enhances efficiency and reduces resource consumption [23].

Read/write memory approximation targets data loaded from or written to memory, as well as the memory access operations themselves. This method, commonly applied in video and image processing applications, relaxes accuracy requirements to minimize memory operations [24–26].

Functional approximation focuses on identifying components within algorithms that have a minimal impact on final accuracy. Energy consumption can be reduced by approximating these less critical components, leading to improved execution time performance [4,27].

2.1.2. Architectural-Level AxC Techniques

Neural Networks (NNs) can learn the behavior of a standard function implementation by analyzing how it responds to various inputs. Through software-hardware co-design, traditional code can be transformed into NNs with lower output accuracy but enhanced execution time and performance [28].

Memory access skipping combines memoization and function skipping techniques to omit uncritical memory accesses without significant accuracy loss. Approximate NNs leverage this approach to skip reading entire weight matrix rows for non-critical neurons, reducing energy consumption and improving performance [29].

2.1.3. Circuit and Hardware-Level AxC Techniques

Voltage-scaling techniques reduce energy consumption in digital circuits by adjusting the supply voltage at the circuit level. This adjustment directly impacts computation timing and power efficiency, exploiting the inherent error resilience of applications to achieve significant energy savings while maintaining acceptable performance levels [30]. Dynamic Voltage Scaling (DVS) adjusts the supply voltage to decrease power dissipation while maintaining computational accuracy within safe operational limits, balancing power savings against performance degradation [31]. Dynamic Voltage and Frequency Scaling (DVFS) extends this concept by simultaneously adjusting both the voltage and the operating frequency, providing finer control over power and performance trade-offs [30]. In contrast, Voltage Over Scaling (VOS) aggressively reduces the supply voltage beyond nominal levels, intentionally allowing timing violations that can introduce errors in computations [32,33]. While DVS and DVFS ensure accurate results within controlled performance constraints, VOS prioritizes further energy savings by permitting computational inaccuracies, making it suitable for scenarios where approximate results are acceptable.

Hardware-based approximation techniques often employ alternative implementations of arithmetic operators. For instance, variable approximation modes on operators represent one such approach. Hardware approximation also finds application in the image processing domain through approximate compressors [9,16].

Inexact hardware can provide approximation at the hardware level, with numerous examples from the literature of approximate arithmetic circuits designed to balance performance and accuracy. Adders, multipliers, and dividers significantly influence performance and energy efficiency across various computing tasks since they are the most frequent and vital arithmetic components within a processor. Pursuing enhanced speed, power efficiency, and error resilience in numerous applications such as multimedia processing,

recognition systems, and data analytics has propelled the advancement of approximate arithmetic design [3].

The aforementioned AxC techniques are a few examples of the many approximation techniques implemented at various abstraction levels. Due to this complexity, the challenges mentioned in the previous section need to be addressed, and DSE emerges as the most promising solution thus far.

2.2. Precision Metrics

Various precision metrics have been proposed to describe and evaluate the effectiveness of AxC techniques and methodologies, as detailed in Table 1.

Table 1. Precision metrics in AxC [34].

Metric	Description	Application Domain
ER	Error Rate—Erroneous results per total results	General Computing
EP	Error Probability—Probability of error occurrence	
MED/MRED	Mean (Relative) Error Distance	
NMED/NMRED	Normalized Mean (Relative) Error Distance	
Max ED/RED	Maximum (Relative) Error Distance	
PED/PRED	Probability of (Relative) Error Distance > X	
MSE	Mean Squared Error	
RMSE	Root Mean Squared Error	
HD	Hamming Distance	
BER	Bit Error Ratio—Bit errors per total received bits	Digital Systems, Telecommunications
PER	Packet Error Ratio—Incorrect packets per total received packets	
PSNR	Peak Signal-to-Noise Ratio—Quality measurement between images	Image Processing, Video Processing, Computer Vision
SSIM	Structural Similarity—Quality measurement between images	
MPD	Mean Pixel Difference	
Classif. Accuracy	Correct classifications per total classifications	Pattern Recognition, Information Retrieval, Machine Learning
Precision	Relevant instances per total retrieved instances	
Recall	Relevant instances per total relevant instances	

Error Distance (ED) and Error Probability (EP) are basic error metrics used for measuring accuracy degradation while applying AxC techniques. ED is the distance between the correct output and the approximate output of a circuit for each scenario, and EP is the probability of having a wrong answer, which is calculated by the number of wrong answers over the number of all input scenarios. These metrics are formulated in Equations (1) and (2), respectively.

$$\mathrm{ED} = |O_{AxC}^{(i)} - O_{Acc}^{(i)}| \tag{1}$$

$$\mathrm{EP} = \frac{\sum_{i=1}^{N} 1(O_{AxC}^{(i)} \neq O_{Acc}^{(i)})}{N} \tag{2}$$

where N is the number of possible scenarios, $O_{Acc}^{(i)}$ and $O_{AxC}^{(i)}$ are the accurate and approximate outputs of ith scenario, respectively, and $P(S_i)$ is the probability of the ith scenario happening.

Mean Error Distance (MED) is the average of all the error distances, which is calculated as Equation (3).

$$\text{MED} = \sum_{i=1}^{N} \frac{|O_{AxC}^{(i)} - O_{Acc}^{(i)}|}{N} \cdot P(S_i) \tag{3}$$

Mean Relative Error Distance (MRED) is the average of error distances relative to the correct answers formulated as Equation (4).

$$\text{MRED} = \sum_{i=1}^{N} \frac{|O_{AxC}^{(i)} - O_{Acc}^{(i)}|}{|O_{Acc}^{(i)}|} \cdot P(S_i) \tag{4}$$

Absolute Worst-Case Error (AWCE), which is defined as the largest error distance that can happen, is formulated in Equation (5).

$$\text{AWCE} = \max_{\forall i} |O_{AxC}^{(i)} - O_{Acc}^{(i)}| \tag{5}$$

Mean Squared Error (MSE) is the average of the squares of the errors between the approximate values and the accurate values. MSE is formulated as Equation (6).

$$\text{MSE} = \sum_{i=1}^{N} \frac{(O_{AxC}^{(i)} - O_{Acc}^{(i)})^2}{N} \cdot P(S_i) \tag{6}$$

Hamming Distance (HD) is a metric used to measure the difference between two strings of equal length. It is defined as the number of positions at which the corresponding symbols are different. For example, consider the binary strings "1011101" and "1001001". The HD is 2 because there are two positions where the strings differ (second and fourth positions).

Error Rate (ER) measures the number of erroneous results over the total number of results. In Equation (7), N_{err} represents the number of erroneous results, while N_{tot} represents the total number of results.

$$\text{ER} = \frac{N_{err}}{N_{tot}} \tag{7}$$

Bit Error Ratio (BER) is the number of bit errors per unit of time, calculated by dividing the number of bit errors by the total number of bits transferred during a given time interval. BER is a unitless performance measure and is often expressed as a percentage. It is a critical metric for evaluating the accuracy and reliability of data transmission in communication systems. In Equation (8), N_{bit_err} represents the number of erroneous bits, while the N_{bit_tot} represents the total number of bits.

$$\text{BER} = \frac{N_{bit\_err}}{N_{bit\_tot}} \tag{8}$$

Peak Signal-to-Noise Ratio (PSNR) is a metric that evaluates the quality of reconstructed images or videos by comparing them to their originals. It measures the ratio between the maximum possible signal power and the power of corrupting noise, expressed in decibels (dB). Higher PSNR values indicate better quality. PSNR is formulated as

Equation (9) where R is the maximum possible pixel value (e.g., 255 for an 8-bit image), and MSE corresponds to the MSE between the original and the distorted image.

$$\text{PSNR} = 10 \log_{10} \left(\frac{R^2}{\text{MSE}} \right) \tag{9}$$

Structural Similarity Index Measure (SSIM) measures image quality by comparing structural information, luminance, and contrast between an image and a reference. Unlike PSNR, SSIM focuses on perceptual similarity. SSIM values range from -1 to 1, with 1 indicating perfect similarity. SSIM is defined in Equation (10), where x and y are the two images being compared. The parameters μ_x and μ_y represent the mean pixel values of images x and y, respectively. σ_x^2 and σ_y^2 are the variances of x and y, while σ_{xy} denotes the covariance between the two images. The constants C_1 and C_2 are included to stabilize the division, particularly when the denominators approach zero, and are typically defined as $C_1 = (K_1 L)^2$ and $C_2 = (K_2 L)^2$, where L is the dynamic range of pixel values (e.g., 255 for 8-bit images), and K_1 and K_2 are small constants (commonly set to 0.01 and 0.03, respectively).

$$\text{SSIM}(x, y) = \frac{(2\mu_x \mu_y + C_1)(2\sigma_{xy} + C_2)}{(\mu_x^2 + \mu_y^2 + C_1)(\sigma_x^2 + \sigma_y^2 + C_2)} \tag{10}$$

Mean Pixel Difference (MPD) is a simpler metric that calculates the average absolute difference in pixel values between two images. Lower MPD values indicate higher similarity. MPD is formulated as Equation (11), where N is the number of pixels, P_i is the original pixel value, and Q_i is the compared pixel value.

$$\text{MPD} = \frac{1}{N} \sum_{i=1}^{N} |P_i - Q_i| \tag{11}$$

Binary classification is widely used in applied Machine Learning across fields like medicine, biology, meteorology, and malware analysis. Performance metrics are crucial in research to evaluate and report the effectiveness of classification models. In binary classification, a two-by-two confusion matrix captures the model's performance, consisting of True Positives (TPs), True Negatives (TNs), False Positives (FPs), and False Negatives (FNs). Key metrics such as classification accuracy, precision, and recall are derived from these values [35].

Classification accuracy is the proportion of correct classifications out of all classifications, both positive and negative. Classification accuracy is formulated as Equation (12) and reflects the overall correctness of the classifier. A perfect model has an accuracy of 1.0 (or 100%), meaning no FPs or FNs.

$$\text{Classification Accuracy} = \frac{\text{TP} + \text{TN}}{\text{TP} + \text{TN} + \text{FP} + \text{FN}} \tag{12}$$

Recall or TP rate is the proportion of actual positives that were correctly identified. This metric is formulated in (13) and measures the model's ability to detect all positive instances. A perfect recall is 1.0, indicating a 100% detection rate with no FNs.

$$\text{Recall} = \frac{\text{TP}}{\text{TP} + \text{FN}} \tag{13}$$

False Positive Rate (FPR), also known as the probability of false alarm, is the proportion of actual negatives that were incorrectly classified as positives. This metric is formulated in Equation (14) and indicates the likelihood of misclassifying a negative instance as positive. A perfect model has a False Positive Rate (FPR) of 0.0 (or 0%), meaning no FPs.

$$\text{FPR} = \frac{\text{FP}}{\text{FP} + \text{TN}} \tag{14}$$

The precision metric is the proportion of correct positive classifications. This metric is formulated in Equation (15). Precision assesses the accuracy of the positive predictions made by the model. A perfect precision score is 1.0, achieved when there are no FPs. While precision improves as FPs decrease, recall improves as FNs decrease.

$$\text{Precision} = \frac{\text{TP}}{\text{TP} + \text{FP}} \tag{15}$$

2.3. DSE

DSE is a systematic process of analyzing and evaluating various design alternatives to find optimal solutions that meet specific requirements and constraints. In the context of embedded system design and AxC, DSE can be defined as systematically evaluating and analyzing various design alternatives to find optimal solutions that balance performance, power consumption, and accuracy trade-offs. The goal of DSE is to find the best approximate versions of an application or circuit that provide significant gains in efficiency (power, speed, area) while maintaining acceptable output quality for the target application [36,37].

Designing modern embedded computer systems presents numerous formidable challenges, as these systems typically must adhere to various strict and sometimes conflicting design criteria. Embedded systems aimed at mass production and battery-powered or passive cooling devices require cost-effectiveness and energy efficiency. In safety-critical applications like avionics and space technologies, dependability is paramount, especially with the growing autonomy of systems. Furthermore, many of these systems are expected to support multiple applications and standards simultaneously, necessitating real-time performance. For instance, mobile devices must accommodate various communication protocols and digital content coding standards [38].

Additionally, these systems must offer flexibility for future updates and extensions while maintaining programmability despite the necessity of implementing significant portions in dedicated hardware blocks due to performance, power consumption, and cost constraints. Consequently, modern embedded systems often adopt a heterogeneous multi-processor architecture comprising a mix of programmable cores and dedicated hardware blocks for time-sensitive tasks. This trend has led to the development of integrated heterogeneous Multi-Processor System-on-Chip (MPSoC) architectures [39].

