Politecnico di Torino

ScuDo
Scuola di Dottorato ~ Doctoral School
WHAT YOU ARE, TAKES YOU FAR

Doctoral Dissertation
Doctoral Program in Computer and Control Engineering (37^{th} cycle)

Fast and Accurate Prediction of the Impact of Approximate Operators on a Complex Computation

Sepide Saeedi

* * * * *

Supervisors
Prof. Stefano Di Carlo, Supervisor
Prof. Alessandro Savino, Co-supervisor

Politecnico di Torino
May 15, 2026

I hereby declare that the contents and organization of this dissertation constitute my own original work and do not compromise in any way the rights of third parties, including those relating to the security of personal data.

. . . . . .

Sepide Saeedi

Turin, May 15, 2026

Summary

Approximate Computing (AxC) has emerged as a powerful paradigm for improving computational efficiency by strategically introducing acceptable levels of inaccuracy in computations, thereby reducing power consumption, execution time, and hardware complexity. However, identifying optimal approximation configurations requires navigating a high-dimensional design space, posing many methodological challenges. This thesis addresses these challenges through Design Space Exploration (DSE) approaches aimed at enabling systematic application of AxC techniques to computational systems.

Firstly, we introduce an Interval Arithmetic (IA)-based methodology tailored to evaluate the impact of precision reduction in Spiking Neural Networks (SNNs). The IA-based model provides an analytical estimate of how bit-width truncation errors propagate through neuron computations, enabling the identification of acceptable approximation levels with limited degradation in accuracy under the evaluated settings. This approach enables faster exploration of precision-reduction strategies, striving to reduce exploration time relative to exhaustive search methods in terms of computational efficiency and speed. This method is further enhanced by integrating "watchers," mechanisms that facilitate fine-grained, neuron-specific precision adjustments, aiming to limit memory usage and computational overhead while maintaining the desired classification accuracy.

Secondly, we propose a Reinforcement Learning (RL)-based approach for exploring approximation opportunities in conventional CPU-based applications. Our method automates the selective replacement of arithmetic operations with approximate alternatives from a predefined operator library. The RL agent is utilized to perform a DSE with the aim of finding a trade-off among multiple objectives, including accuracy, power consumption, and execution time. This DSE goal is realized by the RL agent, iteratively evaluating design choices and adapting based on performance feedback. Experimental evaluations on representative benchmarks, such as matrix multiplication and FIR filters, suggest that the RL methodology can identify configurations that reduce power and computation time subject to the imposed accuracy constraints.

Collectively, these methodologies were proposed with the aim of providing practical solutions for systematically exploring approximation strategies across the studied computational domains. The contributions of this thesis can support the deployment of AxC techniques and motivate future investigations in the design of resource-efficient computing systems.

Acknowledgements

I would like to express my sincere gratitude to my supervisors, Stefano Di Carlo and Alessandro Savino, for their continuous guidance, insightful feedback, and support throughout this PhD journey. Their expertise and perspective have been essential in shaping the direction and quality of this work.

This project has received funding from the European Union's Horizon 2020 (H2020) Marie Skłodowska-Curie Innovative Training Networks (ITN) H2020-MSCA-ITN-2020 call, under the Grant Agreement no. 956090. I gratefully acknowledge this support, which made this research possible. I also express my appreciation to Politecnico di Torino and the Department of Control and Computer Engineering for providing the necessary resources and infrastructure to carry out this study.

I would also like to acknowledge the many people who contributed to my research and the technical aspects of the project. Professors from Politecnico di Torino, the APROPOS project, and other collaborators, including Alberto Bosio, Paolo Bernardi, Bastien Deveautour, Marcello Traiola, Jari Nurmi, Aleksandr Ometov, and many others, whose help has been invaluable. I am also sincerely grateful to the thesis defense committee members, particularly Angeliki Kritikakou and Nima TaheriNejad, for reviewing my thesis and providing thoughtful and constructive feedback.

I would like to thank the members of Lab 6, the SMILIES research group, for providing a stimulating and supportive research environment. The discussions, exchanges of ideas, and collaborative atmosphere within the group have greatly enriched this work. Additionally, I greatly value the collaborations and friendships with the early-stage researchers from the APROPOS project. Their insights and collaborative spirit have significantly contributed to the development of this work.

Finally, I am deeply grateful to my family and friends for their unwavering support, encouragement, understanding, and patience during this journey.

List of Acronyms

AI Artificial Intelligence. 79, 83, 92

ANN Artificial Neural Network. 23, 49

ASIC Application-Specific Integrated Circuit. 8, 14, 20, 21, 22, 23, 30, 79, 80, 81, 82, 83, 84, 85, 86, 88, 89, 90, 91, 92, 94, 97

AWCE Absolute Worst-Case Error. 17

AxC Approximate Computing. xiii, xv, 2, 3, 4, 5, 6, 7, 9, 10, 11, 12, 13, 14, 15, 16, 19, 20, 22, 23, 24, 25, 27, 48, 49, 51, 53, 54, 55, 57, 60, 61, 63, 64, 65, 66, 72, 73, 75, 76, 85, 88, 89, 94

BER Bit Error Ratio. 17, 98, 100

BMF Boolean Matrix Factorization. 22, 94

CNN Convolutional Neural Network. 21, 81, 97

CPU Central Processing Unit. 14, 20, 22, 23, 51, 71, 80, 81, 84, 87, 88, 93

DCT Discrete Cosine Transform. 81, 82, 88, 90, 91

DNN Deep Neural Network. 7, 20, 22, 23, 71, 80, 81, 82, 87, 90, 96, 98, 99

DSE Design Space Exploration. xiii, xiv, 5, 6, 7, 8, 9, 10, 11, 14, 15, 19, 20, 21, 22, 23, 24, 25, 36, 38, 39, 41, 42, 45, 47, 48, 49, 51, 54, 55, 56, 57, 60, 61, 62, 64, 65, 66, 70, 71, 72, 75, 76, 79, 80, 81, 84, 85, 88, 89, 94, 95, 96, 98, 100

DSP Digital Signal Processing. 64, 65

DVFS Dynamic Voltage and Frequency Scaling. 14

DVS Dynamic Voltage Scaling. 14

EA Evolutionary Algorithm. xiii, 7, 8, 20, 21, 23, 24, 60, 70, 71, 72, 80, 88, 96

ED Error Distance. 15

EP Error Probability. 15

ER Error Rate. 17, 97, 100

ES Evolutionary Strategy. 21, 23, 81

EXDC External Don't-Care. 22, 94

FFT Fast Fourier Transform. 81, 82, 83, 88, 90, 92

FIR Finite Impulse Response. xv, 63, 64, 66, 67, 69, 70, 71, 72, 75, 79, 80, 82, 83, 86, 87, 90, 91, 92

FN False Negative. 18

FP False Positive. 18

FPGA Field-Programmable Gate Array. 8, 14, 20, 21, 22, 23, 30, 31, 71, 79, 80, 81, 84, 85, 86, 88, 89, 94, 95, 97

GA Genetic Algorithm. 7, 8, 20, 21, 23, 70, 72, 80, 81, 82

GD Gradient Descent. 21, 83

GPU Graphics Processing Unit. 21, 22, 23, 81, 84, 89, 93

HD Hamming Distance. 17, 95, 100

HEVC High-Efficiency Video Coding. 81, 82

HLS High-Level Synthesis. 70, 82, 91

IA Interval Arithmetic. xiii, 8, 9, 10, 13, 27, 30, 32, 33, 34, 35, 36, 37, 38, 39, 40, 41, 43, 47, 48, 49, 75, 76

ICT Information and Communication Technology. 1

ISA Instruction Set Architecture. 13

LUT Look-Up Table. 23, 88, 89, 95, 97, 100

MAE Mean Absolute Error. 4, 8, 58, 68, 95, 96, 98, 100

MBO Multi-objective Bayesian Optimization. 20, 23, 71, 79

MCM Multiple Constant Multiplier. 81, 88

MCTS Monte Carlo Tree Search. 20, 23, 71, 79

MDP Markov Decision Process. 53, 54, 55, 56

MED Mean Error Distance. 4, 14, 15, 99

ML Machine Learning. xiii, 3, 6, 8, 9, 20, 21, 22, 23, 24, 52, 63, 70, 71, 73, 79, 80, 82, 83, 84, 85, 91, 95

MOP Multi-objective Optimization Problem. 23

MPD Mean Pixel Difference. 8, 18, 97

MPSoC Multi-Processor System-on-Chip. 19

MRED Mean Relative Error Distance. 14, 15, 57, 61, 62, 95, 98, 99

MSE Mean Squared Error. 4, 8, 14, 17

NAS Neural Architecture Search. 54, 81

NLP Natural Language Processing. 83

NN Neural Network. 3, 6, 8, 13, 20, 23, 24, 49, 63, 65, 76, 92

NPU Neural Processing Unit. 4, 22, 83, 92

NSGA-II Non-dominated Sorted Genetic Algorithm-II. 7, 8, 21, 23, 81, 84, 85

PDF Probability Density Function. 4

PDP Power-Delay-Product. 99

PMF Probability Mass Function. 4

PSNR Peak Signal-to-Noise Ratio. 4, 8, 17, 18, 23, 95, 96, 97, 98, 99

QOR Quality of Results. 95, 97

RFVP Rollback-Free Value Prediction. 93

RL Reinforcement Learning. xv, 7, 8, 9, 10, 14, 20, 21, 23, 24, 51, 52, 53, 54, 55, 56, 57, 58, 59, 60, 61, 62, 63, 64, 65, 66, 67, 70, 71, 72, 73, 75, 76, 79, 80

RMSE Root Mean Squared Error. 4, 99

SAD Sum of Absolute Differences. 81, 82, 89, 90

SNN Spiking Neural Network. xiii, xv, 8, 9, 10, 13, 23, 24, 27, 28, 29, 30, 31, 32, 33, 34, 36, 37, 38, 39, 40, 41, 43, 45, 48, 49, 75

SSIM Structural Similarity Index Measure. 4, 8, 18, 95

TN True Negative. 18

TP True Positive. 18

TS Tabu Search. 21, 82

VOS Voltage Over Scaling. 4, 14, 22, 91

Contents

List of Tables

List of Figures

Chapter 1

Introduction

1.1 Motivation and Context

The escalating volume of data managed by modern computing systems has placed extreme demands on performance and energy resources. This trend amplifies long-standing challenges such as the memory wall and dark-silicon constraints, where data movement and thermal limits increasingly dominate system-level design choices. Projections estimate that, within the next decade, data centers worldwide will handle approximately fifty times more data, whereas the number of processors will only increase by a factor of ten [8]. Consequently, simply scaling out compute nodes or over-provisioning hardware becomes economically and energetically unsustainable, motivating techniques that co-optimize accuracy with performance and power. This clear mismatch between data growth and processor scaling highlights the limitations of relying on over-provisioned infrastructures alone. In parallel, the energy cost of moving bits across the memory hierarchy often exceeds the cost of computation, so reducing precision and computation can yield outsized system-level savings when aligned with application error tolerance. Alongside these concerns, the energy consumption of Information and Communication Technology (ICT) devices is on a path to nearly 21% of global electricity usage by 2030 [9]. These projections necessitate architectural and algorithmic responses that treat energy as a first-class constraint without compromising the essential quality of service (QoS). Constraints on power and thermal management aggravated by the diminishing returns of technology scaling underscore the urgency of alternative design paradigms aimed at improving energy efficiency.

CMOS technology scaling has historically driven gains in performance by allowing more transistors per chip and higher clock frequencies. However, post-Dennard designs have decoupled frequency from voltage scaling, limiting further energy efficiency improvements through traditional scaling alone. Dennard (constant-field) scaling holds that proportionally reducing MOSFET linear dimensions ($1/\kappa$) together with supply and threshold voltages ($1/\kappa$) preserves electric fields, reduces

capacitance and delay (roughly $1/\kappa$), and keeps power density approximately constant, enabling decades of frequency gains at similar chip power [10]. Although such downscaling initially offered substantial benefits for computational speed and energy efficiency, by the mid-2000s, continued reductions in transistor size encountered fundamental limitations, including leakage currents, restricted supply-voltage and threshold scaling, device variability and reliability concerns, and increased thermal dissipation. The immediate consequence was a post-Dennard era in which further increases in frequency violate power/thermal envelopes and large fractions of a chip must remain dark [11, 12]. At the same time, variability and reliability concerns at advanced nodes complicate worst-case design, further incentivizing application-aware relaxation of exactness. These practical limitations intensify interest in innovative solutions that can deliver high computational throughput while respecting stringent power budgets. Approximate Computing (AxC) has emerged as a compelling response to these challenges, leveraging application-specific tolerance for small inaccuracies to reduce energy consumption and enhance performance. Crucially, AxC shifts the optimization objective from exact bit-level accuracy to meeting application-level accuracy constraints under given power, performance, and area budgets. As diverse application domains, from mobile devices to large-scale data centers, increasingly exhibit resilience to imprecise calculations, AxC methods are poised to fill the gap left by the stagnation of conventional scaling trends. This motivates systematic methodologies to decide where approximation is permissible, how much to approximate, and how to verify compliance with domain-specific accuracy requirements.