To address the intricate design challenges inherent in such systems, a new design methodology known as system-level design has emerged over the past 15 to 20 years [40]. Its primary objective is to elevate the level of abstraction in the design process, thereby enhancing design productivity. Central to this approach are MPSoC platform architectures, which facilitate the reuse of IP components, along with the concept of high-level system modeling and simulation [41,42].

High-level system modeling and simulation enable the representation of platform components and their interactions at a refined level of abstraction. These models streamline the modeling effort, optimize execution speed, and are thus invaluable for early-stage DSE. Various design alternatives can be examined during DSE, including the number and type of processors utilized, the interconnection network employed, and the spatial and temporal binding of application tasks to processor cores [43,44].

Initiating DSE at the early stages of the design process is crucial, as the choices made can significantly impact the success or failure of the final product. However, the challenges the large design space poses, particularly in its early stages, cannot be overlooked. For instance, when considering different mappings of application tasks to processing resources to optimize system performance or power consumption, the design space expands exponentially with the number of tasks and processors, presenting an NP-hard problem [45]. Consequently, notable research has focused on developing efficient and effective DSE methods in recent years, aiming to tackle the complexities of exploring such expansive design spaces [45].

DSE approaches can be categorized and classified based on many aspects [46]. However, one of the possible classifications is categorizing a DSE as single-objective or multi-

objective. Multi-objective DSE refers to simultaneously optimizing multiple conflicting objectives or goals during the design phase. These objectives typically include performance, power consumption, area utilization, reliability, and cost in circuit design. Unlike single-objective optimization, where a single criterion is optimized at the expense of others, multi-objective DSE aims to find a set of trade-off solutions, known as the Pareto-optimal front, where improving one objective comes at the cost of degrading another. That is why finding an optimal solution that optimizes all objectives is impossible. Such a solution does not exist, and this happens when the objectives conflict with each other. Therefore, optimal decisions need to be taken with a trade-off between design criteria [44].

When applying AxC techniques to the design, what makes choosing among different AxC techniques difficult is that all of them can be employed together, requiring the evaluation of the impact of each combination of AxC techniques on the overall computation accuracy. The two main approaches for evaluating this impact comprise executing different approximated versions of the application several times with different configurations [47,48] or devising modeling techniques to simulate different approximated versions of the application in a time-optimized fashion [11,49]. The disadvantage of the first approach is that the evaluation time increases when the exploration reaches an exhaustive search. Applying pruning techniques to the exploration space reduces the exploration time to the application execution time, at best. Conversely, the second approach is more suitable for estimating the impact of AxC techniques on the application computation accuracy because it is faster than the first approach, even though it costs an error margin in the computation accuracy evaluation.

3. Literature Search Methodology

The objective of this survey was to systematically identify and classify existing literature on DSE methodologies proposed for finding the most suitable AxC techniques to be applied to a program or hardware design.

The keywords used for the search were carefully chosen based on common terminology found in influential works in the field. These included terms such as "Approximate Computing", "Design Space Exploration", "Multi-objective Optimization", and "Approximate Hardware/Software". Boolean operators and advanced filtering techniques were utilized to refine the search results in academic databases. This ensured a comprehensive yet focused set of papers covering various abstraction levels in both hardware and software implementations.

The selection criteria were established to maintain the relevance and quality of the review. We prioritized papers that provided detailed descriptions of DSE methodologies and excluded those that relied solely on exhaustive search methods. This approach aimed to highlight more sophisticated and efficient DSE approaches.

Once the relevant papers were identified, they were categorized based on the type of search algorithm employed for conducting the DSE. These categories included search algorithms such as ML, Evolutionary Algorithms (EAs), and custom algorithms. Papers were further sorted based on the target hardware for which the DSE was conducted, such as Field-Programmable Gate Arrays (FPGAs), Application-Specific Integrated Circuits (ASICs), Central Processing Units (CPUs), and Graphics Processing Units (GPUs). Additionally, we considered the application domains, including image processing, scientific computing, and signal processing. The categorization process is shown in Figure 2 to better visualize the process.

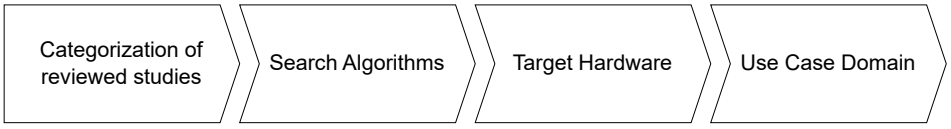

Figure 2. Steps of categorizing the proposed DSE approaches of the reviewed studies.

For instance, some studies focus on FPGAs and ASICs, particularly in the context of designing accelerators using AxC techniques. Some other studies do not specify a particular target hardware, indicating their proposed methods can be applied universally across different platforms. Additionally, some papers address hardware- or software-level approximations for GPUs.

This survey also considers the application domains of the programs targeted for approximation. The application domains usually considered in most studies for selecting benchmarks include image processing, signal processing, scientific computing, financial analysis, Natural Language Processing (NLP), 3D gaming, and robotics.

Additionally, information about employed AxC techniques was extracted from each study to better compare different studies based on the AxC techniques applied at software, architectural, or hardware levels.

In a nutshell, this survey aimed to identify and evaluate the proposed DSE methods employed to explore the extensive design space of approximate versions of a design. The focus was on understanding whether these methods were well-known search algorithms or custom approaches. Following this structured and systematic methodology, the survey provides a comprehensive overview of the current state-of-the-art proposed DSE methodologies for finding the most suitable AxC techniques, highlighting the diversity of approaches and their applicability to various hardware platforms and application domains. Figure 3 shows the aforementioned process of categorizing different studies.

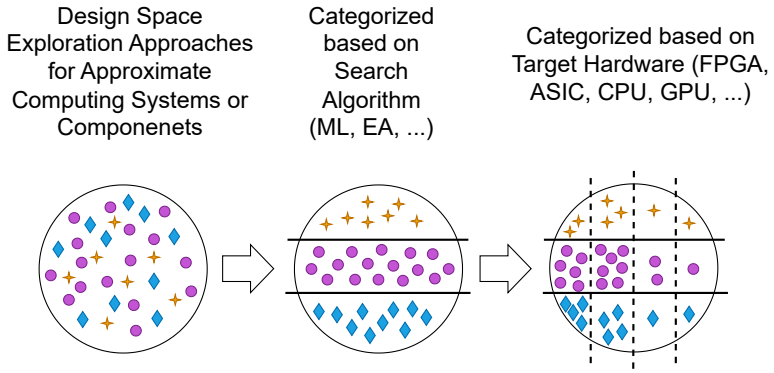

Figure 3. Categorizing the proposed DSE approaches of the reviewed studies based on employed search algorithms and target hardware.

4. Comparison and Analysis

This section provides an overview and comparison of the proposed DSE approaches in the literature for applying AxC techniques to programs or hardware designs. Though many different search algorithms have been proposed in the literature to explore the vast design space of approximate programs or hardware designs, two categories of algorithms are commonly leveraged: ML algorithms and EAs. ML approaches often leverage data-driven techniques to predict and explore optimal design configurations. At the same time, EAs use bio-inspired strategies such as Genetic Algorithms (GAs) to navigate the design space.

Table 2 provides information about the research works that took an ML approach to perform the DSE, while Table 3 includes information about the research works that leveraged EAs to perform the DSE. All the remaining research works that perform the DSE using other heuristic algorithms or combining different optimization algorithms are listed in Table 4. While Tables 2–5 provide an overview to allow comparison among different studies based on the employed search algorithm, target hardware, and use case domain, Tables 6–9 provide an overview of the same sets of studies to allow comparison among different studies based on AxC techniques applied in each study.

Table 2. Research works using ML-based search algorithms to perform the DSE.

Year	Ref.	Target Hardware	Use Case Domains	Benchmarks	Search Algorithm
2018	[50]	FPGA	Image processing	Iris scanning	RL
2021	[51]	FPGA	Image processing	Kernel-based Gaussian blur filter	MBO
2019	[12]	Accelerator (ASIC)	Image processing	Sobel and Gaussian blur filters (one with fixed coefficients, and one with a 3×3 kernel)	ML-based heuristic
2023	[52]	Accelerator (ASIC)	Signal and image processing and general arithmetic	Adder Tree, RGB2gray, FIR, and Gaussian blur filters	AI-based heuristic: modified MCTS
2021	[53]	FPGA, ASIC, general-purpose computing system	Image processing	RGB2GRAY, Ternary sum. FIR, image sharpening, and Gaussian blur filters	ML
2015	[54]	General-purpose hardware (heterogeneous mobile, tablet, and server processors)	Video encoder, financial analysis, image processing, search engine, digital signal processing, netlist place-and-route, image similarity search, and clustering algorithm	x264, Swaptions, Bodytrack, Swish++, Radar, Canneal, Ferret, and Streamcluster	RL
2020	[55]	General-purpose CPU and DNN accelerator	DNNs for Image and Digit Classification	AlexNet, SimpleNet, LeNet, MobileNet, ResNet-20, 10-Layers, and VGG-11/16	RL
2023	[37]	General-purpose CPU	ML, digital signal processing, and image processing	Matrix multiplication and FIR filter	RL

Table 3. Research works using EAs as a search algorithm to perform the DSE.

Year	Ref.	Target Hardware	Use Case Domains	Benchmarks	Search Algorithm
2021	[56,57]	FPGA	Image Processing	Pixel-streaming pipeline	GA
2023	[58]	FPGA-based approximate accelerator	HEVC	Multiplierless MCM	ES algorithm and NSGA-II
2022	[59]	Accelerator (FPGA and ASIC)	Image processing (JPEG compression)	DCT	NSGA-II
2014	[60]	General-purpose CPU (implied)	Scientific computing, 3D gaming, 3D image rendering, signal, and image processing	FFT, SOR, MC, SMM, LU, Zxing, JMEint, Imagefill, and raytracer	GA
2023	[61]	GPU	Image classification using CNNs	MobileNetV2 and ResNet50V2	NAS algorithms: EvoApproxNAS (based on NSGA-II)

Table 4. Research works using custom search algorithms to perform the DSE.

Year	Ref.	Target Hardware	Use Case Domains	Benchmarks	Search Algorithm
2016	[62]	ASIC, and FPGA	HEVC	SAD	Custom
2020	[63]	ASIC, and FPGA	Image classification using DNNs	ResNet-18/34/38/74, MobileNetV2, and Transformer-base/WikiText-103	Custom

Table 4. *Cont.*

Year	Ref.	Target Hardware	Use Case Domains	Benchmarks	Search Algorithm
2016	[64]	ASIC (implied)	Handwriting recognition, general arithmetic, multimedia, signal, and image processing	Array Multiplier, Carry Lookahead Adder, Kogge Stone Adder, Multiple and Accumulate, SAD, Euclidean distance, DCT, FFT, and FIR. All used in a DNN vector accelerator	Heuristic.
2019	[65]	ASIC (implied)	HEVC	SAD	Custom
2021	[66]	ASIC (implied)	Image processing	Sobel, FIR, and Gaussian blur filters, a ReLu Neuron, and Euclidean distance	Custom
2017	[67]	VLSI systems and HLS, aligning with ASIC design	Image processing	Average number calculator, inverse DCT calculator, Sobel, FIR, interpolation, and decimation filters	Custom
2020	[68]	HLS tools for accelerator design	Image processing	Sobel and Sharpen filters	TS with potential integration of GAs
2018	[69]	hardware accelerator (ASIC implied)	ML, digital signal processing, and image processing	Matrix multiplication, Sobel filter, and DCT	Custom
2016	[70]	ML accelerator (ASIC)	Image processing and NLP/text classification	Eye and face detection, optical digit, digit, webpage, and text classification	GD
2021	[71,72]	ASIC (AI Accelerator)	Image processing, NLP, and speech recognition	VGG16, ResNet50, InceptionV3, InceptionV4, MobileNetV1, SSD300, YoloV3, YoloV3-Tiny, BERT, a two-layer LSTM, and a four-layer bidirectional LSTM	Custom
2016	[73]	NPU (ASIC)	Financial analysis, robotics, 3D gaming, image compression, signal, and image processing	Blackscholes, FFT, Inversek2j, Jmeint, JPEG, Sobel filter	Custom
2019	[74]	Not specified (implied general-purpose)	Image processing	Matrix multiplication and FIR filter	Custom
2016	[75]	GPU	For GPU: ML, signal processing (pattern recognition), image processing, medical imaging, scientific computing, and web mining. For CPU: scientific computing and optimization.	For GPU: Backprop, Fastwalsh, Gaussian, Heartwall, Matrixmul, Particle filter, Similarity score, S.reduce, S.srad2, and String match. For CPU: Bwaves, CactusADM, FMA3D, GemsFDTD, Soplex, and Swim.	Custom

Table 5. Research works that perform the DSE for approximate functions design space instead of a complete system.

Year	Ref.	Target Hardware	Use Case Domains	Benchmarks	Search Algorithm
2021	[76]	FPGA (also ASIC implied)	NA	apex2, b12, clip, duke2, and vg2 benchmarks from IWLS'93 Benchmark Set	NSGA-II
2014	[77]	FPGA and ASIC (implied)	NA	Ripple Carry, Carry Lookahead, and Kogge Stone Adders. Wallace, and Dadda Multipliers	Custom

Table 5. *Cont.*

Year	Ref.	Target Hardware	Use Case Domains	Benchmarks	Search Algorithm
2021	[78]	FPGA and ASIC (designing fault-tolerant architectures)	Safety-critical applications: Quadruple Approximate Modular Redundancy (QAMR)	Generic combinational circuits	NSGA-II
2021	[79]	FPGA and ASIC (implied)	NA	Approximate adders, multipliers, divisor, barrel shifter, Sine, and Square	Heuristic
2022	[80]	FPGA	NA	Approximate adders, multipliers, decoders, and ALUs	NSGA-II

Table 6. AxC techniques in research works using ML-based search algorithms to perform the DSE.

Year	Ref.	Target Hardware	Benchmarks	AxC Techniques		
				Software	Architectural	Hardware
2018	[50]	FPGA	Iris scanning	Reducing search window size and the region of interest in iris images, reducing the parameters of iris segmentation	NA	Reducing the filter kernel size
2021	[51]	FPGA	Kernel-based Gaussian blur filter	Approximation of the window size, mode, and stride length for convolution kernels	NA	Approximate multipliers and adders
2019	[12]	Accelerator (ASIC)	Sobel, and Gaussian blur filters (one with fixed coefficients, and one with a 3×3 kernel)	NA	NA	Approximate adders and multipliers
2023	[52]	Accelerator (ASIC)	Adder Tree, RGB2gray, FIR, and Gaussian blur filters	NA	NA	Approximate adders and multipliers
2021	[53]	FPGA, ASIC, general-purpose computing system	RGB2GRAY, Ternary sum. FIR, image sharpening, and Gaussian blur filters	NA	NA	Approximate adders and multipliers
2015	[54]	General-purpose hardware (heterogeneous mobile, tablet, and server processors)	x264, Swaptions, Bodytrack, Swish++, Radar, Canneal, Ferret, and Streamcluster	PowerDial (changes program inputs data structure) and Loop Perforation	NA	NA
2020	[55]	General-purpose CPU and DNN accelerator	AlexNet, SimpleNet, LeNet, MobileNet, ResNet-20, 10-Layers, and VGG-11/16	DNN layer Quantization	NA	NA
2023	[37]	General-purpose CPU	Matrix multiplication and FIR filter	NA	NA	Approximate adders and multipliers

Table 7. AxC techniques in research works using EAs as a search algorithm to perform the DSE.

Year	Ref.	Target Hardware	Benchmarks	AxC Techniques		
				Software	Architectural	Hardware
2021	[56,57]	FPGA	Pixel-streaming pipeline	NA	NA	Sparse LUTs, precision scaling, approximate adders
2023	[58]	FPGA-based approximate accelerator	Multiplierless MCM	NA	NA	Approximate adders and multipliers
2022	[59]	Accelerator (FPGA and ASIC)	DCT	NA	NA	Approximate adders
2014	[60]	General-purpose CPU (implied)	FFT, SOR, MC, SMM, LU, Zxing, JMEint, Imagefill, and raytracer	Program static instructions	NA	NA
2023	[61]	GPU	MobileNetV2, and ResNet50V2	Approximate 8xN bit multipliers emulated using LUTs, approximate depthwise convolution, and quantization-aware training	NA	NA

Table 8. AxC techniques in research works using custom search algorithms to perform the DSE.

Year	Ref.	Target Hardware	Benchmarks	AxC Techniques		
				Software	Architectural	Hardware
2016	[62]	ASIC and FPGA	SAD	NA	NA	Approximate adders and logic blocks
2020	[63]	ASIC and FPGA	ResNet-18/34/38/74, MobileNetV2, and Transformer-base/WikiText-103	Progressive Fractional Quantization (PFQ), Dynamic Fractional Quantization (DFQ)	NA	NA
2016	[64]	ASIC (implied)	Array Multiplier, Carry Lookahead Adder, Kogge Stone Adder, Multiple and Accumulate, SAD, Euclidean distance, DCT, FFT, and FIR. All used in a DNN vector accelerator	NA	NA	Logic isolation using latches or AND/OR gates at the inputs, MUXes at the output, and power gating
2019	[65]	ASIC (implied)	SAD	NA	NA	Approximate adders
2021	[66]	ASIC (implied)	Sobel, FIR, and Gaussian blur filters, a ReLu Neuron, Euclidean distance	NA	NA	Clock gating and precision reduction of primary inputs at the RTL level
2017	[67]	VLSI systems and HLS, aligning with ASIC design	Average number calculator, inverse DCT calculator, Sobel, FIR, interpolation and decimation filters	Source-Code Pruning Based on Profiling	Functional Unit Substitution (additions and multiplications) at the HLS level	Internal signal substitution and Bit-Level Optimization at the RTL level
2020	[68]	HLS tools for accelerator design	Sobel and Sharpen filters	NA	NA	Approximate adders and multipliers

Table 8. *Cont.*

Year	Ref.	Target Hardware	Benchmarks	AxC Techniques		
				Software	Architectural	Hardware
2018	[69]	Hardware accelerator (ASIC implied)	Matrix multiplication, Sobel filter, and DCT	Partial product perforation in approximate multipliers, and truncation in approximate adders/subtractors	NA	Inexact compressor in approximate multipliers, approximate full adder, and VOS
2016	[70]	ML accelerator (ASIC)	Eye and face detection, optical digit, digit, webpage, and text classification	NA	NA	Clock overgating
2021	[71,72]	ASIC (AI Accelerator)	VGG16, ResNet50, InceptionV3, InceptionV4, MobileNetV1, SSD300, YoloV3, YoloV3-Tiny, BERT, a two-layer LSTM, and a four-layer bidirectional LSTM	NA	Precision reduction	Approximated activation functions, pooling, normalization, and data shuffling
2016	[73]	NPU (ASIC)	Blackscholes, FFT, Inversek2j, Jmeint, JPEG, Sobel filter	NA	Use of NNs (NPU accelerator)	NA
2019	[74]	Not specified (implied general-purpose)	Matrix multiplication and FIR filter	NA	NA	Approximate adders and multipliers
2016	[75]	GPU	For GPU: Backprop, Fastwalsh, Gaussian, Heartwall, Matrixmul, Particle filter, Similarity score, S.reduce, S.srad2, and String match. For CPU: Bwaves, CactusADM, FMA3D, GemsFDTD, Soplex, and Swim.	NA	NA	RFVP

Table 9. AxC techniques in research works that perform the DSE for approximate functions design space instead of a complete system.

Year	Ref.	Target Hardware	Benchmarks	AxC Techniques		
				Software	Architectural	Hardware
2021	[76]	FPGA (also ASIC implied)	apex2, b12, clip, duke2, and vg2 benchmarks from IWLS'93 Benchmark Set	NA	NA	Logic falsification
2014	[77]	FPGA and ASIC (implied)	Ripple Carry, Carry Lookahead, and Kogge Stone Adders, Wallace, and Dadda multipliers	NA	NA	Boolean network simplifications allowed by EXDCs
2021	[78]	FPGA and ASIC (designing fault-tolerant architectures)	Generic combinational circuits	NA	NA	Logic falsification
2021	[79]	FPGA and ASIC (implied)	Approximate adders, multipliers, divisor, barrel shifter, Sine, and Square	NA	NA	Approximation based on BMF for truth tables
2022	[80]	FPGA	Approximate adders, multipliers, decoders, and ALUs	NA	NA	Customized approximation of Boolean networks

4.1. DSE Using ML Algorithms

As reported in Table 2, the most popular ML algorithm is RL [37,50,54,55]. While authors in [51,52] use MBO and modified MCTS, respectively; authors in [12,53] mention using ML-based search algorithm. Among these research works, though the target hardware varies from FPGAs and ASICs to general-purpose CPUs, the use-case domain always includes image and signal processing benchmarks, ranging from traditional image processing to image classification using NNs. Moving the comparison to the AxC techniques applied at different levels, as reported in Table 6, replacing the exact adders and multipliers with approximate counterparts is the most common hardware-level approximation investigated [12,37,51–53]. However, the investigated software-level AxC techniques are noticeably application-specific: In [50,51], algorithm parameters—that indicate the iterations of executing a code basic block or the size of the inputs processed at each iteration—are decreased to reduce the execution time or program memory while sacrificing output accuracy. A similar approach of loop perforation is applied in [54] alongside changing the input data structure. Interestingly, in [55], an ML algorithm is employed to search the design space of an ML application, proposing a DSE framework to find the optimal quantization level for each layer of a DNN.

4.2. DSE Using EAs

Table 3 lists the research works that leveraged EAs to perform the DSE. Between the prominently used subsets of EAs, an ES algorithm is only used in [58]. All approaches enlisted here either use GA or its subset NSGA-II to explore the design space. More precisely, authors in [56,57,60] employ GA, authors in [58,59] use NSGA-II, and authors in [61] developed a NAS algorithm based on NSGA-II. Comparing the target hardware of the reviewed research works, most works consider optimizing an accelerator design for FPGA and ASIC implementation as expected. At the same time, the research in [61] targets GPUs for optimizing CNN designs.

Comparing the benchmarks in Table 3 to those listed in Table 2, most of the benchmarks fall under the image processing category, though the types of benchmarks are slightly different. Comparing the applied AxC techniques, as reported in Table 7, in [56,57] authors investigate employing sparse LUTs, precision scaling, and approximate adders for a pixel-streaming pipeline application accelerated with an FPGA. Similarly, in [58,59] authors explore using approximate adders and multipliers for optimizing video and image compression accelerators. In [60], authors try to optimize benchmarks from different domains such as scientific computing, 3D gaming, 3D image rendering, signal processing, and image processing when the approximation is applied at a software level, altering the program's static instructions. Distinctively, in [61], authors propose approximating multipliers using LUTs and a customized approximate convolutional layer to support quantization-aware training of CNNs and dynamically explore the design space. It is noteworthy that the aim is to optimize a CNN design usually trained on a GPU. Hence, approximation at the hardware level is not an option, while such an AxC technique can be emulated at the software level.

4.3. DSE Using Custom Algorithms

Table 4 reports a list of reviewed papers that neither rely on ML nor EAs to explore the design space. In [68], authors mention using a TS algorithm, with potential integration of GAs into the DSE framework. Notably, TS focuses on iteratively improving a single solution, whereas GAs work with a population of solutions and evolve them over generations using crossover, mutation, or other genetic operators. Hence, taking a TS approach might not seem the best choice when the MOP does not have a single optimum solution, and a Pareto front of non-dominated solutions may represent the optima better. In [70], authors select a GD approach to search the design space. Contrary to the fact that GD is a widely used optimization technique in ML, GD is not employed as a part of an ML search algorithm in the aforementioned work. All the remaining works in Table 4 employ custom algorithms.

In some cases, the DSE includes multiple stages of exploration, where pruning techniques are used before applying the search algorithm to reduce the design space size or after applying the search algorithm to refine the obtained solution sets.

Table 4 categorizes studies by target hardware, starting with those focused on FPGAs and ASICs, and continues with studies on optimized accelerator design. Table 8 reports the AxC techniques applied in each study enlisted in Table 4. Similar to the other sets of studies presented in Tables 2 and 3, only a few works reported in Table 4 are hardware-independent or target general-purpose CPUs and GPUs. The target hardware in [62] includes both FPGAs and ASICs. In [62], the DSE is performed to optimize a hardware accelerator design for a video processing application using approximate adders and logic blocks. Similarly, in [63], the target hardware includes both FPGAs and ASICs. In this study, the DSE is performed to optimize the design of DNNs accelerated using FPGAs and ASICs, while the AxC techniques applied are quantization techniques aimed at approximating the DNN design at the software level. In [64], authors perform the DSE with a heuristic search algorithm to optimize the hardware implementation of different functions used in a DNN vector accelerator. To apply approximation through logic isolation, the portions of logic in the circuit that consume significant power but contribute only minimally to output accuracy are identified. Then the DSE is performed to find the best trade-off between DNN classification accuracy and energy savings. It can be implied that the target hardware can be categorized as ASIC. In [65,66], the DSE is performed with custom algorithms, applying hardware approximations to hardware implementations of video and image processing benchmarks. The target hardware in these studies can be categorized as ASIC. In [67,68], the authors propose to modify the HLS tools to study the approximation effects.