In the present era, or more precisely the "Post-Moore" era, the historical exponential growth in transistor density (Moore's law) and its effect on power and performance have markedly slowed down; per-node frequency and energy gains no longer arrive "for free." Together with the breakdown of Dennard scaling, this implies that continued performance or efficiency improvements cannot rely solely on CMOS scaling any longer, motivating domain-specific accelerators and "More-than-Moore" integration [13]. Beyond the economic and technological limitations mentioned earlier, workload characteristics in contemporary systems have shifted decisively toward sensing, perception, and learning pipelines (e.g., vision, audio, sensor fusion, and data-driven inference). Inputs are intrinsically noisy (sensor noise, compression artifacts, stochastic training), objectives are statistical (loss minimization, confidence thresholds, top-k accuracy), and multiple outputs can be equally acceptable from a user-experience or task-level standpoint. In such contexts, exactness at every bit is not always necessary at the application level, which motivates treating accuracy as a first-class design parameter. In a post-Moore setting, this view complements traditional levers such as voltage-frequency scaling and parallelism: relaxing exactness within bounded, application-aware limits can deliver disproportionately large energy/performance gains before accuracy begins to drop sharply (a qualitative Pareto shape that many workloads exhibit). This

motivation is reinforced by two facts: (i) the energy/time cost of moving and operating on wide, precise data grows quickly with bit-width and memory distance (register → cache → DRAM), and (ii) many task metrics and human perceptual judgments tolerate bounded perturbations, especially when summarized at frame, clip, or decision level. Viewed across the stack, this makes accuracy a tunable "design knob" at multiple layers, number representation and operators at the circuit/architecture level, scheduling and skipping at the compiler/runtime level, and precision/algorithm choices at the application level, as long as application-level accuracy constraints are enforced. These observations naturally raise three questions: *(1) Where and when should approximation be introduced? (2) Which mechanism should realize it at a given layer (reduced precision, inexact operators, computation/memory skipping)? (3) How do we quantify and control end-to-end accuracy while optimizing energy/performance?* [14]

1.2 Approximate Computing and its Foundations

AxC embodies a strategy that selectively relaxes exact numerical correctness in portions of hardware or software, so long as the overall user experience or application-level outcome remains acceptable. This relaxation is governed by explicit accuracy budgets or constraints that bound the permissible deviation of outputs from a precise reference. Many of the workloads dominating current computational activity, such as image filtering, video streaming, Machine Learning (ML), and data analytics, are inherently error-resilient because small deviations in intermediate computations often do not compromise usability [15, 16, 17]. Error resilience typically arises from redundancy, noise tolerance, and perceptual or statistical objectives that aggregate local inaccuracies into globally acceptable outcomes. For instance, slight inaccuracies in pixel intensities can pass unnoticed in image processing applications, and rounding errors in iterative data-mining tasks may not significantly affect final cluster structures. In safety- or mission-critical settings, by contrast, the admissible error bounds must be substantially tighter and accompanied by conservative validation. Similar resilience applies in certain numerical solvers, recommendation systems, and Neural Networks (NNs), where inputs are often already noisy and multiple "close enough" outputs can still be valid [18, 19].

Studies on AxC have categorized approximate techniques at three primary abstraction levels: software-level, architecture-level, and hardware-level [20, 17]. This taxonomy clarifies the levers available to designers and highlights opportunities for cross-layer co-design. At the software level, approximations may be introduced by skipping loop iterations or function calls [21, 22], by reducing data precision [23, 24], or by caching computations using memoization [25]. Compiler support and profiling can further localize approximation to hot spots that have a limited impact

on output quality. At the architecture level, dynamic strategies such as memory-access skipping [26] or entire approximate Neural Processing Units (NPUs) can be deployed [27], while specialized hardware-level approaches feature approximate arithmetic units, Voltage Over Scaling (VOS) [28], or approximate memory. Hardware techniques often deliver the most significant energy gains per operation, while software and architectural approaches provide flexibility and portability across platforms. These methods enable substantial gains in energy and performance by exploiting the fact that many applications and end users do not require perfect bit-level accuracy. The central challenge is to align the chosen technique with the application error sensitivity and to provide guarantees that the system-level accuracy constraints remain satisfied.

In practice, introducing approximation requires a disciplined workflow for accuracy assessment. This workflow can be organized into three stages that connect device-level behavior to application-level accuracy:

1. component or operation error characterization, where computation units and kernels are profiled using error metrics (e.g., Mean Error Distance (MED), Mean Squared Error (MSE)) and error distributions (e.g., Probability Density Functions (PDFs) for continuous errors, Probability Mass Functions (PMFs) for discrete errors) to capture both magnitude and likelihood of error [29, 30].

2. error propagation, where local error is transferred through the computation using analytical models, interval and affine arithmetic for conservative bounds, and accelerated emulation or simulation to keep evaluation time tractable [30].

3. application-level evaluation, where the propagated error is checked against task metrics (e.g., Peak Signal-to-Noise Ratio (PSNR), Structural Similarity Index Measure (SSIM) for vision codecs, accuracy/precision/recall [1] for classifiers, Mean Absolute Error (MAE), MSE, Root Mean Squared Error (RMSE) for numerical kernels) and against user-specified accuracy constraints [31, 30].

The key design challenge is that the prediction of the accuracy degradation must be sufficiently conservative to be acceptable while still fast enough to enable large-scale design space exploration. These metrics mentioned here are explained in detail in Table 2.1 in Chapter 2, and their definition and usage are further explained in [29, 31, 30].

Deep learning applications have become a major driver of AxC. Approximation spans reduced-precision number formats, quantization-aware and mixed-precision inference or training, structural pruning at weight, kernel, or tensor granularity, and weight sharing or compression, which trade precision for gains in throughput, memory footprint, and energy while preserving target accuracy and motivating approximate accelerators and mixed-precision execution paths [17].

Nevertheless, introducing deliberate imprecision entails critical design choices. Designers must set acceptable error ranges, define validation procedures that reflect end-user utility, and localize approximation to computations where applying approximation may yield the best energy/performance trade-offs possible, while keeping accuracy degradation and eventually the output accuracy within acceptable bounds. These decisions benefit from domain knowledge (e.g., perceptual thresholds in vision) and tooling that accelerates evaluation without sacrificing rigor. As objectives extend beyond a single target (e.g., accuracy versus energy/-time), the resulting degrees of freedom make exploration in pursuit of suitable AxC techniques nontrivial and directly motivate principled Design Space Exploration (DSE). The next section details why designing approximate systems is complex and which factors drive that complexity.

1.3 Complexity of Designing Approximate Systems

The pursuit of approximate solutions raises several challenges. First, approximation effects are often non-linear and input-dependent, complicating efforts to predict end-to-end behavior from local changes. One involves determining where approximation yields the best energy or speed improvements without sacrificing essential correctness. This typically requires sensitivity analysis to identify computations with low impact on task-level metrics. This determination often demands insight into application semantics, an understanding of which code blocks are less critical to output accuracy, or knowledge of data distributions that allow certain arithmetic operations to be relaxed. Granularity also matters: coarse-grained approximation may leave savings on the table, while overly fine-grained decisions can inflate exploration cost. Another challenge concerns the selection among a wide variety of AxC techniques, each of which makes different trade-offs and alters specific performance or power parameters. Technique interactions (e.g., reduced precision with approximate multipliers) can produce emergent effects that are difficult to estimate compositionally. Methods such as loop perforation, approximate memory read/write, approximate functional units, or partial product perforation in multipliers each carry different implications for final output accuracy [32, 33]. Hence, designers need robust constraints and bounds that ensure accuracy constraints are respected under representative workloads. Moreover, the effectiveness of an approximation strategy is application-dependent, necessitating verification or modeling at multiple abstraction levels. Validation should combine analytical bounds for conservatism with sampling-based or emulation-based checks for representativeness.

Evaluation itself poses a serious bottleneck because enumerating all possible approximate versions of a design can be prohibitively expensive. The number of

approximate configurations grows combinatorially with the number of sites, available operators, and precision choices [34, 35]. Exhaustive simulation can be done for small circuit blocks like adders or multipliers [36], yet it becomes infeasible for entire NNs or real-time systems. Scalable strategies must therefore prune the space early and prioritize promising regions. Researchers have proposed a variety of solutions: from domain-specific heuristics and Bayesian or ML-based error estimators [37, 38], to partial verification that prunes low-value approximation paths early on. Surrogate models and transfer learning can further reduce evaluation cost by reusing knowledge across inputs or design variants. These sophisticated strategies converge on the same goal of identifying a set of approximate configurations that optimize power or performance while respecting user-defined accuracy constraints. Ultimately, success hinges on combining fast accuracy prediction with adaptive search to navigate the vast design space effectively.

Cross-layer interactions multiply the complexity of evaluating the effect of applying approximation. For example, approximation choices at the number representation level can influence the design of computation units and the feasibility of inexact operators; both, in turn, affect memory access and scheduling. When the design space grows combinatorially, the DSE time and complexity increase, since exploring all the possible choices for approximation becomes computationally infeasible within a reasonable time. Two considerations are therefore pivotal: *(a)* fast, conservative accuracy models to prune the search space, and *(b)* adaptive search that focuses effort on promising regions of the design space and localizes approximation decisions.

1.4 Importance of Design Space Exploration

DSE has emerged as an essential approach for systematically analyzing and comparing potential approximate configurations. Formulating the problem explicitly (e.g., as constrained or multi-objective optimization) enables methodical exploration with clear stopping criteria and guarantees. Traditional embedded and high-level system design methodologies already rely on DSE to balance objectives such as performance, power, and area [39, 40], and AxC extends these objectives to include accuracy. Incorporating accuracy as a constraint or optimization objective yields richer Pareto surfaces that capture trade-offs unattainable with exact designs alone. Once accuracy is introduced into the optimization problem, the trade-off surfaces can become multi-dimensional: a single best solution rarely exists, and designers often seek a Pareto front of non-dominated configurations [41]. Decision-makers can then choose operating points that meet system-level budgets or application-specific QoS targets. Each point along this frontier represents a unique balance among energy consumption, performance metrics, area overhead, and permitted error margins for accuracy. Thus, DSE provides both exploration

efficiency and actionable design insights for approximate systems.

While applying AxC techniques to the design, the exploration problem magnifies further because each level of approximation can be applied independently or in combination. This combinatorial growth necessitates search strategies that are both sample-efficient and accuracy-aware. Software transformations such as loop perforation might be integrated with approximate multipliers or approximate memory read/write operations, creating a combinatorial explosion of possibilities [42, 43]. In general, applying multiple approximations in different levels of abstraction can increase the design space size [17]. Consequently, scalable DSE must integrate surrogate accuracy models, early infeasibility detection, and adaptive sampling.

High-level modeling or partial evaluation can guide which approximations are feasible. Such models act as filters to reject designs that are unlikely to satisfy accuracy constraints, thereby saving evaluation time. Multi-objective optimization algorithms, such as Genetic Algorithms (GAs), Non-dominated Sorted Genetic Algorithm-II (NSGA-II), Reinforcement Learning (RL), or custom heuristics, have been explored to navigate this vast space efficiently, as surveyed extensively in [2]. Algorithm selection should reflect the structure of the design space (e.g., discreteness, non-convexity) and the availability of gradient information or surrogates. Several main classes of DSE strategies have been introduced in the literature, each with distinct benefits. Evolutionary Algorithms (EAs), for example, work well for hardware-level approximation when searching for Pareto-optimal circuit designs under area, power, and accuracy constraints [44, 36]. They excel when objective functions are black-box and highly non-linear. RL can systematically explore approximate software or hardware parameters, adjusting the level or placement of the approximation in real time, as shown for Iris scanning [45] or Deep Neural Network (DNN) quantization [46]. Policy-based exploration is particularly attractive when approximation decisions are sequential, local, and context-dependent. Custom heuristic algorithms remain common in application-specific contexts, especially when domain knowledge can effectively prune large parts of the design space [43, 33]. Hybrid approaches that combine domain heuristics with learning-based search often achieve strong efficiency-accuracy trade-offs.

1.5 A Brief Look at State-of-the-Art Design Space Exploration Approaches for Approximate Systems

While an in-depth discussion of existing DSE methods for AxC is reserved for the background chapter (cf. Chapter 2), it is worth outlining several key themes that emerge from a high-level look at current literature [2]. A recurring theme is

the need for fast accuracy prediction to keep exploration tractable without sacrificing conservatism. Many studies adopt *custom exploration algorithms* combining domain-specific heuristics with partial evaluations, especially when the application domains include image or signal processing. Such tailoring leverages application structure (e.g., spatial locality in images) to accelerate screening of candidate designs. A comparable fraction of works leverages EAs (such as GA or NSGA-II), particularly in the context of Field-Programmable Gate Arrays (FPGAs) or Application-Specific Integrated Circuits (ASICs). Evolutionary methods are well-suited to discrete operator choices and architectural variants. Additionally, RL strategies are increasingly popular, albeit typically aligned with specific application classes like NNs or dynamically reconfigurable systems. Here, the environment's reward function encodes accuracy and resource objectives, guiding policies toward feasible approximations.

Another recurring pattern in approximate designs involves employing *multiple* approximation strategies. Cross-layer co-optimization can unlock additive or even synergistic savings, but complicates correctness reasoning. For instance, hardware-level optimizations such as partial product perforation in multipliers may be combined with software-level loop perforation, thereby expanding the DSE problem into multiple dimensions. Therefore, mechanisms for constraint enforcement and run-time monitoring become increasingly important. Meanwhile, the prevalent benchmarks vary in their tolerance for approximation errors. Image-focused tasks usually rely on metrics like PSNR, SSIM, or Mean Pixel Difference (MPD) [1, 6], whereas scientific or ML applications impose tighter bounds on accuracy, frequently measured via MSE, MAE, or classification accuracy. Metric choice directly shapes the Pareto frontier and must reflect end-user utility and domain semantics. These examples underscore how the pursuit of optimal approximate designs can differ dramatically based on the metric used to quantify acceptable error and use case domain [2]. Collectively, these observations motivate the two complementary lines of inquiry pursued in this thesis.

Nevertheless, two challenges persist in efforts to propose a suitable DSE approach for approximate systems: One challenging decision is about the cost of evaluation, as accuracy assessment may increase the application execution time, especially when simulation is preferred to analysis. On the other hand, automating locality becomes a challenge when the DSE is performed to pinpoint where the approximation should be applied at fine granularity [17]. The two methods advanced in this thesis directly target these axes: Interval Arithmetic (IA)-based modeling to obtain fast, conservative bounds in Spiking Neural Networks (SNNs), and RL-based exploration to automate per-operator/per-variable approximation under explicit accuracy constraints.

1.6 Objectives and Contributions

The work in this thesis builds on the broad context established by prior DSE research for AxC, aiming to simplify the identification and adoption of approximation strategies under multi-objective constraints (e.g., performance, area, and accuracy). The unifying goal is to reduce evaluation cost while preserving conservative guarantees on application-level correctness. The primary objectives include improving the modeling of approximation-induced errors in complex workloads and automating the selection of approximate operators or parameter settings across the software-hardware boundary. Accordingly, the thesis develops methods that are both analysis-driven (for bounds under the considered model) and learning-driven (for scalable search in the studied spaces). The research follows two main thrusts: the first explores the use of IA to predict how precision scaling affects application-level outputs in SNNs, while the second employs ML in particular, RL for a multi-objective DSE that adaptively balances accuracy and resource usage. These thrusts are complementary: interval-based prediction prunes infeasible designs early, while reinforcement learning automates localized choices among many approximation options. Through these contributions, the thesis aligns with the broader vision of creating more energy-efficient, high-performance architectures suited to modern data-driven workloads. The resulting frameworks are designed to be modular, facilitating adaptation to application domains with explicit accuracy constraints.

Research Questions

Guided by the gaps and tool requirements discussed in the literature, this thesis addresses the following questions.

1. Is it possible to design a fast, conservative error estimation method that enables exploration of precision reduction in complex designs such as SNNs without exhaustive simulation?

2. Is it possible to automate localized approximation decisions to simultaneously improve power and execution time while keeping the accuracy degradation within application-level accuracy constraints?

These questions target the dual bottlenecks of accuracy evaluation cost and fine-grained decision automation in large design spaces.

Thesis Contributions

To answer these questions, the research contributions are:

- An IA based approach to estimate the effect of applying precision scaling on the output accuracy of an SNN, while further improving the approach to use a mechanism to fine-tune the precision of all network weights.

- An RL based DSE approach that aims to find which arithmetic operators can be approximated by selecting from a range of possible approximate operators with the goal of optimizing *power* and *execution time* under accuracy constraints.

Together, these contributions show that, in the studied settings, conservative, analysis-guided pruning and adaptive, policy-driven exploration can be integrated to support the identification of non-dominated (Pareto) trade-offs under explicit accuracy budgets.

1.7 Structure of the Thesis

This introduction has situated AxC within the wider landscape of performance and power challenges in contemporary systems, discussing how the diminishing returns of CMOS scaling, coupled with growing data demands, have prompted considerable interest in approximation-based methods. It has also articulated why accuracy should be treated as a tunable design parameter that is constrained but not fixed by application semantics. Chapter 2 provides a deeper examination of AxC classifications and reviews different AxC strategies and error metrics, drawing on the extensive body of literature and the survey of DSE methodologies. This chapter consolidates terminology, metrics, and problem formulations used throughout the thesis to ensure consistency. Chapter 3 then focuses on IA approaches for predicting approximation effects, particularly within SNNs, where local truncation or bit-width scaling can yield substantial power savings without catastrophic accuracy loss. The chapter emphasizes bounds tightness, scalability, and integration with exploration workflows. Chapter 4 presents a RL-based DSE framework that provides the means for multi-objective exploration of a design space comprised of approximate designs, automating the selection of approximate arithmetic operators under user-defined accuracy constraints. It details state representations, reward shaping, constraint handling, and convergence considerations relevant to practical deployment. Finally, Chapter 5 summarizes the main outcomes, highlights the trade-offs uncovered, and suggests opportunities for future work, including broader cross-level approximations, integration with emerging hardware paradigms, and refinements to DSE search strategies for large-scale approximate designs. The thesis concludes with a discussion of generalization, limitations, and potential extensions toward runtime-adaptive approximation and cross-layer co-optimization.

Chapter 2

Background and Related Work

2.1 Setting the Scene

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 [42]. 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 subsections provide an overview of how different AxC techniques can be classified, followed by a description of the DSE paradigm.

2.1.1 Classification of approximate Computing 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 2.1. For instance, memory read approximation can be achieved through pure software approaches or within memory control units. While Figure 2.1 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 [20, 17].

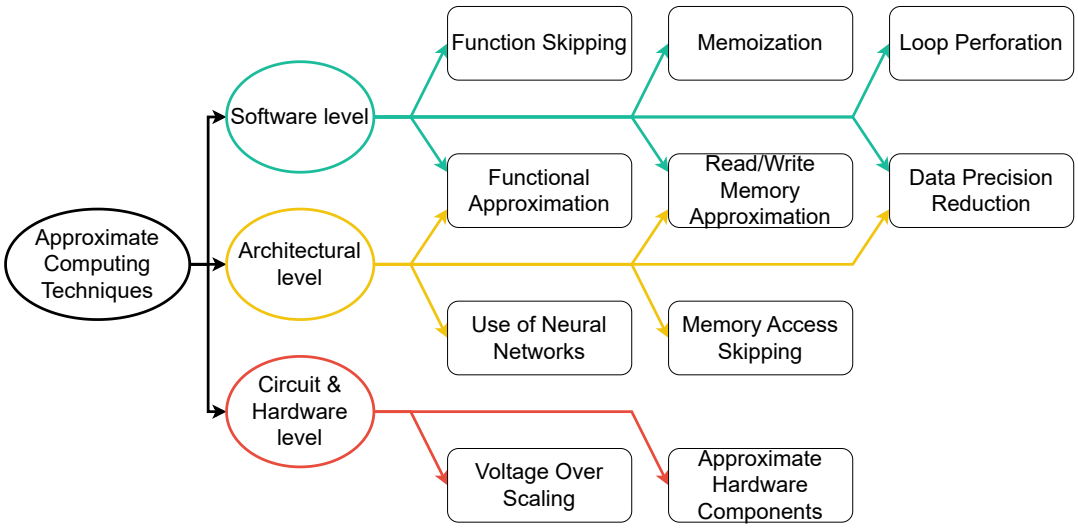

Figure 2.1: A classification of the AxC techniques [2].

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 [21]. Similarly, function-skipping involves bypassing specific code blocks during runtime based on predefined conditions [47, 22, 48].

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 [25].

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 [49, 50, 51].

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 [52, 15].

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 [23, 24]. Here, one commonly used form of precision scaling is often referred to as bit truncation (or fixed-point quantization). In this approach,

a floating-point (or higher-precision) value is converted into a more compact fixed-point format, and several of the least significant bits are deliberately truncated. Truncating k fractional bits effectively limits the numeric resolution, yielding an approximate value $v - \epsilon$, where ϵ is bounded by the magnitude of those truncated bits. This can substantially reduce energy consumption and execution time, especially on resource-constrained devices, as long as the resulting error remains tolerable for the application at hand. Chapter 3 demonstrates how bit truncation can be used in a SNN framework, using IA to provide a fast prediction and manage the errors introduced by precision reduction. In this work, the applied precision reduction is fixed-point truncation of fractional bits. Truncation while dropping the k least-significant fractional bits, yields a one-sided, bounded error that is convenient for conservative analysis. In comparison, round-to-nearest quantization produces symmetric errors with different bounds. Truncation is adopted in this work specifically because its bounds integrate cleanly with the IA error-propagation model used to predict SNN accuracy under reduced precision (see Chapter 3).

Architectural-Level AxC Techniques

Architectural-level AxC techniques introduce bounded inaccuracy by reconfiguring how computations and data are scheduled and managed at the microarchitecture/system level (e.g., memory hierarchies, on-chip interconnects, execution scheduling, and accelerator interfaces) while typically preserving the Instruction Set Architecture (ISA) and without necessarily redesigning primitive computation circuits. Instead of altering computation units at the circuit level or modifying algorithms at the code source level, these methods redirect tasks onto approximate execution paths, bypass or coalesce memory accesses, modulate precision in accelerator kernels, or relax cache management policies (e.g., replacement or prefetch aggressiveness), all under compile-time/runtime control that maintains target accuracy constraints.

For instance, 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 [53].

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 [26].

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 [28]. 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 [54]. 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 [28]. In contrast, VOS aggressively reduces the supply voltage beyond nominal levels, intentionally allowing timing violations that can introduce errors in computations [55, 56]. 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 [20, 17].

Inexact hardware can provide approximation at the hardware level, with numerous examples from the literature of AxC 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 computational 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 AxC design [11].

While AxC units can be custom-designed, there also exist curated operator libraries that provide ready-to-use implementations of inexact adders, multipliers, and other circuits. These libraries, such as `EvoApproxLib` [4, 5], catalog multiple variants of approximate operators that differ in terms of latency, power consumption, and accuracy levels (often measured by Mean Relative Error Distance (MRED), MED, or MSE). The ability to choose from a range of approximate operators allows designers to explore the trade-off between performance, power, and output error in various application domains, including ASICs, FPGAs, or even software emulation on Central Processing Units (CPUs). As shown in Chapter 4, leveraging such libraries within a DSE process (e.g., guided by RL) enables automated selection of the suitable approximate adders and multipliers under specific accuracy constraints. In practice, operator substitution in this thesis is realized at the software level by instrumenting CPU code so that additions and multiplications

on selected variables call approximate kernels from a characterized library (EvoApproxLib) instead of precise operators. Each library variant is identified by its bit width and exhibits a measured accuracy profile, along with implementation proxies for power and delay; these figures guide the DSE. Beyond choosing the adder/multiplier variant, we also employ per-variable *approximation masks* (i.e., only variables flagged by the exploration are computed via approximate operators), which enables heterogeneous, fine-grained application of hardware-level approximation within a single program (see Chapter 4).

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 earlier need to be addressed, and DSE is a widely adopted strategy for addressing such challenges.

2.1.2 Precision Metrics

Various precision metrics have been proposed to describe and evaluate the effectiveness of AxC techniques and methodologies, as detailed in Table 2.1.

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 i^{th} 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 (2.1) and (2.2), respectively [2].

$$\mathrm{ED} = |O_{AxC}^{(i)} - O_{Acc}^{(i)}| \tag{2.1}$$

$$\mathrm{EP} = \frac{1}{N}\sum_{i=1}^{N} \mathbf{1}\Big\{ O_{AxC}^{(i)} \neq O_{Acc}^{(i)} \Big\}. \tag{2.2}$$

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

MED is the average of all the error distances, which is calculated as shown in Equation (2.3).

$$\mathrm{MED} = \frac{1}{N}\sum_{i=1}^{N} |O_{AxC}^{(i)} - O_{Acc}^{(i)}| \tag{2.3}$$

MRED is the average of error distances relative to the correct answers, formulated in Equation (2.4).

$$\mathrm{MRED} = \frac{1}{N}\sum_{i=1}^{N} \frac{|O_{AxC}^{(i)} - O_{Acc}^{(i)}|}{|O_{Acc}^{(i)}|} \tag{2.4}$$

Table 2.1: Precision metrics in AxC [1, 2].

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	

Note. For samples with $O_{Acc}^{(i)} = 0$, the output is excluded from the average, or a small ε in the denominator can be used; alternatively, $NMRED$ avoids this issue by normalizing with respect to full-scale.

$$\text{NMED} = \frac{1}{N}\sum_{i=1}^{N}\frac{\left|O_{AxC}^{(i)} - O_{Acc}^{(i)}\right|}{R}, \qquad \text{NMRED} = \frac{1}{N}\sum_{i=1}^{N}\frac{\left|O_{AxC}^{(i)} - O_{Acc}^{(i)}\right|}{\max\left(\left|O_{Acc}^{(i)}\right|, \varepsilon\right)} \tag{2.5}$$

where R is the full-scale value (e.g., 255 for 8-bit), and ε is a small constant.

Absolute Worst-Case Error (AWCE) is the largest error distance that can occur, as shown in Equation (2.6).

$$\text{AWCE} = \max_{\forall i} |O_{AxC}^{(i)} - O_{Acc}^{(i)}| \tag{2.6}$$

MSE is the average of the squares of the errors between the approximate values and the accurate values, formulated in Equation (2.7).

$$\text{MSE} = \frac{1}{N}\sum_{i=1}^{N}(O_{AxC}^{(i)} - O_{Acc}^{(i)})^2 \tag{2.7}$$

Hamming Distance (HD) measures the difference between two strings of equal length. It is defined as the number of positions at which the corresponding symbols differ. For instance, the binary strings "1011101" and "1001001" differ in two positions [2].