Continuing through Table 4, in [69] the DSE is performed to optimize hardware accelerator design, investigating both hardware-level and software-level approximation techniques. The other three works also perform the DSE to optimize the accelerator design, specifically for ML applications [70–72]. In [73], authors target a very different type of acceleration using NPUs. While using NPUs for acceleration purposes can be categorized as applying approximation at the architectural level, the target hardware can be classified in the ASIC category. In [74], the target hardware is not explicitly mentioned; the proposed methodology applies to any DSE performed on general-purpose CPUs as target hardware. In [75], authors perform the DSE to find the best configuration when applying their proposed hardware-level approximation technique, which is specific to GPUs. However, the approximation technique is also applied to some benchmarks executed on general-purpose CPUs to provide a fair comparison between the results obtained by performing the DSE for both hardware targets.

Comparing the use case domains across Table 4, image and signal processing are the prevalent categories of applications. Moreover, ML applications for image and text classification, pattern and speech recognition, and NLP tasks are considered in many works. Some works also target image compression tasks. Many works include matrix multiplication, DCT, FIR, and Sobel filters in their studies, as these functions are crucial for many image-processing tasks. Some works also consider benchmarks from financial analysis, robotics, 3D gaming, and scientific computing domains.

Considering the AxC techniques mentioned in Table 8, studies in [62,65,68,69,74] investigate using approximate adders and multipliers. In [64], authors propose applying a hardware-level AxC technique called logic isolation using latches or AND/OR gates at the inputs, MUXes at the output, and power gating. In [66], authors propose applying another hardware-level AxC technique called clock gating alongside the precision reduction of primary inputs at the RTL level. Similarly, authors in [70] propose to apply a clock overgating technique. In [69], authors propose to use VOS alongside approximate adders and multipliers at the hardware level while approximating the additions and multiplications also on the software level. In [67], authors propose very different AxC techniques: Internal Signal Substitution and Bit-Level Optimization at the RTL level, Functional Unit Substitution (additions and multiplications) at the HLS level, and Source-Code Pruning Based on Profiling

at the software level. Also, in [71,72], authors propose applying AxC techniques at multiple levels while designing an AI accelerator. They propose applying precision reduction to DNN data as well as using approximated versions of fundamental DNN functions such as activation functions, pooling, normalization, and data shuffling in the network accelerator design. Though also aiming at optimizing DNN designs, authors in [63] propose to apply AxC techniques at the software level to enable dynamic quantization of the DNN during the training phase. Interestingly, in [73], authors propose to use a very different AxC technique compared with all of these reviewed studies. The proposed approach includes approximating the entire program using an NPU as an accelerator. Another interesting AxC technique is proposed in [75] to tackle memory bottlenecks while executing the program on a GPU and transferring the data from CPUs to GPUs and vice versa.

4.4. DSE of Approximate Functions Design

Some studies in the literature propose approaches to efficiently explore the design space for approximate logic synthesis and consider approximate versions of circuits generated by approximating selected portions (or sub-functions) of Boolean networks. These studies are reported separately in Table 5, while the applied AxC techniques in these studies are reported in Table 9. The approximation is applied at the hardware level and involves logic falsification in [76,78]. The approximation technique in [77] is based on Boolean network simplifications allowed by EXDCs. The approximation in [79] is based on BMF for truth tables. And, in [80], a customized approximation of Boolean networks is applied. The search algorithm to explore the design space is an NSGA-II in [76,78,80] while in [77,79] authors employ customized and heuristic algorithms. While the benchmarks for all of these studies include well-known approximate adders and multipliers in the literature, other circuits such as ALUs, decoders, shifters, and multiple combinational circuits have been employed as benchmarks. Interestingly, in [78], the study targets safety-critical applications. The Quadruple Approximate Modular Redundancy (QAMR) approach is opposed to Triple Modular Redundancy (TMR), where all modules are exact circuits.

4.5. Evaluated Parameters in DSE

While Tables 2–5 provide an overview to allow comparison among different studies based on employed search algorithm, target hardware, and use case domain, Tables 10–13 provide an overview of the same sets of studies to allow comparison among different studies based on evaluated parameters involved in the trade-off imposed by approximation.

Table 10. Evaluated metrics in research works using ML-based search algorithms to perform the DSE.

Year	Ref.	Accuracy	Error Metric(s)	Power or Energy	Time	Performance	Memory	Area	Pareto Front
2018	[50]	Industry-level threshold for iris encoding	HD of any two images	Energy	Execution time	NA	Memory utilization	Total logic utilization	Speedup and Average Error
2021	[51]	Application-level error	MAE	NA	NA	NA	NA	LUT count	LUT count and Application-level error
2019	[12]	QORs	SSIM	Energy	NA	NA	NA	Area	SSIM and area. SSIM and energy
2023	[52]	Output accuracy	MRED and PSNR	NA	NA	NA	NA	Area	Area savings and error
2021	[53]	Output accuracy	MRED and PSNR	Power	NA	NA	NA	Area	NA

Table 10. *Cont.*

Year	Ref.	Accuracy	Error Metric(s)	Power or Energy	Time	Performance	Memory	Area	Pareto Front
2015	[54]	Application accuracy	Precision and recall, PSNR, Swaption price, track quality, wire length, similarity, quality of clustering	Energy efficiency	NA	Speedup	NA	NA	Energy savings and accuracy
2020	[55]	Relative DNN accuracy and quantization level	State of relative accuracy and state of quantization	Energy	NA	Speedup	NA	NA	State of relative accuracy and State of quantization
2023	[37]	Output accuracy	MAE	Power	Computation time	NA	NA	NA	NA

Table 11. Evaluated metrics in research works using EAs as a search algorithm to perform the DSE.

Year	Ref.	Accuracy	Error Metric(s)	Power or Energy	Time	Performance	Memory	Area	Pareto Front
2021	[56,57]	Output image quality	Color differences in perceptually uniform color space (CIELAB ΔE)	Power	NA	NA	NA	NA	Power consumption and maximum ΔE
2023	[58]	Output quality, QORs	PSNR	Power	NA	NA	NA	LUT count	Power and PSNR. LUT count and PSNR.
2022	[59]	Output image quality	Mean Structural SIMilarity (MSSIM) and Structural DISSIMilarity (DSSIM)	Power	NA	NA	NA	Area	Area and DSSIM (for ASIC). LUT count and DSSIM (for FPGA). Power and DSSIM (for both).
2014	[60]	QORs	ER, average entry difference, MPD, (average) normalized difference	Energy	NA	NA	NA	NA	NA
2023	[61]	CNN Top-1 Accuracy	Classification accuracy (%)	Energy of multiplication in convolutional layers	CNN training and inference time overhead	NA	NA	NA	Energy and CNN accuracy

Table 12. Evaluated metrics in research works using custom search algorithms to perform the DSE.

Year	Ref.	Accuracy	Error Metric(s)	Power or Energy	Time	Performance	Memory	Area	Pareto Front
2016	[62]	Output quality	BER (%)	Power	NA	Performance (Bit Rate)	NA	Area	NA

Table 12. *Cont.*

Year	Ref.	Accuracy	Error Metric(s)	Power or Energy	Time	Performance	Memory	Area	Pareto Front
2020	[63]	DNN accuracy	Classification accuracy (%)	Training energy	Training latency	NA	NA	NA	NA
2016	[64]	DNN classification accuracy	Classification accuracy	Energy savings	NA	NA	NA	NA	NA
2019	[65]	Output accuracy and coding efficiency	MAE, and Bjontegaard delta PSNR (BD-PSNR)	Power dissipation savings	NA	NA	NA	Circuit area savings	MAE and total power. MAE and Circuit Area. Power dissipation and BD-PSNR. Area and BD-PSNR.
2021	[66]	Output accuracy	MRED	Energy reduction	NA	Performance	NA	Area overhead	Power and output accuracy
2017	[67]	Computation accuracy	MAE	Power	NA	Performance	NA	Circuit Area	Circuit Area and MAE
2020	[68]	Output accuracy	MED and PSNR	PDP	NA	NA	NA	Area	Area and MED. PDP and MED.
2018	[69]	Output accuracy	MRED	Power	NA	NA	NA	NA	Power and Error
2016	[70]	Output quality	Classification accuracy	Energy	NA	NA	NA	NA	NA
2021	[71,72]	DNN accuracy	Classification accuracy (%)	NA	Inference Latency	Inference compute efficiency and training throughput	NA	NA	NA
2016	[73]	Output quality	MRED, miss rate, image difference	Energy reduction	NA	Speedup	NA	NA	NA
2019	[74]	Output accuracy	Normalized weighted error	NA	NA	NA	NA	NA	NA
2016	[75]	Output quality	For GPU: MRED, average Euclidean distance, image difference, total mismatch rate, and normalized RMSE (also for CPU)	Energy	Execution time	Speedup	Memory bandwidth	NA	Product of energy dissipation, execution time, and error traded off with predictor size

Table 13. Evaluated metrics in research works that perform the DSE for approximate functions design space instead of a complete system.

Year	Ref.	Accuracy	Error Metric(s)	Power or Energy	Time	Performance	Memory	Area	Pareto Front
2021	[76]	Circuit output accuracy	BER	NA	NA	NA	NA	NA	Resources (in terms of Cubes and Literals) and BER
2014	[77]	Circuit output accuracy	Error magnitude, error frequency constraints	NA	NA	NA	NA	NA	Normalized gate count and error frequency
2021	[78]	NA	NA	NA	Critical path delay	NA	NA	Circuit Area	Delay gain and area gain
2021	[79]	Circuit output accuracy	Normalized HD, and MAE	Power	NA	NA	NA	Area	Power and MAE. Area utilization and MAE.
2022	[80]	Accuracy degradation observable at primary outputs (POs)	ER	NA	NA	NA	NA	LUT count	ER and LUT count

Since AxC trades off accuracy for performance and energy efficiency, the first important parameter to evaluate during DSE is accuracy. Depending on the approximation goals, parameters measured during DSE in different studies may vary.

Predictably, power consumption is a key parameter frequently targeted in the reviewed studies, as it directly impacts energy efficiency. However, many studies choose to target energy consumption instead of power consumption. This approach is entirely valid because energy savings inherently indicate power savings, considering that energy is the product of power and time. By measuring energy directly, these studies effectively capture the combined impact of power reduction and execution time, providing a comprehensive view of the gains in efficiency achieved through AxC techniques.

The second most in-demand parameter, especially when designing accelerators, is the circuit area. Understandably, when approximations are applied to optimize a design, specifically in the case of employing the design on FPGAs, reducing the area utilization, or LUT count, is one of the approximation goals.

After area, performance and execution time are the most commonly measured parameters. In applications such as Artificial Neural Networks (ANNs), where execution time is inherently high, one of the primary goals of applying approximation is to reduce this execution time, particularly for inference and, when feasible, training. The lengthy execution times of these applications also directly impact the DSE time, as evaluating even a few approximate instances can become highly time-consuming. While typical application execution times may range from seconds to minutes, the DSE time needed to explore and evaluate possible approximations often extends to hours or days. In the case of ANNs, the execution time for inference alone can take hours, and the DSE time required to assess even a limited number of approximate instances can span several days. Therefore, in applications where the execution time is already considerable and hence a primary target

to trade-off with accuracy, proposing DSE methodologies that can assess more approximate instances in a reasonable time becomes crucial.

Memory utilization is often the least frequently evaluated parameter in the reviewed studies. Many AxC techniques are primarily applied to optimize execution time, energy, or performance rather than specifically targeting memory utilization. However, these techniques can still impact memory utilization. For instance, some techniques aimed at reducing execution time, energy consumption, or improving performance may also affect memory usage as a secondary outcome. This indirect influence on memory is an important consideration, even though it is not the primary focus of these techniques. For example, in [50], authors propose to explore the design space comprised of approximate versions of an iris scanning pipeline. The approximation includes reducing the search window size and the region of interest in iris images, reducing the parameters of iris segmentation, and reducing the kernel size of the filter. Though the main target is to reduce program execution time, the memory needed to store the intermediate and final output images and program parameters is reduced. In [75], authors propose an AxC technique to mitigate the bottlenecks of limited off-chip bandwidth and long access latency when the data are transferred from CPU to GPU and back. When a cache miss happens, RFVP predicts the requested values. In this case, the main goal of approximation is to achieve off-chip memory bandwidth consumption reduction, while speedup and energy reductions are also reported.