Error Rate (ER) quantifies the number of erroneous results over the total number of results, as in Equation (2.8), where N_{err} is the number of erroneous results and N_{tot} is the total number of results.

$$\text{ER} = \frac{N_{err}}{N_{tot}} \tag{2.8}$$

Bit Error Ratio (BER) is calculated by dividing the number of bit errors by the total number of bits transferred during a given time interval, as in Equation (2.9). BER is crucial for evaluating data transmission accuracy in communication systems.

$$\text{BER} = \frac{N_{bit\_err}}{N_{bit\_tot}} \tag{2.9}$$

PSNR evaluates the quality of reconstructed images or videos, measuring the ratio between the maximum possible signal power and the power of corrupting noise in decibels (dB). Higher PSNR values correspond to better quality. Equation (2.10) formulates PSNR, where R is the maximum possible pixel value (e.g., 255 for 8-bit images), and MSE corresponds to the MSE between the original and distorted images.

$$\text{PSNR} = 10 \log_{10}\left(\frac{R^2}{\text{MSE}}\right) \tag{2.10}$$

SSIM measures perceptual similarity by comparing structural information, luminance, and contrast between an image and a reference image. Unlike PSNR, it focuses on how humans perceive image quality. SSIM ranges from -1 to 1, with 1 indicating perfect similarity, and is defined in Equation (2.11), where x and y represent the two images being compared, μ_x and μ_y are their mean pixel values, σ_x^2 and σ_y^2 their variances, and σ_{xy} is their covariance. In addition, C_1 and C_2 are two small constants introduced to stabilize the division in case the denominators are close to zero. They are typically defined as $C_1 = (K_1 \cdot R)^2$ and $C_2 = (K_2 \cdot R)^2$, where R is the dynamic range of the pixel values (e.g., 255 for 8-bit grayscale images), and K_1 and K_2 are small positive scalar values (commonly set to $K_1 = 0.01$ and $K_2 = 0.03$).

$$\text{SSIM}(x, y) = \frac{(2\mu_x\mu_y + C_1)(2\sigma_{xy} + C_2)}{(\mu_x^2 + \mu_y^2 + C_1)(\sigma_x^2 + \sigma_y^2 + C_2)} \tag{2.11}$$

MPD is a simpler measure of how similar two images are, defined by the average absolute difference in their pixel values, as shown in Equation (2.12).

$$\text{MPD} = \frac{1}{N}\sum_{i=1}^{N}\left|P_i - Q_i\right| \tag{2.12}$$

Here, N denotes the number of pixels (or samples) in the images.

In binary classification tasks, a confusion matrix composed of True Positives (TPs), True Negatives (TNs), False Positives (FPs), and False Negatives (FNs) captures model performance [57]. From this, classification accuracy (Equation (2.13)), recall (Equation (2.14)), false positive rate (Equation (2.15)), and precision (Equation (2.16)) are commonly derived [2].

$$\text{Classification Accuracy} = \frac{\text{TP} + \text{TN}}{\text{TP} + \text{TN} + \text{FP} + \text{FN}} \tag{2.13}$$

$$\text{Recall} = \frac{\text{TP}}{\text{TP} + \text{FN}} \tag{2.14}$$

$$\text{FPR} = \frac{\text{FP}}{\text{FP} + \text{TN}} \tag{2.15}$$

$$\text{Precision} = \frac{\text{TP}}{\text{TP} + \text{FP}} \tag{2.16}$$

2.1.3 Design Space Exploration

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 multiple design alternatives to balance performance, power consumption, and accuracy trade-offs. The goal is to identify the best approximate versions of an application or circuit that offer significant gains in efficiency (power, performance, area) while preserving acceptable output quality for the target application [58, 6].

Designing modern embedded computer systems presents numerous challenges, as these systems typically must meet strict and often conflicting design criteria. Devices intended for mass production and battery-powered or passive-cooling environments demand cost-effectiveness and energy efficiency, while safety-critical applications in avionics or space technology require high dependability. Additionally, many systems must support multiple applications and standards, operate in real-time, and accommodate future updates and extensions, all within tight power and performance constraints [59].

Consequently, heterogeneous Multi-Processor System-on-Chips (MPSoCs) have become increasingly common, comprising programmable cores and dedicated hardware blocks. System-level design methodologies have emerged to tackle the complexity of such designs by elevating the level of abstraction and relying on high-level system modeling and simulation [40, 60, 61].

Early DSE can help developers explore decisions such as the number and types of processors, the interconnection network, and the mapping of application tasks to processor cores [62, 39]. Yet, the size of the design space increases with the number of tasks and processing elements, often rendering the problem NP-hard [63]. This challenge intensifies when AxC is introduced, since multiple AxC techniques can be applied simultaneously, necessitating an evaluation of each combination's effect on accuracy, performance, and power consumption.

DSE approaches can be broadly categorized as single-objective or multi-objective, where multi-objective DSE attempts to optimize multiple, potentially conflicting goals (e.g., performance, power, area, reliability) simultaneously, typically resulting in a Pareto-optimal set of trade-offs [39, 41]. Practical DSE processes often combine heuristic or modeling-based methods to prune vast design spaces. Exhaustive evaluation of all approximate configurations [64, 65] can be infeasible, leading some researchers to develop faster approximation models [37, 66] to assess error impacts. Despite the inherent complexity, DSE remains a pivotal strategy for systematically harnessing the benefits of AxC.

2.2 Related Works

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 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 rely on data-driven methods to predict and explore optimal design configurations, while EAs use bio-inspired strategies such as GAs to navigate the design space [2].

Table A.1 provides information about the research works that took an ML approach to perform the DSE, while Table A.2 includes information about the research works that leveraged EAs to perform the DSE. All the remaining research works that perform DSE using other heuristic algorithms or combining different optimization algorithms are listed in Table A.3. While Tables A.1–A.4 offer an overview that allows comparison among different studies based on the employed search algorithm, target hardware, and use case domain, Tables A.5–A.8 provide an overview of the same sets of studies, but arranged by the AxC techniques applied in each study.

2.2.1 Design Space Exploration Using Machine Learning Algorithms

As reported in Table A.1, the most popular ML algorithm is RL [45, 67, 46, 6], although authors in [68, 69] use Multi-objective Bayesian Optimization (MBO) and modified Monte Carlo Tree Search (MCTS), respectively, and authors in [38, 32] mention general ML-based search algorithms. While target hardware in these studies varies from FPGAs and ASICs to general-purpose CPUs, the use-case domain generally involves image and signal processing benchmarks, ranging from traditional image processing to NNs for image classification. Table A.5 indicates that replacing exact adders and multipliers with approximate counterparts is the most common hardware-level approximation investigated [68, 38, 69, 32, 6], although software-level AxC is also explored. In [45, 68], algorithm parameters are reduced (e.g., loop iterations, input sizes) to save computation time or memory at the expense of accuracy. Similarly, [67] applies loop perforation and changes data structures. In [46], an ML search refines quantization levels in DNNs to find accuracy/efficiency trade-offs [2].

Prior solutions such as RL-driven DSE for layer/parameter selection or quantization tuning for DNNs [45, 46], data-driven search for approximate operators [38, 32], and model-based optimization or tree search [68, 69] demonstrate that ML-based search algorithms can guide DSE for approximation use cases efficiently. However, most of the proposed DSE methods either (i) optimize a single dominant objective

(e.g., accuracy or energy) or (ii) restrict the abstraction level (e.g., only quantization), making scalability to multi-objective, cross-abstraction exploration difficult. In this thesis, the goal was to propose a multi-objective, *scalable* RL-based DSE that jointly optimizes power and execution time under explicit accuracy constraints while operating over approximate operator substitutions drawn from characterized libraries (see Chapter 4).

2.2.2 Design Space Exploration Using Evolutionary Algorithms

Table A.2 lists research efforts employing EAs for DSE. Most studies use GA or NSGA-II. An Evolutionary Strategy (ES) algorithm appears in [70]. The chosen hardware targets generally include FPGAs, ASICs, or Graphics Processing Units (GPUs), often for image- or video-processing tasks. Table A.6 shows that hardware-level approximation (e.g., approximate adders/multipliers) predominates. Some studies alter software-level instructions, as in [71], where different domains (scientific computing, 3D rendering, and image processing) are explored by modifying static program instructions. In [72], authors emulate approximate multipliers in Convolutional Neural Networks (CNNs) layers to achieve quantization-aware training and dynamic DSE [2].

EA-based approaches [71, 72, 70] effectively handle multi-objective trade-offs on ASIC/FPGA/GPU targets, but typically incur high evaluation costs and require extensive parameter tuning, limiting scalability on very large design spaces and mixed hardware/software settings. While EA-based methods are effective, their population-pruning approach can make scalability challenging when the design space explodes: keeping populations tractable typically requires aggressive (and stochastic) selection and pruning, which can prematurely discard high-quality regions or introduce high evaluation variance as the space grows. In contrast, ML-based methods, especially RL-based methods, learn a reusable policy that amortizes exploration across states and can reduce the instances of repeatedly evaluating dominated regions, thereby reducing search overhead while preserving multi-objective trade-offs. Hence, in this thesis, an RL-based search algorithm was selected instead of an EA-based approach, to reduce search overhead while preserving multi-objective optimality characteristics and scalability option for large design spaces.(see Chapter 4).

2.2.3 Design Space Exploration Using Custom Algorithms

Table A.3 includes works that neither rely on ML nor EAs for DSE. For instance, [73] uses a Tabu Search (TS) algorithm and suggests possible integration of GAs later. In [74], a Gradient Descent (GD)-based method is employed. All other entries

reference custom multi-step or heuristic algorithms, sometimes with pruning phases before or after the main search to handle large design spaces.

Hardware targets often include ASICs or FPGAs, although [75] implies general-purpose hardware and [76] focuses on GPUs (with a CPU comparison). Many of these approaches employ approximate adders/multipliers [43, 77, 73, 78, 75]. Others introduce specialized hardware-level AxC techniques such as logic isolation [79], clock gating [80, 74], or VOS [78]. Some focus on approximate DNNs at different abstraction layers, e.g., dynamic quantization [33] or approximate NPUs [27].

Custom heuristics and pruning pipelines [73, 74, 43, 78, 75, 80] are highly effective in problem-specific contexts, but their portability across abstraction levels and application classes is limited. In this thesis, the goal was to adopt a general DSE approach that enables search over different approximation methods, thereby enabling portability and automated scaling (Chapter 4). Hence, instead of using a custom algorithm for the DSE and bounding the method to specific hardware/-software characteristics, an ML-based search algorithm was adopted.

2.2.4 Design Space Exploration of Approximate Functions Design

Some studies concentrate on approximate logic synthesis rather than entire systems, focusing on approximate versions of Boolean networks or sub-functions. These are shown in Table A.4, and their corresponding AxC techniques in Table A.8. For example, [81, 82] apply logic falsification to generate approximate circuits, whereas [83] proposes Boolean network simplifications allowed by External Don't-Cares (EXDCs), [84] uses Boolean Matrix Factorization (BMF) for approximate truth tables, and [85] employs customized Boolean approximations. Typical benchmarks involve approximate adders and multipliers, though some works also explore ALUs, decoders, shifters, and other combinational circuits. Interestingly, [82] targets safety-critical scenarios through Quadruple Approximate Modular Redundancy (QAMR), as opposed to traditional TMR with exact circuits [2].

DSE approaches proposed for approximate logic synthesis and operator design [83, 81, 82, 84, 85] provide rich libraries but often stop short of a system-level framework that selects and integrates these components under multi-objective constraints. Hence, these proposed DSE approaches are not suitable for system-level analysis of approximation opportunities.

2.2.5 Evaluated Parameters in Design Space Exploration

While Tables A.1–A.4 compare different DSE studies by search algorithm, hardware target, and application domain, Tables A.9–A.12 compare those same studies based on the evaluated parameters involved in approximation trade-offs [2].

Since AxC aims to trade off accuracy for performance and energy efficiency, accuracy naturally remains the first parameter to evaluate in DSE. Different works may track power consumption, energy usage, area utilization, or execution time to represent their main design objectives. Studies targeting FPGAs often focus on Look-Up Table (LUT) or resource usage; others may concentrate on ASIC gate count, GPU memory bandwidth, or general-purpose CPU runtime. Large Artificial Neural Networks (ANNs) introduce particularly long execution times, amplifying the importance of DSE frameworks that prune huge search spaces efficiently. Some works also track memory usage as a lesser but still relevant factor, especially when approximate loop parameters or truncated data structures reduce memory demands (e.g., [45, 76]).

Besides high-level metrics like speedup or circuit area, most studies employ specialized error metrics tailored to their domain (e.g., PSNR for image/video applications, classification accuracy for DNNs). Table 2.1 (in the previous section) summarized many such metrics. In effect, DSE performed to find the suitable AxC techniques to be applied to a design, often involves solving a Multi-objective Optimization Problem (MOP), aiming to identify a Pareto front that balances accuracy, energy/power, area, and execution time [2].

2.3 In Closing

A synthesis of the literature reviewed in Section 2.2 reveals several trends and remaining gaps. In studies that employ ML-based algorithms for DSE, RL, and other data-driven approaches such as MCTS or MBO are among the most widely adopted techniques. These approaches have demonstrated utility for exploring diverse design spaces. Yet most of them focus primarily on a single design parameter, such as accuracy or energy efficiency, rather than performing multi-objective optimization across multiple system metrics. Moreover, these studies generally emphasize standard NNs or signal-processing benchmarks, leaving many other computing paradigms, such as SNNs, underexplored.

In contrast, works adopting EAs, notably GAs, NSGA-II, and ESs, have demonstrated effectiveness in optimizing multi-objective problems for image and signal processing applications, particularly on hardware targets such as ASICs, FPGAs, and GPUs. However, these approaches are computationally expensive and often require significant tuning of population size, crossover, and mutation parameters. As a result, their scalability to very large design spaces or to problems with tightly coupled hardware-software co-design constraints remains limited.

Meanwhile, many different custom DSE algorithms and heuristic methods exhibit greater adaptability to specific design contexts, often incorporating problem-specific pruning, clock-gating, or logic isolation strategies. These techniques are particularly effective for small to medium-sized design spaces, where domain-specific

knowledge can be exploited to reduce exploration time. Yet, such methods are difficult to generalize and lack the flexibility required to scale across multiple abstraction levels or diverse application domains.