Through Tables 10–13, besides the accuracy column, there is an error metric(s) column that reports the error metric(s) presented in each study to measure accuracy degradation due to applying approximation. Among all the parameters mentioned—power consumption, execution time, performance, memory utilization, and circuit area—accuracy is unique because the metrics used to measure accuracy degradation are often more complex and application-specific. For example, while power consumption differences are reported simply as ED between the measurements from approximate and exact versions, accuracy degradation error metrics involve a variety of sophisticated measures that are tailored to the specific application domain. In Table 1, the most popular error metrics are listed.

Pareto Front Analysis

Finally, every proposed DSE approach results in a solution or a set of optimal solutions for the MOP. In some cases, an optimal solution exists. In many other cases, no global optimal solution can be found, and a Pareto front (a set of non-dominated solutions) is presented. The last column in Tables 10–13 indicates the studies that reported a Pareto front as the result of the DSE performed, or at least compared a set of solutions resulted from the proposed DSE approach with a Pareto front obtained by exhaustive search or other methods. In most cases, the obtained Pareto front shows a trade-off between accuracy on the one hand and an evaluated parameter, such as energy efficiency, on the other hand [12,50–52,54,56–59,61,65–69,76–80].

The rest of the reviewed studies that did not obtain a Pareto front but provided other analysis methods for comparing the DSE results are considered hereafter.

In some studies, a single threshold or multiple thresholds for the acceptable accuracy degradation was set, and then the DSE was performed for each accuracy threshold. For example, in [53], a solution was provided for each accuracy threshold. In [62], performance is plotted for different accelerator designs. However, no Pareto front is provided. Also, in [64], three different DNN accuracy thresholds were set, and the DSE was performed for each threshold. Hence, the plots show the energy reductions for each DNN accuracy threshold instead of a Pareto front. Similarly, in [70], the plots show the energy reductions for each DNN accuracy threshold instead of a Pareto front.

In [55], two application-specific error metrics were proposed to evaluate the accuracy for DNN quantization and plot the quantization space Pareto frontier for these two error metrics called State of Relative Accuracy and State of Quantization.

In [37], an RL approach was selected for performing the DSE, and steps of exploration have been plotted for evaluated parameters, including accuracy; however, a Pareto front is

not obtained. In [60], plots show the accuracy and energy against multiple thresholds for the number of program instructions to be approximated. However, a comparison to the Pareto front is not provided.

In [63], the quantization is applied dynamically during training and inference of the DNN. Therefore, the plots show the changes in the DNN accuracy concerning the number of MACs used in the computations. Also, in the same plot, the results are compared with other quantization-aware approaches in the literature instead of comparing the results with a Pareto front obtained by other DSE methods. Since in dynamic approximation of the DNNs, the changes in accuracy during the training or inference are more representative of the approach effectiveness, plotting a Pareto front seems unnecessary.

In [71] and the previous studies with the same framework [72], no Pareto front was presented. Instead, for each DNN, compute efficiency, training throughput, and inference latency were reported. In [73], an NPU is employed as an accelerator for a frequently executed region of code or function to approximate the function by replacing it with a neural network. Since an ANN is employed, similar to other works on ANNs, multiple thresholds for function quality loss, or in other words, different ANN accuracy levels, were investigated. Hence, the speedup and energy reduction for multiple thresholds of function quality loss were plotted. In consequence, no Pareto front was demonstrated.

4.6. DSE Methodologies Comparison by Use Case Domain

In this subsection, we categorize the reviewed studies based on their general use case domains, such as image processing, ML, signal processing, and scientific computing. For each use case domain, the studies are compared based on metrics used in DSE, such as accuracy, power savings, execution time, and area utilization.

4.6.1. Image Processing Applications

Several studies applied DSE methodologies to optimize image processing applications, balancing accuracy with power and area savings.

- Hashemi et al. [50] focused on iris scanning applications using RL to reduce filter kernel sizes. This study achieved notable energy savings and area utilization improvements while maintaining acceptable accuracy, measured by the HD between images.
- Ullah et al. [51] used MBO to approximate Gaussian blur filters. The study reported reduced LUT count and power savings, with output accuracy evaluated using MAE.
- Mrazek et al. [12] applied approximate multipliers and adders to Sobel and Gaussian blur filters, achieving significant energy savings with minimal SSIM loss.
- Rajput et al. [52] employed AI-based heuristics to optimize image processing tasks such as RGB2gray and Gaussian blur filters. The study reported accuracy levels evaluated with MRED and PSNR while achieving area and energy savings.
- Awais et al. [53] optimized image processing applications, including RGB2gray and FIR filters, by applying approximate adders and multipliers, achieving power savings and maintaining acceptable accuracy levels.
- Manuel and Kreddig [56,57] used EAs to optimize a pixel-streaming pipeline for image processing, achieving power savings with limited color differences, measured using the CIELAB ΔE metric.
- Barbareschi et al. [59] applied NSGA-II for JPEG compression optimization, balancing power, area, and image quality, evaluated using MSSIM and DSSIM metrics.
- Savino et al. [74] used custom algorithms for optimizing image processing tasks (matrix multiplication and FIR filters), reporting reductions in area and power with minimal accuracy degradation.

Overall, these studies reveal that different DSE approaches have been successfully applied to various image processing tasks. Most studies show that approximate hardware components, such as adders and multipliers, provide significant power savings with minimal accuracy loss. While some methods, like those used by Hashemi et al. [50] and Mrazek et al. [12], focus on energy efficiency, others, like Rajput et al. [52] and Barbareschi

et al. [59], emphasize balancing area utilization with accuracy and energy consumption. However, the wide range of objectives and varying use cases across these studies makes it difficult to perform a unified performance comparison beyond these metrics.

4.6.2. ML Applications

The reviewed studies also applied DSE methods to optimize Machine Learning models, particularly Deep Neural Networks (DNNs).

- Elthakeb et al. [55] used RL for quantization of CNN layers, achieving accuracy improvements and energy savings by adjusting quantization levels dynamically.
- Pinos et al. [61] applied NAS algorithms for optimizing CNN models like MobileNetV2 and ResNet50V2, reporting energy savings during inference with minimal accuracy degradation.
- Fu et al. [63] focused on DNN training, using dynamic fractional quantization to optimize training energy and latency while maintaining classification accuracy across multiple ResNet models.
- Venkataramani et al. [71,72] applied precision reduction techniques in AI accelerators for DNNs like VGG16 and BERT, achieving inference latency and energy reductions with minor accuracy trade-offs.

All studies in this category target improving energy efficiency while maintaining acceptable accuracy levels for Machine Learning applications. While Elthakeb et al. [55] and Pinos et al. [61] focus on optimizing DNN quantization and inference energy, Fu et al. [63] specifically optimize training energy and latency. Venkataramani et al. [71,72] focus on overall system-level reductions in inference latency and compute efficiency. The primary challenge in comparing these methods lies in the varying objectives, such as training versus inference optimization. However, all studies demonstrate that precision reduction and quantization techniques are highly effective in balancing accuracy and energy consumption in Machine Learning applications.

4.6.3. Signal Processing Applications

Signal processing tasks require optimization in terms of power consumption, performance, and accuracy, which several studies achieved by applying hardware AxC techniques.

- Mrazek et al. [12] used approximate multipliers and adders in Sobel and Gaussian blur filters, achieving significant power savings and minimal SSIM loss.
- Rajput et al. [52] employed AI-based heuristics to optimize signal processing tasks like FIR filters, showing a trade-off between area savings and output accuracy, measured using MRED and PSNR.
- Saeedi et al. [37] applied RL to optimize FIR filters and matrix multiplication, achieving power savings with acceptable MAE levels.
- Alan et al. [66] used custom algorithms to optimize Sobel and Gaussian blur filters, achieving power and area savings by applying clock gating and precision reduction techniques.

In signal processing applications, the main focus is on achieving power and area savings while maintaining output accuracy. While Mrazek et al. [12] and Alan et al. [66] emphasize hardware-level optimizations, such as approximate multipliers and clock gating, Rajput et al. [52] and Saeedi et al. [37] focus on trade-offs between accuracy and energy savings. Despite these differences, all studies demonstrate that approximate computing techniques are effective in optimizing power consumption with minimal impact on accuracy.

4.6.4. Scientific Computing Applications

Scientific computing tasks often involve balancing execution time, energy efficiency, and accuracy. Several studies focused on optimizing scientific computing applications using DSE techniques.

- Park et al. [60] applied genetic algorithms to scientific computing tasks such as FFT and Successive Over-Relaxation (SOR), achieving energy reductions with slight accuracy trade-offs, measured using normalized difference.
- Yazdanbakhsh et al. [75] applied custom DSE techniques to optimize tasks on GPUs, showing improvements in speedup and energy savings while balancing accuracy and memory bandwidth.
- Mahajan et al. [73] applied NPUs as accelerators to scientific tasks such as FFT, reporting significant energy reductions with acceptable accuracy, measured using miss rate and image difference.

In scientific computing applications, the primary goal is to reduce execution time and energy consumption while maintaining accuracy. Although the studies employ different techniques, such as genetic algorithms in Park et al. [60], GPU optimization in Yazdanbakhsh et al. [75], and NPU acceleration in Mahajan et al. [73], they all report improvements in speedup and energy savings. Due to the variety of techniques and the specific application focus of each study, further direct comparison between the studies is challenging.

4.6.5. Video and Audio Processing Applications

Several studies targeted video processing applications, with a focus on optimizing power, area, and accuracy.

- Hoffmann et al. [54] applied RL for video encoding tasks like x264, achieving energy savings through techniques like loop perforation while maintaining acceptable accuracy levels.
- Prabakaran et al. [58] used NSGA-II to optimize HEVC video processing tasks, showing trade-offs between area, power, and accuracy, evaluated using PSNR.
- Shafique et al. [62] applied custom DSE methods to optimize HEVC processing, achieving power savings through approximate adders with minimal quality loss, measured by BER.

The reviewed studies on video and audio processing show that DSE methodologies focus on balancing power and area savings with maintaining acceptable accuracy or quality metrics like PSNR and BER. Hoffmann et al. [54] applied more general techniques like loop perforation, whereas Prabakaran et al. [58] and Shafique et al. [62] used hardware-specific optimizations. Though they all focus on video encoding and compression tasks, the diversity in techniques and objectives limits the potential for direct comparison beyond energy savings and accuracy metrics.

4.6.6. Robotics, Financial Analysis, and Other Applications

A few studies focused on specialized domains such as robotics and financial analysis.

- Mahajan et al. [73] applied NPU accelerators in robotics and financial analysis applications, achieving energy reductions while maintaining acceptable accuracy levels, as measured by metrics such as MRED and miss rate.
- Hoffmann et al. [54] applied RL to financial analysis tasks, such as Swaptions, showing significant energy savings while balancing accuracy, measured by Swaption price.

For robotics and financial analysis applications, the reviewed studies focus primarily on energy savings, using techniques such as NPU acceleration (Mahajan et al. [73]) and RL-based optimization (Hoffmann et al. [54]). While these studies achieve notable results, the specificity of their application domains and techniques limits further comparison.

In conclusion, while each study employs DSE methodologies to optimize approximate designs based on metrics like output accuracy, power savings, and execution time, comparing them even within the same use case domain is challenging due to differences in target hardware, benchmarks, AxC techniques, and optimization goals. For example, some of the reviewed studies focused on image processing applications, such as Sobel filters and Gaussian blur, and employed approximate hardware to achieve power savings. In

contrast, some studies targeting Machine Learning applications, such as Neural Networks for image classification (e.g., ResNet or MobileNet models), employ approximate computing techniques like quantization to balance energy consumption and accuracy. Despite shared metrics, the diversity in techniques and objectives, such as reducing neural network inference latency on FPGAs versus optimizing power consumption on ASICs, makes direct comparisons difficult. Instead, most studies highlight how their proposed DSE methods can help identify the most suitable approximate designs within specific domains, optimizing metrics according to the unique needs of each target software or hardware.

4.7. Comparison of the Reviewed DSE Approaches by Frequent Categorizes

Figure 4 presents the distribution of the reviewed studies based on the search algorithms employed to perform DSE for approximate designs. The results reveal that 48.48% of the studies opted for custom algorithms, underscoring that nearly half of the approaches rely on application-specific methods, potentially due to the need for pruning large design spaces and addressing particular hardware or software constraints. This significant portion suggests that custom algorithms remain popular for solving highly specialized DSE problems in the AxC domain.

On the other hand, 27.27% of the studies employed EAs, and 24.24% utilized ML-based methods. While the data indicate that roughly a quarter of the surveyed works leveraged ML techniques such as RL, EAs are slightly more popular, likely due to their proven flexibility and scalability when dealing with a wide variety of design spaces and their robustness in handling complex MOP.