Finally, studies focusing on the approximation of functions or components, rather than entire systems, concentrate mainly on Boolean network simplification and AxC component design. These efforts have led to the creation of standardized libraries such as EvoApproxLib, enabling faster reuse of approximate operators. However, they often omit higher-level exploration frameworks capable of automatically selecting and integrating these components into broader architectures. Consequently, there remains a clear gap between component-level approximation and system-level DSE that accounts for interdependencies among multiple design parameters and application-specific quality metrics.

Motivated by these insights, the research conducted in this work followed two complementary trajectories. First, a framework was proposed to provide a quick yet acceptable estimate of the effect of approximation-induced errors on the output accuracy of an SNN. For this purpose, not only the accuracy degradation at the computational output was considered, but also, as many works in the literature suggest, the accuracy metric specific to the target ML application was considered. The framework was further extended to use a quick estimate of output accuracy degradation to find the most approximated version of the NN that still has acceptable output accuracy. Second, an RL-based DSE framework was proposed to find the approximation configuration that targets simultaneous optimization of power and execution time while keeping the output accuracy degradation within the acceptable range. As the literature review shows, the algorithms selected for performing the DSE in the state of the art can be categorized into three large groups: ML, EA, or custom. Custom algorithms are usually preferred when the size of the design space is smaller than the size of the problem that can be solved with ML or EA algorithms, or the needs of the specific hardware or software design make some algorithms a more preferable choice than the ML or EA options. Since the aim of the research was primarily to propose a framework to perform DSE in large design spaces where EA or many other algorithms may incur high evaluation costs for performing the exploration, the ML category was selected. As many works in the literature suggest, RL is a common algorithm choice for this purpose, as it has shown promising results for DSE tasks with large design spaces. Also, as the literature review suggests, most proposed DSE approaches do not cover the simultaneous balancing of multiple design parameters against accuracy. Hence, proposing a DSE framework that can account for all design parameters, such as power and execution time, while also incorporating other parameters into simultaneous multi-objective exploration, became one of the research questions.

Through these research paths, this thesis contributes toward addressing some of the open challenges identified in the literature: the need for fast, yet acceptable, estimation of approximation-induced error effects and the need for scalable,

automated exploration frameworks that can perform multi-objective optimization across vast design spaces. Together, these research efforts aimed at advancing the state of the art in the DSE approaches for AxC systems and providing a foundation for future extensions toward broader classes of computing architectures and application domains.

Chapter 3

Design Space Exploration Using Interval Arithmetic

3.1 Prediction of the Impact of Approximate Computing on Spiking Neural Networks via Interval Arithmetic

This section describes how SNNs can be modeled when applying AxC techniques by leveraging IA to estimate the impact of approximation on final network outputs. This section covers the fundamental SNN structure, the data quantization and precision reduction steps applied to the trained network, the IA-based model for propagating approximation errors, and the corresponding experimental findings.

3.1.1 Spike Neuron Model and Network Description

SNNs are a class of biologically inspired neural models that emulate the temporal dynamics of biological neurons by exchanging discrete electrical impulses, refered to as "spikes", instead of continuous activation values. This event-driven computation mechanism allows SNNs to process information asynchronously and efficiently, consuming energy only when spikes occur. Such behavior makes SNNs particularly suitable for hardware implementations targeting energy-efficient and low-latency applications, especially when deployed on edge or embedded systems.

The network architecture generally consists of three types of layers: an input layer that encodes sensory information into spike trains, one or more hidden layers composed of spiking neurons, and an output layer that decodes spike activity into interpretable results. In the considered model [86], the hidden layer contains both excitatory and inhibitory neurons, inspired by cortical structures where excitation and inhibition coexist to regulate neural activity and prevent uncontrolled firing. Figure 3.1 illustrates the general organization of an SNN, highlighting the behavior

of each spiking neuron in the layer: each neuron has two computational steps; an *excitatory* step and an *inhibitory* step. Spikes are generated by an input layer that transforms data (e.g., image pixels) into a sequence of binary spikes (bits set to 1), which enter the neuron through the excitatory step (green arrows in Figure 3.1). The excitatory step accumulates a *membrane potential* (V_0 in Figure 3.2), which increases by a sum of weights whenever it detects active spikes; when V_0 exceeds a designated threshold, the neuron fires a spike (green arrows exiting the excitatory block in Figure 3.1). This spike also feeds into the inhibitory step (black arrows in Figure 3.1), where the membrane potential of the neuron is decreased by the spikes of other neurons in the layer. Finally, the last layer sends spikes to an output layer that translates them into application-relevant results.

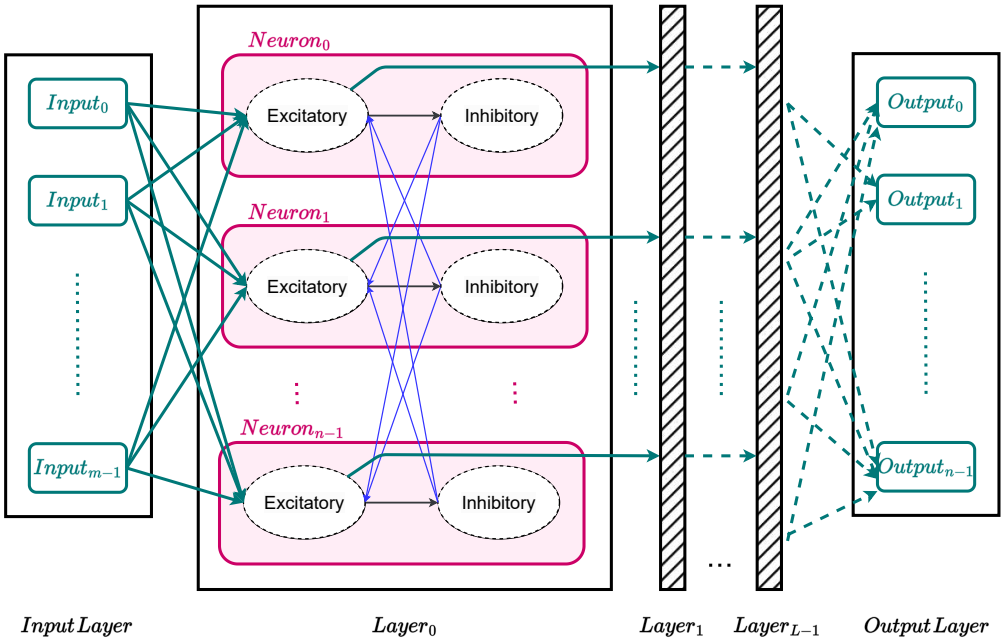

Figure 3.1: An excitatory and an inhibitory layer of an SNN [3].

In the excitatory step, neurons integrate the incoming spikes into a membrane potential variable, V_0, which accumulates charge over time. Each input spike, weighted by its synaptic strength, contributes to an increase in V_0. When this potential exceeds a defined threshold, V_{thresh}, the neuron emits an output spike and immediately resets its potential to V_{reset}. This mechanism mimics the integrate-and-fire process observed in biological neurons, where the firing event represents the transfer of information to connected neurons. The inhibitory neurons, conversely, provide balancing feedback by reducing the membrane potential of other neurons when they fire, ensuring that only relevant neurons remain active in response to specific input patterns.

It is important to note that in the reference implementation of the target SNN [86], lateral inhibition within the hidden layer is realized implicitly through signed synaptic contributions and the reset/threshold dynamics, rather than via a dedicated inhibitory processing block. In practice, spikes from neurons that play an inhibitory role contribute negative (or subtractive) terms to the postsynaptic membrane update, thereby suppressing over-activation and stabilizing the firing regime across the layer. This design choice consolidates excitation and inhibition within the same accumulation datapath and control flow, avoiding additional hardware stages while preserving the functional effect of inhibition at inference time. Consequently, the present chapter focuses on excitatory accumulation and the threshold-reset mechanism, while accounting for inhibitory interactions as subtractive contributions to the weighted summation and temporal evolution of the membrane potential, consistent with the behavior described in the original SNN model.

It is also worth mentioning that, in the adopted SNN implementation [86], the input layer converts each image into time-discretized binary spike trains such that, at every time step, each input connection delivers a bit $inp_i \in \{0,1\}$; the details of the Poisson-based encoding procedure and its parameterization are provided in subsection 3.1.4.

Figure 3.2 provides more detail for a single layer. The increment of V_0 is governed by each connection's weight, computed during training ($w_{0,i}$ in the example). When active spikes arrive, V_0 increases by summing the corresponding weights. Formally, each spike-bearing connection i contributes $w_{j,i}$ if $inp_i = 1$, as in Equation 3.1:

$$sum_j = \sum_{i=0}^{n-1} inp_i \cdot w_{j,i}, \tag{3.1}$$

where n is the number of input connections, and inp_i is 1 for an active spike.

Equation 3.1 describes the weighted summation step that forms the basis of spike integration. Each active input connection contributes a weight $w_{j,i}$ to the membrane potential, while inactive connections have no effect. Because spikes are binary, the multiplication simplifies to either adding the weight or skipping it. This simplification allows efficient hardware implementation and enables the modeling of spike-induced voltage changes at the neuron level.

Once V_0 exceeds V_{thresh}, the neuron fires a spike and resets V_0 to V_{reset} (the "yes" branch in Figure 3.2). If no active spike is detected, V_0 decays at each time step, approximated by a right shift on hardware [86]. Some SNNs (e.g., [86]) also maintain a counter of spikes for each neuron, which can aid in classification [3].

The dynamics of the membrane potential include a decay process that continuously reduces V_0 in the absence of new input spikes, preventing neurons from retaining outdated information. Mathematically, this exponential decay is often represented as $V_0(t+1) = V_0(t) \cdot e^{-\lambda}$, where λ is a decay constant. However, for

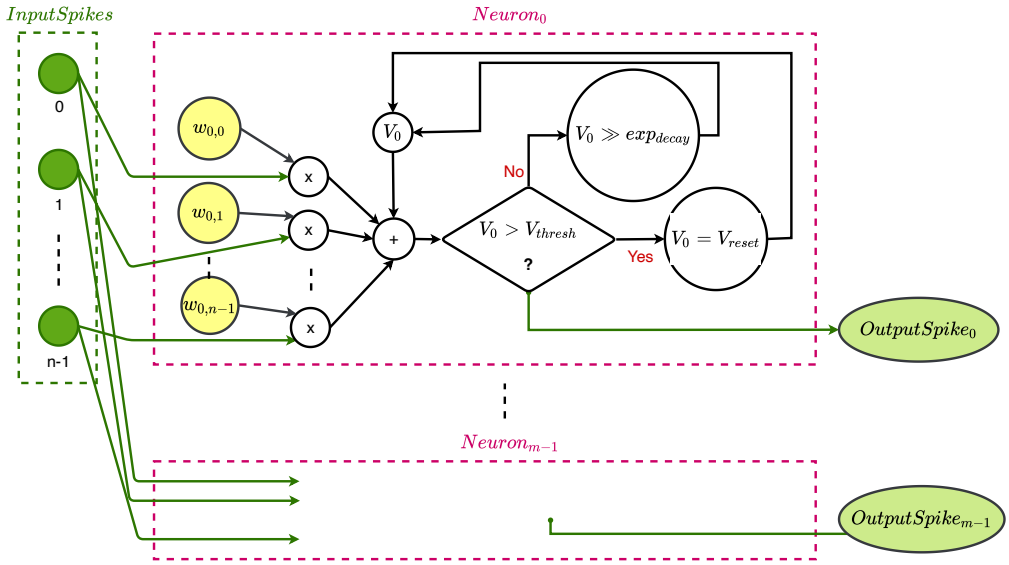

Figure 3.2: Computation flow of an SNN layer [3].

efficient hardware realization, the exponential term can be approximated as a right-shift operation, which effectively divides the potential by a power of two [86]. This hardware-friendly simplification reduces implementation complexity and resource utilization on FPGA or ASIC platforms while maintaining biologically plausible behavior.

The spiking and reset mechanisms form a feedback loop that enables temporal processing. When a neuron fires, it not only communicates with downstream neurons but also contributes to inhibitory feedback within the same layer, thus maintaining controlled network activity. In this implementation, a spike counter is assigned to each neuron to record the total number of emitted spikes. These counters serve as an indirect representation of neuronal activation and are later used by the classification layer to determine the final prediction outcome. This approach aligns with neuromorphic computing paradigms, where spike statistics rather than instantaneous activations are used for inference.

In summary, the chosen SNN architecture captures both biological fidelity and computational efficiency. Its modular structure, with well-defined excitatory, inhibitory, and decay dynamics, provides a foundation for analyzing how quantization and precision reduction propagate through the computation flow. Understanding this baseline behavior is essential before introducing approximation models such as IA, which will later allow accurate prediction of accuracy degradation without re-simulating the entire network.

3.1.2 Data Quantization and Precision Reduction

Deploying an SNN trained in floating-point arithmetic onto edge hardware (e.g., FPGA) often motivates fixed-point representations to reduce memory footprint and computation cost. Floating-point values require large storage and wide computation units that can quickly saturate available resources; in fixed-point, additions, multiplications, and comparisons map to simpler integer circuits. In [86], the weights, thresholds, and reset values are obtained using a floating-point training procedure [87]. Then, for deployment, each floating-point value is converted into a 32-bit fixed-point format with 16 fractional bits, as shown in Equation 3.2:

$$v = \sum_{i=16}^{31} b_i \cdot 2^{(i-16)} \; + \; \sum_{i=0}^{15} b_i \cdot 2^{-(16-i)}, \tag{3.2}$$

where b_i are bits in the fixed-point representation.