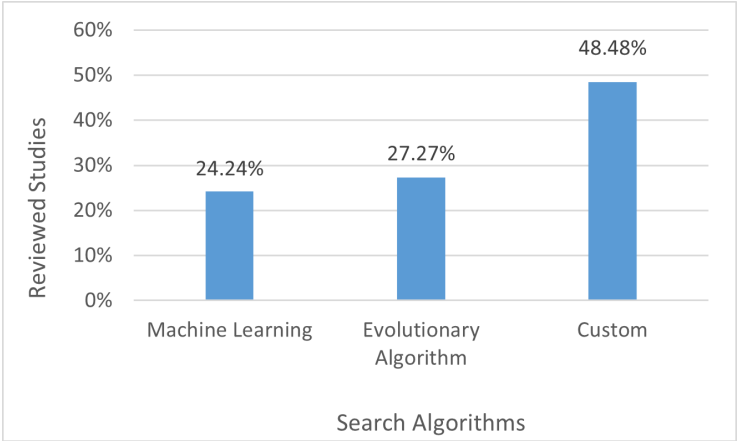

Figure 4. Distribution of the proposed DSE approaches of the reviewed studies, based on employed search algorithms.

Figure 5 shows the distribution of the reviewed studies based on the target hardware of each study. The results indicate that a significant proportion of the studies, 36.36%, targeted ASICs. This reflects the preference of the AxC domain for ASICs, as they allow for highly customized and efficient hardware designs, making them a favorable target for employing AxC techniques. The second largest group of studies, comprising 21.21%, targeted both FPGAs and ASICs, suggesting that many DSE methodologies are designed to be versatile enough to optimize for both reconfigurable and dedicated hardware, which enables flexibility depending on the specific design requirements.

On the other hand, 18.18% of the studies focused solely on FPGAs. This percentage highlights the importance of FPGAs in approximate computing, especially in cases where reconfigurability is critical, such as during iterative design processes or for applications requiring adaptable precision levels. Furthermore, 12.12% of the studies targeted general-purpose CPUs, indicating that even though some studies did not include hardware-specific optimizations, general-purpose processors are still valuable in certain contexts, particularly for applying software-level AxC techniques.

Interestingly, only 6.06% of the studies targeted GPUs, showing that while GPUs are effective for parallel processing, they may not be as frequently used as much as ASICs or FPGAs for applying AxC techniques. The last three columns reflect multi-platform approaches, where the target hardware spans multiple categories, such as FPGAs, ASICs, and general-purpose CPUs, each making up 3.03% of the reviewed studies. This indicates that some DSE approaches aim to be flexible and adaptable to various hardware platforms, which may be driven by the need for multi-objective optimization across different computing systems.

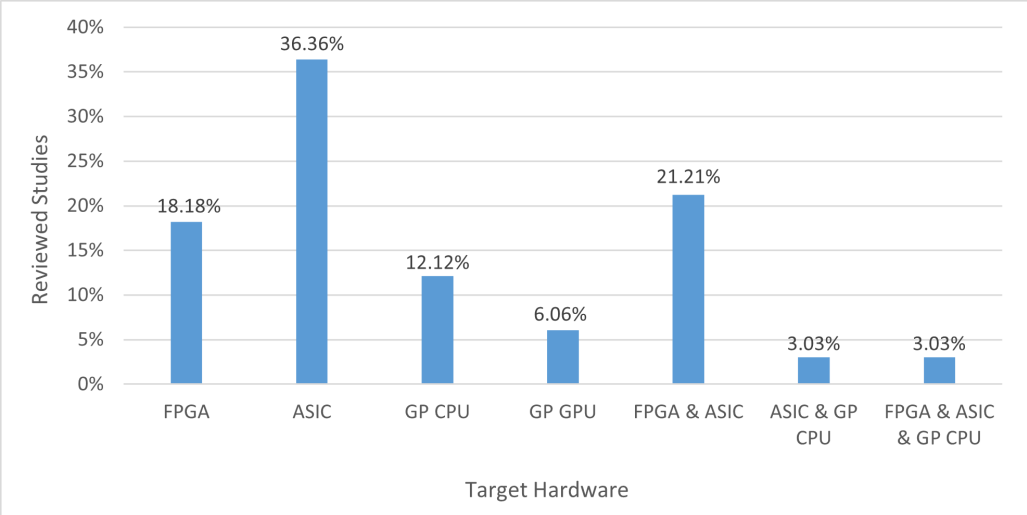

Figure 5. Distribution of the proposed DSE approaches of the reviewed studies, based on the target hardware.

Figure 6 presents the distribution of the reviewed studies based on the use case domains of the case study benchmarks used in each reviewed paper for conducting experiments. It is important to note that most studies examined benchmarks from multiple domains. Hence, such studies are included in counting the total number of studies for each use case domain.

As the total number of studies for each column shows, image processing tasks, such as Sobel and Gaussian blur filters, are the most popular among the reviewed studies. ML applications are also popular among the reviewed studies. The three columns labeled "DNNs for Image Classification and Pattern Recognition", "Machine Learning", and "Natural Language Processing" are all considered ML applications. It is noteworthy that the column labeled "Machine Learning" shows the number of case studies that cannot be categorized under the other two columns. Signal Processing, especially Digital Signal Processing (DSP) applications, and Video Processing (especially HEVC) applications are the next commonly used applications in the reviewed studies, respectively.

To conclude, the distribution of use case domains demonstrates a clear preference for image processing and Machine Learning tasks as key benchmarks for applying AxC techniques. This trend can be attributed to the inherent tolerance of these domains to approximation, where minor accuracy losses are often acceptable in exchange for significant gains in power and resource savings. Furthermore, signal processing and video processing applications also emerge as prominent areas, reflecting their suitability for applying approximations in embedded systems and real-time processing environments. The diverse set of use case domains illustrates the adaptability of AxC techniques across various computational tasks, emphasizing the versatility and growing importance of DSE methodologies in optimizing a wide range of applications.

Figure 6 also illustrates the distribution of search algorithms employed for performing the DSE for various use case domains. These search algorithms are categorized into three main classes: "Machine Learning", "Evolutionary Algorithm", and "Custom". Custom

algorithms are the most commonly used, especially for image processing, signal processing, and video processing tasks. This popularity may be due to the need for application-specific optimizations and pruning techniques in large design spaces. The same trend is observable for image classification and pattern recognition tasks.

On the other hand, EAs are applied across domains like image processing, video processing, and scientific computing, owing to their robustness in solving MOPs. ML-based DSE approaches are mostly utilized in image processing and signal processing tasks. Interestingly, in these domains, ML and Custom algorithms were employed almost equally to perform the DSE, while EAs were less employed. This trend may be attributed to the fact that ML techniques, such as RL and NAS, excel at dynamically optimizing design trade-offs in structured, data-rich environments like image and signal processing. Additionally, these domains often require real-time processing or adaptive techniques, which ML algorithms are well-suited for. Custom algorithms, on the other hand, are favored for their flexibility in handling domain-specific constraints and heuristics that are tailored to the problem. However, EAs, despite their proven multi-objective optimization capabilities, might be less efficient in environments where specific, fast-converging solutions are necessary due to time or resource constraints.

Overall, while custom algorithms dominate most tasks, ML and EAs show strong adaptability, each offering unique advantages based on the complexity and nature of the design space being explored.

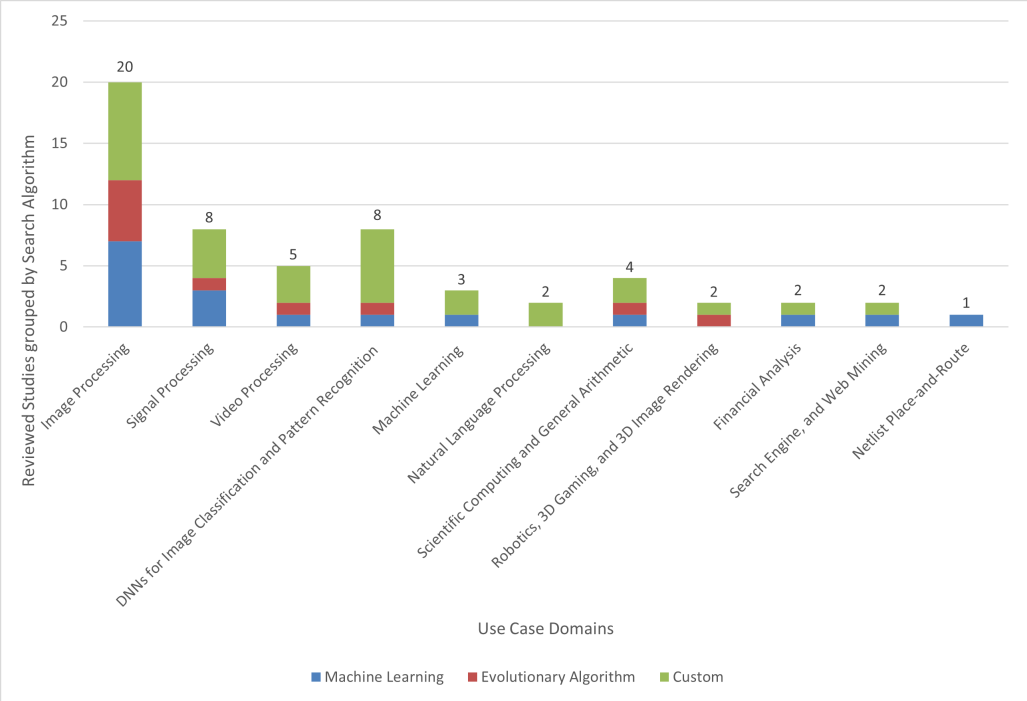

Figure 6. Distribution of the proposed DSE approaches of the reviewed studies, based on the use case domains.

5. Conclusions

This survey systematically reviewed and classified existing literature on DSE methodologies aimed at identifying suitable AxC techniques for various applications and hardware designs. The search strategy focused on papers that provided detailed descriptions of their DSE algorithms, deliberately excluding those utilizing exhaustive search methods to highlight more sophisticated and efficient approaches.

Two dominant categories of DSE methods emerged from the reviewed studies: ML approaches and EA methods. While both methodologies have strengths, the relative advantages depend largely on the complexity of the design space and the target application.

The ML approaches, especially those leveraging RL, are highly effective in domains like image processing and DNNs, where the design space is structured and substantial data from previous executions is available. These approaches excel at fine-tuning trade-offs between power and accuracy dynamically, making them suitable for applications requiring frequent reconfigurations or those with domain-specific training data [50,55]. However, ML-based DSE methods tend to be more application-specific and less adaptable to broader hardware platforms or large, complex design spaces [51,52].

In contrast, EA-based methods, particularly those utilizing GAs and NSGA-II, offer greater flexibility and scalability. These methods are especially suited for complex design spaces like FPGAs, ASICs, and other hardware-specific optimizations. GAs have been proven to deliver robust, Pareto-optimal solutions across a variety of applications, including image and video processing, where they consistently outperform other approaches in balancing circuit area, power consumption, and execution time [56,59]. Moreover, NSGA-II has shown particular promise in optimizing DNNs and delivering effective performance-power trade-offs in NN accelerator designs [61].

In conclusion, while ML-based DSE approaches offer fine-tuned control and are highly effective in specialized applications, EA-based DSE methods, especially GAs, are better suited for complex and large-scale design spaces that require general-purpose optimization. Future research may combine the adaptability of EA-based DSE approaches with the fine-grained control of ML-based methods, creating hybrid models that can optimize across diverse hardware environments and application domains.

Furthermore, a persistent challenge is the increasing complexity posed by heterogeneous systems, where different hardware platforms such as FPGAs, ASICs, and GPUs introduce unique constraints. Addressing this challenge will require the development of generalizable DSE methodologies that can operate across multiple hardware platforms, an area that necessitates further research.

Additionally, as discussed in Section 4, several studies have successfully integrated pruning techniques to reduce the complexity and size of the design space, leading to more efficient exploration. Nonetheless, future work must focus on reducing the computational overhead of performing DSE itself, particularly for large-scale applications such as Deep Neural Networks, where DSE can be time-intensive and resource-demanding. By developing more efficient exploration techniques and expanding generalizability across hardware platforms, future research can help overcome the current limitations of DSE methodologies employed for AxC systems.

In addition to discussing the strengths of various ML- and EA-based DSE methods, it is important to discuss the computational challenges and resource demands associated with performing DSE for approximate designs. When performing DSE for approximate designs, computational complexity and hardware resource requirements vary significantly, depending on the scope of the exploration and the chosen methods. Some studies face challenges related to time, memory utilization, and processing power, especially when exploring large design spaces with numerous approximate versions of hardware or software designs. However, not all reviewed studies provide detailed reports on these parameters, making direct comparisons difficult.

Where exploration time or resource usage is discussed, the computational burden is influenced by factors such as the type of search algorithm, design space size, and the target hardware. For instance, DSE-targeting FPGA designs may differ in complexity and time requirements compared with DSE-targeting ASICs or GPUs, even when similar methodologies are used.

Several studies employ pruning techniques to reduce the design space size, which helps facilitate the exploration process and lowers the computational demands. Such strategies are particularly prominent in studies using custom algorithms, as discussed in Section 4.3. Pruning can significantly reduce exploration time by focusing on the most promising candidate designs.

However, due to inconsistent reporting on exploration time and hardware resource consumption across different studies, drawing broad conclusions about the computational costs of different DSE methods remains challenging. This gap presents an opportunity for future work to provide more detailed comparisons of resource requirements across DSE methodologies.

Author Contributions: S.S. and A.P. contributed to the conceptualization, design, and methodology of the study. S.S. performed the literature review and analysis, while A.P. assisted in drafting and refining the manuscript. Both authors engaged in discussions regarding the findings and conclusions of the survey. B.D., I.O., A.B., A.S. and S.D.C. contributed to supervision, paper writing, project administration, and funding acquisition. All authors have read and agreed to the published version of the manuscript.

Funding: This research has received funding from the APROPOS project in the European Union's Horizon 2020 research and innovation program under the Marie Skłodowska-Curie grant agreement No. 956090. Views and opinions expressed are, however, those of the author(s) only and do not necessarily reflect those of the European Union. Neither the European Union nor the granting authority can be held responsible for them.

Data Availability Statement: Not applicable.

Conflicts of Interest: The authors declare no conflicts of interest. The funders had no role in the design of the study, in the collection, analyses, or interpretation of data, in the writing of the manuscript, or in the decision to publish the results.

References

1. Mittal, S. A survey of techniques for approximate computing. *ACM Comput. Surv. (CSUR)* **2016**, *48*, 1–33. [CrossRef]
2. Jones, N. How to stop data centres from gobbling up the world's electricity. *Nature* **2018**, *561*, 163–166. [CrossRef] [PubMed]
3. Jiang, H.; Liu, C.; Liu, L.; Lombardi, F.; Han, J. A Review, Classification, and Comparative Evaluation of Approximate Arithmetic Circuits. *J. Emerg. Technol. Comput. Syst.* **2017**, *13*, 60. [CrossRef]
4. Venkataramani, S.; Chakradhar, S.T.; Roy, K.; Raghunathan, A. Approximate computing and the quest for computing efficiency. In Proceedings of the 2015 52nd ACM/EDAC/IEEE Design Automation Conference (DAC), San Francisco, CA, USA, 8–12 June 2015; pp. 1–6. [CrossRef]
5. Chippa, V.K.; Chakradhar, S.T.; Roy, K.; Raghunathan, A. Analysis and characterization of inherent application resilience for approximate computing. In Proceedings of the 50th Annual Design Automation Conference, Austin, TX, USA, 29 May–7 June 2013; pp. 1–9.
6. Esmaeilzadeh, H.; Sampson, A.; Ceze, L.; Burger, D. Neural Acceleration for General-Purpose Approximate Programs. In Proceedings of the 2012 45th Annual IEEE/ACM International Symposium on Microarchitecture, Vancouver, BC, Canada, 1–5 December 2012; pp. 449–460. [CrossRef]
7. Han, J.; Orshansky, M. Approximate computing: An emerging paradigm for energy-efficient design. In Proceedings of the 2013 18th IEEE European Test Symposium (ETS), Avignon, France, 27–30 May 2013; pp. 1–6. [CrossRef]
8. Traiola, M.; Virazel, A.; Girard, P.; Barbareschi, M.; Bosio, A. Testing approximate digital circuits: Challenges and opportunities. In Proceedings of the 2018 IEEE 19th Latin-American Test Symposium (LATS), Sao Paulo, Brazil, 12–14 March 2018; pp. 1–6. [CrossRef]
9. Bosio, A.; Ménard, D.; Sentieys, O. (Eds.) *Approximate Computing Techniques*; Springer International Publishing: Berlin/Heidelberg, Germany, 2022. [CrossRef]
10. Traiola, M.; Savino, A.; Barbareschi, M.; Di Carlo, S.; Bosio, A. Predicting the impact of functional approximation: From component-to application-level. In Proceedings of the 2018 IEEE 24th International Symposium on On-Line Testing And Robust System Design (IOLTS), Platja d'Aro, Spain, 2–4 July 2018; pp. 61–64.
11. Traiola, M.; Savino, A.; Di Carlo, S. Probabilistic estimation of the application-level impact of precision scaling in approximate computing applications. *Microelectron. Reliab.* **2019**, *102*, 113309. [CrossRef]
12. Mrazek, V.; Hanif, M.A.; Vasicek, Z.; Sekanina, L.; Shafique, M. autoax: An automatic design space exploration and circuit building methodology utilizing libraries of approximate components. In Proceedings of the 56th Annual Design Automation Conference 2019, Las Vegas, NV, USA, 2–6 June 2019; pp. 1–6.
13. Sekanina, L.; Vasicek, Z.; Mrazek, V. Automated search-based functional approximation for digital circuits. In *Approximate Circuits: Methodologies and CAD*; Springer: Cham, Switzerland, 2019; pp. 175–203.
14. Barbareschi, M.; Barone, S.; Mazzocca, N.; Moriconi, A. Design Space Exploration Tools. In *Approximate Computing Techniques: From Component-to Application-Level*; Springer: Berlin/Heidelberg, Germany, 2022; pp. 215–259.
15. Xu, Q.; Mytkowicz, T.; Kim, N.S. Approximate computing: A survey. *IEEE Des. Test* **2015**, *33*, 8–22. [CrossRef]

16. Rodrigues, G.; Lima Kastensmidt, F.; Bosio, A. Survey on Approximate Computing and Its Intrinsic Fault Tolerance. *Electronics* **2020**, *9*, 557. [CrossRef]
17. Li, S.; Park, S.; Mahlke, S. Sculptor: Flexible approximation with selective dynamic loop perforation. In Proceedings of the 2018 International Conference on Supercomputing, Beijing, China, 12–15 June 2018; pp. 341–351.
18. Goiri, I.; Bianchini, R.; Nagarakatte, S.; Nguyen, T.D. Approxhadoop: Bringing approximations to mapreduce frameworks. In Proceedings of the Twentieth International Conference on Architectural Support for Programming Languages and Operating Systems, Istanbul, Turkey, 14–18 March 2015; pp. 383–397.
19. Vassiliadis, V.; Parasyris, K.; Chalios, C.; Antonopoulos, C.D.; Lalis, S.; Bellas, N.; Vandierendonck, H.; Nikolopoulos, D.S. A programming model and runtime system for significance-aware energy-efficient computing. *ACM SIGPLAN Not.* **2015**, *50*, 275–276. [CrossRef]
20. Raha, A.; Venkataramani, S.; Raghunathan, V.; Raghunathan, A. Quality configurable reduce-and-rank for energy efficient approximate computing. In Proceedings of the 2015 IEEE Design, Automation & Test in Europe Conference & Exhibition (DATE), Grenoble, France, 9–13 March 2015; pp. 665–670.
21. Rubio-González, C.; Nguyen, C.; Nguyen, H.D.; Demmel, J.; Kahan, W.; Sen, K.; Bailey, D.H.; Iancu, C.; Hough, D. Precimonious: Tuning assistant for floating-point precision. In Proceedings of the International Conference on High Performance Computing, Networking, Storage and Analysis, Denver, CO, USA, 17–22 November 2013; pp. 1–12.
22. Hsiao, C.C.; Chu, S.L.; Chen, C.Y. Energy-aware hybrid precision selection framework for mobile GPUs. *Comput. Graph.* **2013**, *37*, 431–444. [CrossRef]
23. Sinha, S.; Zhang, W. Low-Power FPGA Design Using Memoization-Based Approximate Computing. *IEEE Trans. Very Large Scale Integr. (VLSI) Syst.* **2016**, *24*, 2665–2678. [CrossRef]
24. Sampaio, F.; Shafique, M.; Zatt, B.; Bampi, S.; Henkel, J. Approximation-aware multi-level cells STT-RAM cache architecture. In Proceedings of the 2015 IEEE International Conference on Compilers, Architecture and Synthesis for Embedded Systems (CASES), Amsterdam, The Netherlands, 4–9 October 2015; pp. 79–88.
25. Tian, Y.; Zhang, Q.; Wang, T.; Yuan, F.; Xu, Q. ApproxMA: Approximate memory access for dynamic precision scaling. In Proceedings of the 25th Edition on Great Lakes Symposium on VLSI, Pittsburgh, PA, USA, 20–22 May 2015; pp. 337–342.
26. Ranjan, A.; Venkataramani, S.; Fong, X.; Roy, K.; Raghunathan, A. Approximate storage for energy efficient spintronic memories. In Proceedings of the 52nd Annual Design Automation Conference, San Francisco, CA, USA, 7–11 June 2015; pp. 1–6.
27. Shafique, M.; Ahmad, W.; Hafiz, R.; Henkel, J. A low latency generic accuracy configurable adder. In Proceedings of the 52nd Annual Design Automation Conference, San Francisco, CA, USA, 7–11 June 2015. DAC '15. [CrossRef]
28. St. Amant, R.; Yazdanbakhsh, A.; Park, J.; Thwaites, B.; Esmaeilzadeh, H.; Hassibi, A.; Ceze, L.; Burger, D. General-purpose code acceleration with limited-precision analog computation. *ACM SIGARCH Comput. Archit. News* **2014**, *42*, 505–516. [CrossRef]
29. Zhang, Q.; Wang, T.; Tian, Y.; Yuan, F.; Xu, Q. ApproxANN: An approximate computing framework for artificial neural network. In Proceedings of the 2015 IEEE Design, Automation & Test in Europe Conference & Exhibition (DATE), Grenoble, France, 9–13 March 2015; pp. 701–706.
30. Mohapatra, D.; Chippa, V.K.; Raghunathan, A.; Roy, K. Design of voltage-scalable meta-functions for approximate computing. In Proceedings of the 2011 Design, Automation & Test in Europe, Grenoble, France, 14–18 March 2011; pp. 1–6. [CrossRef]
31. Kumar, V.I.; Kapat, S. Per-Core Configurable Power Supply for Multi-Core Processors with Ultra-Fast DVS Voltage Transitions. In Proceedings of the 2022 IEEE Applied Power Electronics Conference and Exposition (APEC), Houston, TX, USA, 20–24 March 2022; pp. 1028–1034. [CrossRef]
32. Senobari, A.; Vafaei, J.; Akbari, O.; Hochberger, C.; Shafique, M. A Quality-Aware Voltage Overscaling Framework to Improve the Energy Efficiency and Lifetime of TPUs based on Statistical Error Modeling. *IEEE Access* **2024**, *12*, 92181–92197. [CrossRef]
33. Chatzitsompanis, G.; Karakonstantis, G. On the Facilitation of Voltage Over-Scaling and Minimization of Timing Errors in Floating-Point Multipliers. In Proceedings of the 2023 IEEE 29th International Symposium on On-Line Testing and Robust System Design (IOLTS), Crete, Greece, 3–5 July 2023; pp. 1–7. [CrossRef]
34. Leon, V.; Hanif, M.A.; Armeniakos, G.; Jiao, X.; Shafique, M.; Pekmestzi, K.; Soudris, D. Approximate computing survey, Part II: Application-specific & architectural approximation techniques and applications. *arXiv* **2023**, arXiv:2307.11128.
35. Seliya, N.; Khoshgoftaar, T.M.; Van Hulse, J. A study on the relationships of classifier performance metrics. In Proceedings of the 2009 21st IEEE International Conference on Tools with Artificial Intelligence, Newark, NJ, USA, 2–4 November 2009; pp. 59–66.
36. Schafer, B.C.; Wang, Z. High-level synthesis design space exploration: Past, present, and future. *IEEE Trans. Comput.-Aided Des. Integr. Circuits Syst.* **2019**, *39*, 2628–2639. [CrossRef]
37. Saeedi, S.; Savino, A.; Di Calro, S. Design Space Exploration of Approximate Computing Techniques with a Reinforcement Learning Approach. In Proceedings of the 2023 53rd Annual IEEE/IFIP International Conference on Dependable Systems and Networks Workshops (DSN-W), Porto, Portugal, 27–30 June 2023; pp. 167–170.
38. Pimentel, A.D. Methodologies for Design Space Exploration. In *Handbook of Computer Architecture*; Chattopadhyay, A., Ed.; Springer Nature: Singapore, 2022; pp. 1–31. [CrossRef]
39. Wolf, W.; Jerraya, A.A.; Martin, G. Multiprocessor system-on-chip (MPSoC) technology. *IEEE Trans. Comput.-Aided Des. Integr. Circuits Syst.* **2008**, *27*, 1701–1713. [CrossRef]
40. Santos, L.; Rigo, S.; Azevedo, R.; Araujo, G. Electronic System Level Design. In *Electronic System Level Design: An Open-Source Approach*; Rigo, S., Azevedo, R., Santos, L., Eds.; Springer: Dordrecht, The Netherlands, 2011; pp. 3–10. [CrossRef]