A precision reduction step removes some of these fractional bits starting at the least significant bits, introducing an approximation error ϵ. If k bits are removed, Equation 3.3 expresses ϵ in terms of the removed bits b_i:

$$\epsilon = \sum_{i=0}^{k} b_i \cdot 2^{-(16-i)}. \tag{3.3}$$

The range of possible errors, shown in Equation 3.4, depends on whether those k bits are 0 or 1:

$$\underbrace{0}_{\forall b_i=0} \; \leq \; \epsilon \; \leq \; \underbrace{\sum_{i=0}^{k} 2^{-(16-i)}}_{\forall b_i=1} = -\frac{1-2^k}{32768}. \tag{3.4}$$

Thus, the new approximate value can be written as $v - \epsilon$, lowering storage demands without excessive accuracy degradation [3].

However, quantization inevitably introduces a bounded numerical error, ϵ, which propagates through the computations and affects neuron dynamics. In SNNs, this perturbation can influence the membrane potential evolution, potentially altering the spike firing behavior. Therefore, quantization cannot be applied arbitrarily: designers must determine the level of precision reduction that preserves correct spike generation and overall classification accuracy.

It is important to note that at each time step, input spikes are binary events that indicate whether a neuron has fired or not, represented by a single bit with a value of either 0 or 1. Therefore, input signals cannot be approximated or quantized further. The approximations exclusively affect the network parameters, namely, the synaptic weights, thresholds, membrane potentials, and reset values, whose precision directly impacts the evolution of the membrane potential and, consequently, the spiking behavior.

By modeling ϵ explicitly, one can later formalize how these small deviations accumulate during successive weighted summations, threshold comparisons, and exponential decays. The explicit characterization of ϵ as an interval forms the foundation of the following section, where IA is used to describe the propagation of this approximation-induced error throughout the entire computational flow of the network.

3.1.3 Interval Arithmetic Error Propagation Model

IA provides a mathematical framework for representing numerical values along with their possible error bounds, enabling one to evaluate how errors propagate through arithmetic operations. In this work, the target SNN model is modified so that network parameters, such as weights, thresholds, or membrane potential, are represented not by a single deterministic value but by a range of possible values accounting for approximation errors introduced by precision reduction. This enables estimation of the impact of precision reduction on the spiking behavior, reducing the need for repeated full-network simulations. Using the error formulations and bounds, an IA-based model tracks how approximation-induced errors propagate through the SNN computations [88]. The value of each network parameter is represented by an interval $[v_{\min}, v_{\max}]$, while the approximation error ϵ also forms an interval $[\epsilon_{\min}, \epsilon_{\max}]$. Since precision reduction subtracts bits from v, each approximate number is expressed as $v - [\epsilon]$. Crucially,

1. ϵ is strictly monotonic, so $[\epsilon] = [\epsilon_{\min}, \epsilon_{\max}]$ never changes sign.

2. The relationship is subtractive: $|ian| \simeq [v] - [\epsilon]$.

This approach separates $[v]$ from its error $[\epsilon]$, unlike [89], providing flexibility in exploring approximation [3]. Here, $|ian|$ denotes any parameter represented by IA, while $[v]$ is the range of possible values for that parameter represented as an interval, and $[\epsilon]$ is the range of possible errors for that parameter represented as an interval.

Unlike traditional error analysis that models deviations as small additive perturbations, IA maintains lower and upper bounds for every variable, ensuring that all feasible values are included. This makes the approach particularly suitable for modeling operations that are repetitive and nonlinear, as in SNNs. Furthermore, by maintaining distinct intervals for the nominal value $[v]$ and for its corresponding error $[\epsilon]$, the model can apply interval-based arithmetic directly while clearly distinguishing between the true value and its associated approximation.

For a given neuron, the membrane potential update rule involves weighted summations, threshold comparisons, and exponential decays. Within the IA model, each of these operations is translated into an interval-based formulation, which keeps the potential value bounded and ensures that error accumulation remains

controlled. Since the input spikes are binary (0 or 1) and therefore exact, they are treated as constants in the computation. Only the parameters affected by precision reduction contribute to the propagated error.

This modeling choice helps identify, at design time, candidate truncation levels of precision reduction (k-bit truncations) that are likely to preserve the expected network behavior. It also provides a fast, analytical method to evaluate accuracy degradation caused by quantization without requiring hardware resynthesis or full retraining.

The following subsections define how basic arithmetic and comparison operations are adapted within this interval-based formulation, while maintaining the separation between nominal values and error terms, thereby enabling the modeling of neuron dynamics under approximation.

Addition and Subtraction

For two IA numbers $|ian_1|$ and $|ian_2|$, addition and subtraction (Equations 3.5, 3.6) maintain separate intervals for the values and errors:

$$|ian_1| + |ian_2| = \{\underbrace{[v_{1_{min}} + v_{2_{min}}, v_{1_{max}} + v_{2_{max}}]}_{[v]}, \underbrace{[\epsilon_{1_{min}} + \epsilon_{2_{min}}, \epsilon_{1_{max}} + \epsilon_{2_{max}}]}_{[\epsilon]}\} \quad (3.5)$$

$$|ian_1| - |ian_2| = \{\underbrace{[v_{1_{min}} - v_{2_{max}}, v_{1_{max}} - v_{2_{min}}]}_{[v]}, \underbrace{[\epsilon_{1_{min}} - \epsilon_{2_{max}}, \epsilon_{1_{max}} - \epsilon_{2_{min}}]}_{[\epsilon]}\} \quad (3.6)$$

Addition and subtraction are the fundamental interval operations used to model cumulative error propagation across arithmetic steps in an SNN. In both operations, the separate treatment of $[v]$ and $[\epsilon]$ preserves the additive structure of the computation while keeping the error bounds conservative.

During spike propagation, each neuron performs a weighted sum of inputs ($\sum_i inp_i \cdot w_i$). Since input spikes are binary and exact, the only error sources are the quantized parameters such as weights, thresholds, and reset values. The addition rule ensures that the errors of these parameters accumulate linearly across inputs, providing a bounded range for the membrane potential at each time step. Similarly, the subtraction rule models how inhibitory signals or reset mechanisms reduce the membrane potential, keeping both the potential and error ranges consistent with physical neuron behavior.

Maintaining monotonicity ensures numerical stability: even after multiple additions or subtractions, the resulting intervals remain bounded and do not diverge. This property is critical for iterative processes like the membrane potential updates that occur over successive time steps in an SNN.

Multiplication

Multiplication is a special case in which the standard IA product is not directly suitable for the adopted representation $|ian| \simeq [v] - [\epsilon]$; therefore, the product is expanded as in Equation 3.7 and then mapped back to separate intervals for the value $[v]$ and the error $[\epsilon]$.

$$|ian_1| \cdot |ian_2| \simeq ([v_1] - [\epsilon_1]) \cdot ([v_2] - [\epsilon_2]) = \underbrace{[v_1] \cdot [v_2]}_{A} - (\underbrace{[v_1] \cdot [\epsilon_2]}_{B} + \underbrace{[v_2] \cdot [\epsilon_1]}_{C}) + \underbrace{[\epsilon_1] \cdot [\epsilon_2]}_{D} \tag{3.7}$$

As shown in Equation 3.7, the operation expands into four distinct terms: the nominal product $[v_1] \cdot [v_2]$, the two cross terms $[v_1] \cdot [e_2]$ and $[v_2] \cdot [e_1]$, and the product of the errors $[e_1] \cdot [e_2]$. These terms collectively capture how approximated quantities influence one another when multiplied. To recover the interval representation $|ian| = \{[v], [\epsilon]\}$, the bounds are computed from the extremal combinations of these four terms, as detailed in Equation 3.8 and Equation 3.9.

$$|ian_1| \cdot |ian_2| \simeq \left\{ \underbrace{\left[\min(A),\ \max(A)\right]}_{[v]}, \underbrace{\left[\min(D - (B + C)),\ \max(D - (B + C))\right]}_{[\epsilon]} \right\}, \tag{3.8}$$

$$\begin{aligned} A &\rightarrow \left\{v_{1_{\min}} v_{2_{\min}},\ v_{1_{\min}} v_{2_{\max}},\ v_{1_{\max}} v_{2_{\min}},\ v_{1_{\max}} v_{2_{\max}}\right\}, \\ B &\rightarrow \left\{v_{1_{\min}} \epsilon_{2_{\min}},\ v_{1_{\max}} \epsilon_{2_{\min}},\ v_{1_{\min}} \epsilon_{2_{\max}},\ v_{1_{\max}} \epsilon_{2_{\max}}\right\}, \\ C &\rightarrow \left\{v_{2_{\min}} \epsilon_{1_{\min}},\ v_{2_{\max}} \epsilon_{1_{\min}},\ v_{2_{\min}} \epsilon_{1_{\max}},\ v_{2_{\max}} \epsilon_{1_{\max}}\right\}, \\ D &\rightarrow \left\{\epsilon_{1_{\min}} \epsilon_{2_{\min}},\ \epsilon_{1_{\min}} \epsilon_{2_{\max}},\ \epsilon_{1_{\max}} \epsilon_{2_{\min}},\ \epsilon_{1_{\max}} \epsilon_{2_{\max}}\right\}. \end{aligned} \tag{3.9}$$

This formulation applies to all multiplicative operations appearing in the SNN computation flow, including the weighted summation of spikes ($inp_i \cdot w_i$) and the exponential decay of the membrane potential. It provides a consistent means to determine upper and lower bounds on the results of such multiplications, maintaining correctness across both linear and nonlinear neuron operations.

Compared with classical IA, this explicit distinction between the nominal term and the error term simplifies reconfiguration for different bit truncation levels (k

values) without redefining the entire model. It also enables fast analytical estimation of how much accuracy degradation is introduced at each precision level, avoiding the need to test every configuration through full network simulation.

Right Shift and Comparison

An exponential decay approximated by a right shift divides both $[v]$ and $[\epsilon]$ by 2^n (Equation 3.10). The right shift operation models the exponential decay of the membrane potential between consecutive time steps. Implementing the decay as a right shift rather than a multiplication by an exponential term reduces computational complexity in hardware implementations. The IA formulation captures this by dividing both $[v]$ and $[\epsilon]$ by 2^n, scaling the potential and its associated error consistently over time. This ensures that both the membrane potential and the corresponding approximation error decay proportionally, accurately reflecting the neuron's behavior as it relaxes toward its resting potential when no input spikes occur [3].

$$|ian_1| \gg n = \{\underbrace{[v_{1_{min}} \gg n, v_{1_{max}} \gg n]}_{[v]}, \underbrace{[\epsilon_{1_{min}} \gg n, \epsilon_{1_{max}} \gg n]}_{[\epsilon]}\} \quad (3.10)$$

Interval-based comparisons handle potential overlaps by checking whether $|ian_1| - |ian_2|$ remains strictly positive (Equation 3.11) [3]. The comparison operation models the firing condition of the neuron, which occurs when the membrane potential V_0 exceeds the threshold V_{thresh}. Since both are represented as intervals, the comparison $|ian_1| - |ian_2| > 0$ must account for possible overlaps. The expression in Equation 3.11 determines whether, after subtracting the maximum possible error from the potential and adding it to the threshold, the potential still exceeds the threshold. If so, a spike is guaranteed; otherwise, the result is uncertain and marks a transition boundary.

$$|ian_1| - |ian_2| > 0 \Rightarrow |ian_r| > 0$$
$$(v_{r_{max}} - \epsilon_{r_{max}}) - (v_{r_{min}} - \epsilon_{r_{min}}) > 0 \quad (3.11)$$

This conservative comparison avoids false firing events and ensures that the modeled neuron behavior always encloses the real one. Although it may slightly overestimate possible spike occurrences, this guarantees safe and reliable evaluation of approximation effects across all network operations.

By redefining addition, subtraction, multiplication, shifting, and comparison within the IA framework, the model offers a consistent and computationally efficient method for propagating approximation effects through the entire neuron

computation flow. This capability forms the foundation for the subsequent exploration methodology, enabling fast evaluation of how bit-width reductions impact classification accuracy without exhaustive simulation.

3.1.4 Experimental Results for the Baseline Interval Arithmetic Approach

In this work, the SNN model is adopted from [86]. The SNN architecture comprises a single layer with 400 neurons. The SNN is trained on the MNIST [90] dataset, which consists of 70,000 grayscale images of handwritten digits (60,000 training images and 10,000 test images), each with a resolution of 28×28 pixels. Each 28×28-pixel image is converted into a time-discretized sequence of binary input spikes using a Poisson process [91], where the spike generation frequency is proportional to the pixel intensity. In the reference implementation, this Poisson input encoding is realized over a fixed presentation window of duration Δt with time resolution δt, yielding $T = \Delta t/\delta t$ sampling steps (e.g., $\Delta t = 350$ ms and $\delta t = 0.1$ ms, hence $T = 3500$ steps). After normalizing the pixel intensity to a rate, each pixel is mapped to a per-step spike probability, and at every time step a uniform random number is drawn so that the corresponding input bit is set to 1 when the draw falls below that probability and remains 0 otherwise; repeating this Bernoulli trial for T steps produces a binary spike train whose event statistics approximate a Poisson process for sufficiently small δt [86, 91]. This stochastic encoding introduces temporal variability consistent with probabilistic firing patterns, and it is kept identical across all evaluated configurations to ensure that performance differences are attributable to precision reduction rather than input variability.