41. Keutzer, K.; Newton, A.R.; Rabaey, J.M.; Sangiovanni-Vincentelli, A. System-level design: Orthogonalization of concerns and platform-based design. *IEEE Trans. Comput.-Aided Des. Integr. Circuits Syst.* **2000**, *19*, 1523–1543. [CrossRef]
42. Sangiovanni-Vincentelli, A.; Martin, G. Platform-based design and software design methodology for embedded systems. *IEEE Des. Test Comput.* **2001**, *18*, 23–33. [CrossRef]
43. Gries, M. Methods for evaluating and covering the design space during early design development. *Integration* **2004**, *38*, 131–183. [CrossRef]
44. Pimentel, A.D. Exploring exploration: A tutorial introduction to embedded systems design space exploration. *IEEE Des. Test* **2016**, *34*, 77–90. [CrossRef]
45. Singh, A.K.; Shafique, M.; Kumar, A.; Henkel, J. Mapping on multi/many-core systems: Survey of current and emerging trends. In Proceedings of the 50th Annual Design Automation Conference, Austin, TX, USA, 29 May–7 June 2013; pp. 1–10.
46. Palar, P.S.; Dwianto, Y.B.; Zuhal, L.R.; Morlier, J.; Shimoyama, K.; Obayashi, S. Multi-objective design space exploration using explainable surrogate models. *Struct. Multidiscip. Optim.* **2024**, *67*, 38. [CrossRef]
47. Smithson, S.C.; Yang, G.; Gross, W.J.; Meyer, B.H. Neural networks designing neural networks: Multi-objective hyper-parameter optimization. In Proceedings of the 2016 IEEE/ACM International Conference on Computer-Aided Design (ICCAD), Austin, TX, USA, 7–10 November 2016; pp. 1–8. [CrossRef]
48. Dupuis, E.; Novo, D.; O'Connor, I.; Bosio, A. On the Automatic Exploration of Weight Sharing for Deep Neural Network Compression. In Proceedings of the 2020 Design, Automation Test in Europe Conference Exhibition (DATE), Grenoble, France, 9–13 March 2020; pp. 1319–1322. [CrossRef]
49. Savino, A.; Traiola, M.; Carlo, S.D.; Bosio, A. Efficient Neural Network Approximation via Bayesian Reasoning. In Proceedings of the 2021 24th International Symposium on Design and Diagnostics of Electronic Circuits Systems (DDECS), Vienna, Austria, 7–9 April 2021; pp. 45–50. [CrossRef]
50. Hashemi, S.; Tann, H.; Buttafuoco, F.; Reda, S. Approximate computing for biometric security systems: A case study on iris scanning. In Proceedings of the 2018 Design, Automation & Test in Europe Conference & Exhibition (DATE), Dresden, Germany, 19–23 March 2018; pp. 319–324.
51. Ullah, S.; Sahoo, S.S.; Kumar, A. Clapped: A design framework for implementing cross-layer approximation in fpga-based embedded systems. In Proceedings of the 2021 58th ACM/IEEE Design Automation Conference (DAC), San Francisco, CA, USA, 5–9 December 2021; pp. 475–480.
52. Rajput, M.A.; Alyami, S.; Ahmed, Q.A.; Alshahrani, H.; Asiri, Y.; Shaikh, A. Improved Learning-Based Design Space Exploration for Approximate Instance Generation. *IEEE Access* **2023**, *11*, 18291–18299. [CrossRef]
53. Awais, M.; Ghasemzadeh Mohammadi, H.; Platzner, M. LDAX: A learning-based fast design space exploration framework for approximate circuit synthesis. In Proceedings of the 2021 on Great Lakes Symposium on VLSI, Virtual Event, 22–25 June 2021; pp. 27–32.
54. Hoffmann, H. Jouleguard: Energy guarantees for approximate applications. In Proceedings of the 25th Symposium on Operating Systems Principles, Monterey, CA, USA, 4–7 October 2015; pp. 198–214.
55. Elthakeb, A.T.; Pilligundla, P.; Mireshghallah, F.; Yazdanbakhsh, A.; Esmaeilzadeh, H. Releq: A reinforcement learning approach for automatic deep quantization of neural networks. *IEEE Micro* **2020**, *40*, 37–45. [CrossRef] [PubMed]
56. Manuel, M.; Kreddig, A.; Conrady, S.; Doan, N.A.V.; Stechele, W. Model-based design space exploration for approximate image processing on FPGA. In Proceedings of the 2020 IEEE Nordic Circuits and Systems Conference (NorCAS), Oslo, Norway, 27–28 October 2020; pp. 1–7.
57. Kreddig, A.; Conrady, S.; Manuel, M.; Stechele, W. A Framework for Hardware-Accelerated Design Space Exploration for Approximate Computing on FPGA. In Proceedings of the 2021 IEEE 24th Euromicro Conference on Digital System Design (DSD), Palermo, Italy, 1–3 September 2021; pp. 1–8.
58. Prabakaran, B.S.; Mrazek, V.; Vasicek, Z.; Sekanina, L.; Shafique, M. Xel-FPGAs: An End-to-End Automated Exploration Framework for Approximate Accelerators in FPGA-Based Systems. In Proceedings of the 2023 IEEE/ACM International Conference on Computer Aided Design (ICCAD), San Francisco, CA, USA, 28 October–2 November 2023; pp. 1–9.
59. Barbareschi, M.; Barone, S.; Bosio, A.; Han, J.; Traiola, M. A Genetic-algorithm-based Approach to the Design of DCT Hardware Accelerators. *ACM J. Emerg. Technol. Comput. Syst. (JETC)* **2022**, *18*, 1–25. [CrossRef]
60. Park, J.; Ni, K.; Zhang, X.; Esmaeilzadeh, H.; Naik, M. Expectation-oriented framework for automating approximate programming. In Proceedings of the Workshop on Approximate Computing Across the System Stack (WACAS), Salt Lake City, UT, USA, 2 March 2014.
61. Pinos, M.; Mrazek, V.; Vaverka, F.; Vasicek, Z.; Sekanina, L. Acceleration techniques for automated design of approximate convolutional neural networks. *IEEE J. Emerg. Sel. Top. Circuits Syst.* **2023**, *13*, 212–224. [CrossRef]
62. Shafique, M.; Hafiz, R.; Rehman, S.; El-Harouni, W.; Henkel, J. Cross-layer approximate computing: From logic to architectures. In Proceedings of the 53rd Annual Design Automation Conference, Austin, TX, USA, 5–9 June 2016; pp. 1–6.
63. Fu, Y.; You, H.; Zhao, Y.; Wang, Y.; Li, C.; Gopalakrishnan, K.; Wang, Z.; Lin, Y. Fractrain: Fractionally squeezing bit savings both temporally and spatially for efficient dnn training. *Adv. Neural Inf. Process. Syst.* **2020**, *33*, 12127–12139.
64. Jain, S.; Venkataramani, S.; Raghunathan, A. Approximation through logic isolation for the design of quality configurable circuits. In Proceedings of the 2016 IEEE Design, Automation & Test in Europe Conference & Exhibition (DATE), Dresden, Germany, 14–18 March 2016; pp. 612–617.

65. Paim, G.; Rocha, L.M.G.; Amrouch, H.; da Costa, E.A.C.; Bampi, S.; Henkel, J. A cross-layer gate-level-to-application co-simulation for design space exploration of approximate circuits in HEVC video encoders. *IEEE Trans. Circuits Syst. Video Technol.* **2019**, *30*, 3814–3828. [CrossRef]
66. Alan, T.; Gerstlauer, A.; Henkel, J. Cross-layer approximate hardware synthesis for runtime configurable accuracy. *IEEE Trans. Very Large Scale Integr. (VLSI) Syst.* **2021**, *29*, 1231–1243. [CrossRef]
67. Xu, S.; Schafer, B.C. Exposing approximate computing optimizations at different levels: From behavioral to gate-level. *IEEE Trans. Very Large Scale Integr. (VLSI) Syst.* **2017**, *25*, 3077–3088. [CrossRef]
68. Castro-Godínez, J.; Mateus-Vargas, J.; Shafique, M.; Henkel, J. AxHLS: Design space exploration and high-level synthesis of approximate accelerators using approximate functional units and analytical models. In Proceedings of the 39th International Conference on Computer-Aided Design, San Diego, CA, USA, 2–5 November 2020; pp. 1–9.
69. Zervakis, G.; Xydis, S.; Soudris, D.; Pekmestzi, K. Multi-level approximate accelerator synthesis under voltage island constraints. *IEEE Trans. Circuits Syst. II Express Briefs* **2018**, *66*, 607–611. [CrossRef]
70. Kim, Y.; Venkataramani, S.; Roy, K.; Raghunathan, A. Designing approximate circuits using clock overgating. In Proceedings of the 53rd Annual Design Automation Conference, Austin, TX, USA, 5–9 June 2016; pp. 1–6.
71. Venkataramani, S.; Srinivasan, V.; Wang, W.; Sen, S.; Zhang, J.; Agrawal, A.; Kar, M.; Jain, S.; Mannari, A.; Tran, H.; et al. RaPiD: AI accelerator for ultra-low precision training and inference. In Proceedings of the 2021 ACM/IEEE 48th Annual International Symposium on Computer Architecture (ISCA), Valencia, Spain, 14–18 June 2021; pp. 153–166.
72. Venkataramani, S.; Sun, X.; Wang, N.; Chen, C.Y.; Choi, J.; Kang, M.; Agarwal, A.; Oh, J.; Jain, S.; Babinsky, T.; et al. Efficient AI system design with cross-layer approximate computing. *Proc. IEEE* **2020**, *108*, 2232–2250. [CrossRef]
73. Mahajan, D.; Yazdanbakhsh, A.; Park, J.; Thwaites, B.; Esmaeilzadeh, H. Towards statistical guarantees in controlling quality tradeoffs for approximate acceleration. *ACM SIGARCH Comput. Archit. News* **2016**, *44*, 66–77. [CrossRef]
74. Savino, A.; Portolan, M.; Leveugle, R.; Di Carlo, S. Approximate computing design exploration through data lifetime metrics. In Proceedings of the 2019 IEEE European Test Symposium (ETS), Baden-Baden, Germany, 27–31 May 2019; pp. 1–7.
75. Yazdanbakhsh, A.; Pekhimenko, G.; Thwaites, B.; Esmaeilzadeh, H.; Mutlu, O.; Mowry, T.C. RFVP: Rollback-free value prediction with safe-to-approximate loads. *ACM Trans. Archit. Code Optim. (TACO)* **2016**, *12*, 1–26. [CrossRef]
76. Echavarria, J.; Wildermann, S.; Teich, J. Approximate logic synthesis of very large boolean networks. In Proceedings of the 2021 Design, Automation & Test in Europe Conference & Exhibition (DATE), Grenoble, France, 1–5 February 2021; pp. 1552–1557.
77. Miao, J.; Gerstlauer, A.; Orshansky, M. Multi-level approximate logic synthesis under general error constraints. In Proceedings of the 2014 IEEE/ACM International Conference on Computer-Aided Design (ICCAD), San Jose, CA, USA, 2–6 November 2014; pp. 504–510.
78. Traiola, M.; Echavarria, J.; Bosio, A.; Teich, J.; O'Connor, I. Design Space Exploration of Approximation-Based Quadruple Modular Redundancy Circuits. In Proceedings of the 2021 IEEE/ACM International Conference On Computer Aided Design (ICCAD), Munich, Germany, 1–4 November 2021; pp. 1–9.
79. Ma, J.; Hashemi, S.; Reda, S. Approximate logic synthesis using boolean matrix factorization. *IEEE Trans. Comput.-Aided Des. Integr. Circuits Syst.* **2021**, *41*, 15–28. [CrossRef]
80. Echavarria, J.; Keszocze, O.; Teich, J. Probability-based dse of approximated lut-based fpga designs. In Proceedings of the 2022 IEEE 15th Dallas Circuit And System Conference (DCAS), Dallas, TX, USA, 17–19 June 2022; pp. 1–5.