The evaluations strive to answer the question of how accurately the proposed IA-based model can predict the approximation effect on the neuron spike counters. Also, the overall classification performance of the SNN is evaluated, when k bits of fractional precision are progressively truncated [3]. In other words, the experimental validation aims to verify whether the proposed IA model can reproduce the behavior of the real approximated network while drastically reducing the time required for performing a DSE. To ensure a fair comparison, the IA model and the approximated SNN receive identical sequences of spikes, preserving the stochastic spike distribution generated by the Poisson process. The IA-based evaluation uses the same computational flow as the real SNN (as in Figure 3.2) but replaces all arithmetic operations with their interval-based counterparts, thus estimating the possible deviations in membrane potential and spike counts introduced by bit-width reduction.

The entire experimental workflow follows two parallel branches. The first branch executes the original network after applying a k-bit precision reduction to its parameters (weights, thresholds, and reset values) and collects the actual spike counters produced by inference. The second branch executes the IA-based analytical model,

which operates on interval representations derived from the same parameters. By comparing the spike counters from both branches, one can quantitatively evaluate how closely the IA model approximates the real network's output and determine the precision range that preserves classification accuracy.

Single-Bit-Width Reductions

Table 3.1 reports the results for 520 test images across single-bit precision reductions, from $k = 1$ to 15. Deviations between the IA model and the corresponding approximated network remain small, rarely exceeding ± 3 spikes. The SNN classification results remain identical up to $k = 10$, after which the overall accuracy of the approximated network itself begins to degrade [86].

The error values reported in the table represent the difference between the spike counts predicted by the IA model and those observed in the real approximated SNN. Negative errors correspond to slightly lower spike counts predicted by the IA model, a result of its conservative comparison mechanism that may prevent a neuron from firing when its potential interval marginally overlaps the threshold range. This behavior aligns with the intentionally conservative nature of IA, which seeks to avoid false spike events rather than overestimate them.

Reduction (k-bits)	**Errors**			**Wrong Counters (%)**	**SNN Accuracy**
	Min	**Max**	**AVG**		
1	-1,00	1,00	0,0009	0,28	Same
2	-1,00	1,00	0,0011	0,33	Same
3	-2,00	1,00	0,0003	0,58	Same
4	-2,00	1,00	-0,0010	0,86	Same
5	-3,00	2,00	0,0000	1,60	Same
6	-3,00	1,00	-0,0010	2,38	Same
7	-2,00	2,00	-0,0030	3,44	Same
8	-2,00	2,00	-0,0031	4,77	Same
9	-2,00	2,00	-0,0070	5,31	Same
10	-2,00	2,00	-0,0074	6,16	Same
11	-2,00	2,00	-0,0107	6,86	Same
12	-2,00	2,00	-0,0299	7,34	Same
13	-3,00	3,00	-0,0415	7,12	Same
14	-4,00	2,00	-0,0431	5,38	Same
15	-3,00	2,00	-0,0256	2,49	Same

Table 3.1: IA model results for single-precision-reduction configurations (520 images). "Errors" = (IA minus approximated SNN) spike-counter deviations [3].

The table confirms that the IA model can consistently reproduce the spike counts of the approximated network with high precision. The average deviation

(AVG column) remains close to zero, showing that the IA-based analysis introduces negligible bias. Furthermore, the classification accuracy remains stable across reductions up to $k = 10$, demonstrating that precision reduction at this level does not compromise the network's decision boundaries. Beyond $k = 10$, both the real network and the IA model begin to lose classification reliability, highlighting the natural precision limit of the design.

Range-Based Approximations

Table 3.2 summarizes experiments where the precision reduction is analyzed over grouped k ranges (e.g., 1-5, 6-10, and 11-15). The IA model computes these results in a single run per range, providing a more efficient exploration strategy. The classification accuracy remains identical to that of the original network for all tested intervals [86].

Grouping multiple k values allows the IA model to evaluate a broader range of approximations simultaneously. While the reported average error (around three spikes) is slightly higher than in single-k experiments, the exploration time is significantly reduced to approximately one-fifth of the original duration. This confirms the scalability of the IA approach when used for coarse-grained DSE. Importantly, even under this accelerated configuration, the predicted classification results remain fully consistent with the real approximated SNN.

Reduction ($k_{\min}$ **to** $k_{\max}$)	**Errors**			**SNN Accuracy Comparison**
	Min	**Max**	**AVG**	
1 to 5	0	10	3,1777	Same
6 to 10	0	10	3,1950	Same
11 to 15	0	10	3,2776	Same

Table 3.2: IA model exploration of multiple bitwidth reductions in a single run (30 images). Classification remains "Same" [3].

3.1.5 Comparison and Observations

Compared with full brute-force evaluation [64, 65], the IA-based approach [92, 66] helps to rapidly identify feasible truncation levels while maintaining result fidelity. By maintaining explicit intervals for both the nominal value and the approximation error, the IA model efficiently pinpoints the safe truncation range (around $k = 10$) that preserves classification performance. Beyond this limit, the inherent network accuracy of the SNN itself declines [86], rendering small spike-count mismatches less significant.

Overall, these results demonstrate that the IA-based model provides an acceptable analytical approximation of the SNN behavior under precision reduction.

It reproduces both the quantitative (spike counts) and qualitative (classification) outcomes of the real approximated network while requiring only a fraction of the computational time. Therefore, the IA model serves as an effective tool for rapid DSE, identifying acceptable trade-offs between precision, accuracy, and resource utilization without exhaustive hardware or software simulation.

3.2 Fast Exploration of the Impact of Precision Reduction on Spiking Neural Networks

In this section, a more fine-grained DSE methodology based on the methodology introduced in [3] is proposed. This methodology augments the baseline IA model with *watchers*, enabling per-neuron or per-weight precision tuning [7].

3.2.1 Design Space Exploration Methodology Modified with Watchers

The concept of watchers was introduced to overcome one limitation of the baseline approach: the assumption of uniform precision reduction across all parameters. While the previous methodology could efficiently determine a global bit-width (k) that preserved network accuracy, it did not account for the varying sensitivity of individual neurons or parameters to quantization. In practice, some neurons can tolerate aggressive bit truncation without affecting classification accuracy, whereas others require higher precision to maintain stable firing behavior.

To address this, watchers act as observation points distributed throughout the computation flow of the SNN. Each watcher continuously monitors whether the approximation-induced error range, computed through the IA model, remains within an acceptable threshold. When a watcher detects that the propagated error has exceeded this threshold, it triggers an event that instructs the algorithm to revert part of the precision reduction. This adaptive process allows fine-grained control over the number of fractional bits retained for each parameter, producing a mixed-precision configuration that is both compact and accurate.

Instead of assigning a single uniform k to all parameters, watchers allow deeper truncation for some and revert to higher precision for others as needed. The *active watchers* are placed at neuron outputs and *hidden watchers* are placed after each arithmetic step, as in Figure 3.3. If a watcher indicates an unacceptable error range, the relevant parameters reduce k [7].

Two types of watchers are defined:

- **Hidden watchers**, placed after internal arithmetic operations (e.g., additions, multiplications, and decays), monitor intermediate neuron computations such as membrane potential updates.

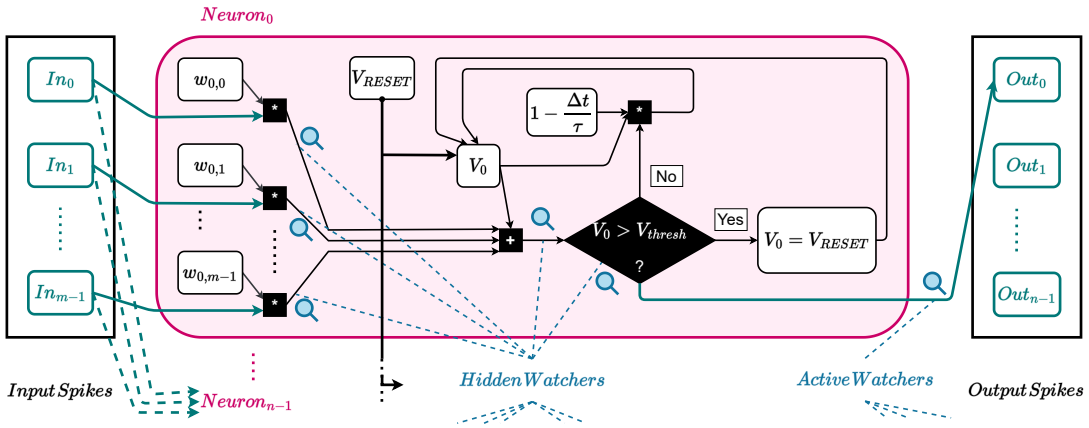

Figure 3.3: Computation flow of an SNN layer with watchers (*hidden* vs. *active*) [7].

- **Active watchers**, positioned at the outputs of neurons, assess whether the final membrane potential intervals and resulting spike decisions remain within the acceptable range. They serve as termination points that determine whether a neuron's computation satisfies the defined precision and accuracy criteria.

Each watcher operates on an IA representation of its monitored variable. The monitored value, represented as $|ian| = \{[v], [\epsilon]\}$, is compared with the predefined acceptable error interval. The acceptable interval is determined from the average of the minimum and maximum possible errors derived analytically in Equation 3.4. This ensures that all comparisons remain consistent with the IA error bounds established earlier in the methodology.

By leveraging these observation points, the modified exploration algorithm performs a feedback-driven refinement process. Initially, all parameters are assigned the maximum allowed precision reduction ($k = 15$). The system then executes the network computations using the IA framework and evaluates watcher outputs. Whenever an active watcher is triggered, the algorithm performs a backward check of hidden watchers to locate the sources of excessive error accumulation. Only the parameters associated with these specific computation paths are adjusted by decreasing their k value by one bit. This iterative process continues until no watcher triggers any violations, ensuring that the final configuration respects all accuracy constraints.

Figure 3.3 illustrates how watchers are inserted into the neuron computation flow. The hidden watchers, positioned after each arithmetic operation, track the evolution of the interval values within the neuron, while the active watchers at the outputs of neurons determine whether the neuron's firing state remains within the permitted error boundaries. This hierarchical structure allows a localized correction mechanism: rather than reconfiguring the entire network precision, only the affected

weights or thresholds are modified.

Algorithm 1 formalizes this process. Initially, the algorithm sets $k_{best}[i][j] = 15$ for all neuron parameters (lines 5-9). The IA model then simulates the entire network while watchers observe every computation stage. If no violation is detected, the process terminates; otherwise, the triggered watchers identify the parameters responsible for the violation, and the precision reduction for those parameters is decreased by one bit (lines 16-21). This adaptive loop continues until all watchers remain inactive, indicating convergence. The stopping criteria ensure that the exploration terminates either when all parameters have reached the minimum bit-width (no further reduction possible) or when all active watchers confirm that the approximation is within acceptable limits.

The proposed watcher-based mechanism transforms the global precision exploration problem into a distributed and self-correcting process. Instead of exhaustively evaluating every possible configuration of bit-truncations across all weights and thresholds, the system automatically converges towards a mixed-precision configuration. This significantly reduces the computational overhead of per-parameter exhaustive search while improving model compactness and preserving classification accuracy. The framework remains fully compatible with the IA representation introduced earlier, allowing the reuse of the same arithmetic and comparison rules.

algorithm 1 outlines the algorithmic implementation of this exploration process [7].

From an implementation perspective, each watcher is realized as a lightweight computational unit that reuses the interval comparison operator of the IA model. The comparison logic can be efficiently integrated into software-based simulation frameworks or synthesized into hardware for on-chip monitoring. Because the watchers operate in parallel, they can simultaneously evaluate multiple points in the computation flow, further accelerating the DSE.

In summary, the watcher-augmented DSE methodology extends the baseline model by incorporating distributed feedback and adaptive precision tuning. This combination enables fast, fine-grained, and reliable exploration of approximation configurations, ensuring that each neuron and parameter retains only as much precision as necessary to maintain functional correctness and classification accuracy.

3.2.2 Experimental Setup and Results

The watchers are applied to the SNN model from [93], again using the MNIST dataset (see Section 3.1.4 for dataset details) and the same SNN architecture with one layer of 400 neurons. Each iteration runs on a batch of images; if watchers never fire, the algorithm is halted. In about eight iterations, most neuron weights stabilize at $k = 10$, matching the best single-value approach [93], but some go to $k = 11$. Classification remains unchanged from the baseline. Figure 3.4 shows the fraction of neurons needing reversion each round [7].

```
1  N := Number of Neurons;
2  A := Number of Active Watchers;
3  H := Number of Hidden Watchers;
4  explorationDone ← False;
5  for i = 0; i < N; i++ do
6  |  for j = 0; j < H; j++ do
7  |  |  k_best[i][j] ← 15;
8  |  end
9  end
10 while explorationDone ≠ True AND ∀k_best > 0 do
11 |  explorationDone ← True;
12 |  runExploration();
13 |  for i = 0; i < N; i++ do
14 |  |  if actWatcher[i].watchFired() == True then
15 |  |  |  explorationDone ← False;
16 |  |  |  for j = 0; j < H; j++ do
17 |  |  |  |  if hidWatcher[i][j].watchFired() == True then
18 |  |  |  |  |  k_best[i][j] −= 1;
19 |  |  |  |  end
20 |  |  |  end
21 |  |  end
22 |  end
23 end
```

Algorithm 1: DSE algorithm using watchers [7].

The purpose of these experiments is to evaluate whether the proposed watcher-based exploration can reach an optimized bit-width configuration faster than the baseline IA model while maintaining equivalent classification performance. The same trained network is used as a reference to ensure a fair comparison, and the watcher-augmented framework operates on the identical computational flow and parameters.

For this evaluation, 1000 test images from the MNIST dataset were processed by both the reference and the watcher-based models. The same Poisson-based input encoding described in Section 3.1.4 is used throughout the watcher-based exploration to keep the input spike trains identical across iterations and configurations.

At each iteration, the exploration algorithm executes the IA-based model of the SNN and evaluates the outcome of all hidden and active watchers. If an active watcher detects that the output spike count or the final membrane potential interval has exceeded the acceptable deviation range, the algorithm inspects the corresponding hidden watchers to identify which arithmetic operations contributed to the violation. The associated parameters, typically neuron weights or thresholds, then have their precision increased by one bit for the next iteration. This adaptive refinement continues until all active watchers remain inactive, signifying convergence.

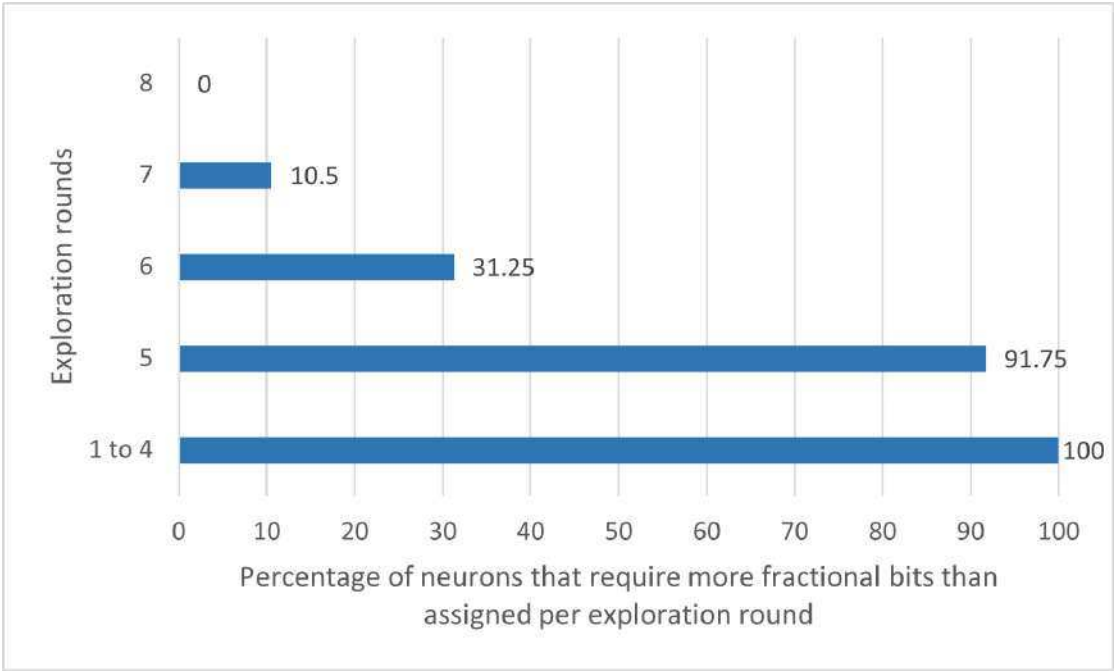

Figure 3.4: Percentage of neurons that require more fractional bits than assigned per exploration iteration [7].

Figure 3.4 illustrates the proportion of neurons whose associated weights required precision restoration during each exploration round. The percentage rapidly decreases after the initial iterations, demonstrating that the algorithm quickly identifies and isolates neurons that are more sensitive to bit truncation. By the eighth iteration, all active watchers remain inactive, confirming that the model reached a stable configuration.

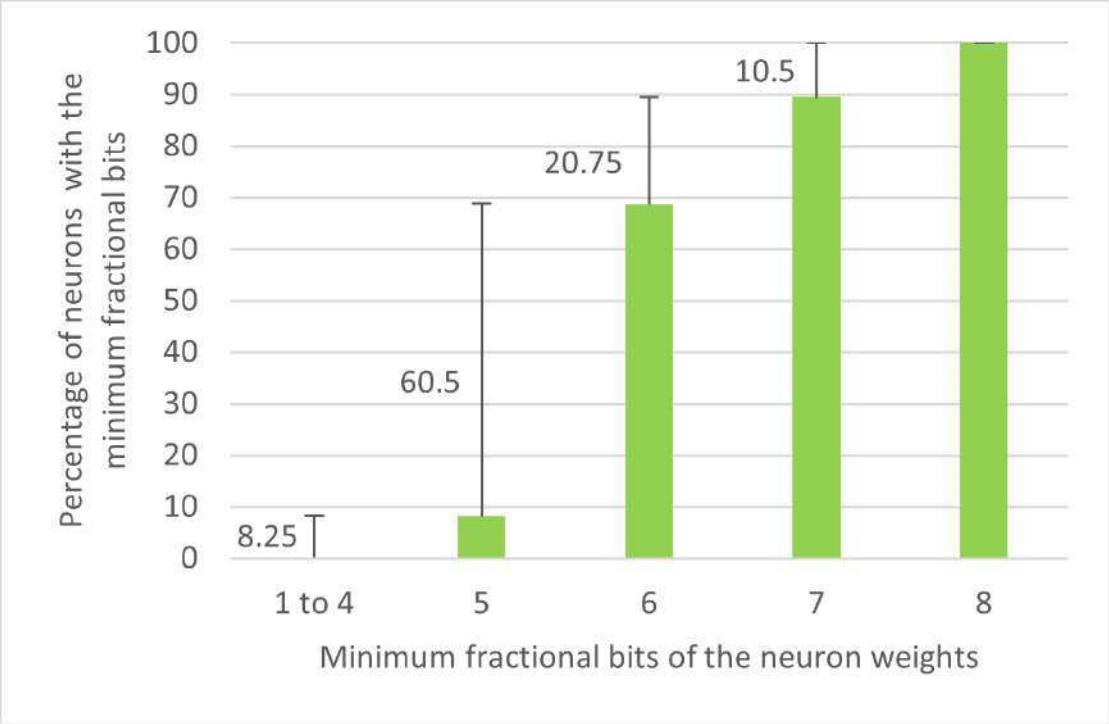

Figure 3.5: Percentage of neurons that require only the minimum fractional bits assigned per exploration iterations [7].

Figure 3.5 helps analyze the data from Figure 3.4 from another point of view. This figure illustrates the percentage of neurons that require only the minimum fractional bits assigned per exploration iteration. In the first iteration of exploration, in which only one fractional bit is preserved for each weight, none of the neurons can operate with such aggressive truncation. The same happens at the second iteration when all weights have one more additional fractional bit compared to the previous iteration, meaning that all weights have two fractional bits in their fixed-point representation. The same scenario repeats until the fourth iteration. Hence, in the plot, the percentage of neurons that could operate with all weights aggressively truncated into fixed-point values with four fractional bits is still zero. At the fifth iteration, the minimum fractional bit is set to five, and 8.25% of the neurons can operate with weights that have undergone bit truncation and are represented by a fixed point value that contains five fractional bits only. At the sixth

iteration of the exploration, the results show that 60.5% of the neurons can tolerate their weights undergoing a bit truncation that results in a fixed point representation with six bits kept for the fractional part. The column at the sixth iteration in the figure shows that, at this stage, 68.75% of the weights can be truncated, reducing their fractional bits to six bits or fewer. Hence, approximately, 70% of the weights are truncated and only 30% of the weights need more precision than six bits for the fractional part. Continuing the exploration in the seventh iteration, the results show that 20.75% of the remaining weights can tolerate a bit truncation that leaves seven bits for the fractional part and do not require more precision. As indicated in the plot, the cumulative value in the column showing the results at the seventh iteration suggests that 89.5% of the network weights are truncated to seven bits for the fractional part or fewer. At the eighth exploration iteration, the remaining 10.5% of the weights of the network are truncated to eight bits for the fractional part, and the network can tolerate such truncation. These exploration results were the most typical scenario among all explorations; hence, this example was chosen to explain the procedure.

The final configuration achieved a heterogeneous bit-width distribution across the network: while most weights converged at 10-bit truncation, several less critical neurons retained 11-bit truncation, allowing additional compression without classification loss. This outcome highlights one of the main advantages of the watcher-based strategy: its capacity to autonomously discover mixed-precision configurations rather than enforcing a uniform precision level across all neurons.

The classification accuracy of the final approximated network, as predicted by the watcher-augmented model, matched the accuracy of the baseline SNN with a fixed 10-bit truncation. This demonstrates that local precision adaptations did not degrade the network's ability to classify MNIST digits correctly. Moreover, the model preserved the same spike-counter statistics as the reference configuration within ±3 spikes, confirming that the refined mixed-precision network maintained functional equivalence with the baseline.

To evaluate the computational efficiency of the proposed DSE methodology, the watcher-based exploration was executed on a workstation equipped with an Intel Core i7 processor and 16 GB of RAM. For $N_{\text{img}} = 1000$ MNIST test images and $N_{\text{iter}} = 8$ exploration iterations, the measured end-to-end runtime was:

$$t_{\text{total}} = 539.842 \text{ s} \tag{3.12}$$

Accordingly, the average runtime per exploration iteration was:

$$t_{\text{iter}} = \frac{t_{\text{total}}}{N_{\text{iter}}} \approx 67.480 \text{ s} \tag{3.13}$$

Although this runtime is longer than a single forward inference of the original SNN, it is orders of magnitude faster than an exhaustive exploration of all possible bit-width assignments.

The hidden layer comprises $N_{\text{neur}} = 400$ neurons, each receiving $28 \times 28 = 784$ input pixels; therefore, the number of synaptic weights is:

$$N_w = N_{\text{neur}} \times 784 = 400 \times 784 = 313600 \tag{3.14}$$

Hence, the network includes 313600 weights, each of which could, in principle, assume 16 distinct precision states. In other words, the exploration considers $L = 16$ discrete precision levels, corresponding to retaining 1 to 16 fractional bits.

Two exhaustive-search interpretations are relevant, depending on whether weights are treated as distinguishable.

Case 1: Distinguishable weights (configuration identity depends on which weight gets which precision).

Because each of the N_w weights can independently select one of the $L = 16$ retained-fractional-bit levels, the number of configurations is:

$$N_{\text{cfg}}^{\text{dist}} = L^{N_w} = 16^{313600} \approx 1.06 \times 10^{377612} \tag{3.15}$$

Let t_{inf} denote the measured time for a single-image inference of the baseline model (here, $t_{\text{inf}} = 0.159$ s). Then, the corresponding upper bound on the runtime of brute-force exploration becomes:

$$t_{\text{bf}}^{\text{dist}} = N_{\text{cfg}}^{\text{dist}} \cdot t_{\text{inf}} = 16^{313600} \times 0.159 \text{ s} \approx 1.69 \times 10^{377611} \text{ s} \tag{3.16}$$

which is approximately 5.36×10^{377603} years (using 1 year = 31536000s). This bound formalizes the infeasibility of brute-force enumeration when per-weight precision is treated as a decision variable.

Case 2: Indistinguishable weights (only the counts of weights per precision level are tracked).

In some contexts, the exhaustive space is approximated by ignoring *which* weights take a given precision level and counting only *how many* weights are assigned to each of the $L = 16$ levels. Under this abstraction, each configuration corresponds to a nonnegative integer vector $(n_1, \ldots, n_{16})$ such that $\sum_{\ell=1}^{16} n_\ell = N_w$, where n_ℓ is the number of weights using level ℓ. The number of such allocations is given by the stars-and-bars count:

$$N_{\text{cfg}}^{\text{ind}} = \binom{N_w + L - 1}{L - 1} = \binom{313600 + 15}{15} \approx 2.13 \times 10^{70} \tag{3.17}$$

Considering t_{inf} as the measured time for a single-image inference of the baseline model ($t_{\text{inf}} = 0.159$ s), the corresponding upper bound on the runtime of brute-force exploration becomes:

$$t_{\text{bf}}^{\text{ind}} = N_{\text{cfg}}^{\text{ind}} \cdot t_{\text{inf}} = \left(2.13 \times 10^{70}\right) \times 0.159 \text{ s} \approx 3.39 \times 10^{69} \text{ s} \tag{3.18}$$

which is approximately 1.07×10^{62} years (using 1 year = 31536000s). This formulation yields a substantially smaller count than Equation 3.15, because it collapses many distinct per-weight assignments into the same allocation histogram. However, because different weights have different functional roles and sensitivities, distinct per-weight assignments that share the same histogram can still lead to different network behaviors; therefore, Equation 3.17 should be interpreted as a coarse abstraction rather than the true configuration space for mixed-precision assignment.

This quantitative gap motivates the watcher-based strategy. In contrast to brute-force enumeration of the mixed-precision design space under either counting assumption, the proposed method converges to a mixed-precision solution within a small number of exploration iterations and yields a practically tractable runtime (as calculated in Equation 3.12). By accounting for heterogeneity in weight sensitivity, the watcher mechanism selectively restores precision only where needed, thereby uncovering local resilience within the network and enabling higher compression of subsets of weights while preserving the imposed accuracy limitations.

In summary, the watcher-enhanced exploration maintains the predictive accuracy of the baseline IA model while reducing DSE time, relative to exhaustive search, by an astronomically large margin. The exhaustive bounds in Equation 3.16 and Equation 3.18 formalize that explicit enumeration of mixed-precision assignments is infeasible in practice, whereas the measured runtime in Equation 3.12 demonstrates that the proposed framework achieves a usable end-to-end evaluation cost. Consequently, the framework provides an effective trade-off among analytical conservatism, computational overhead, and model compression, making it suitable for rapid optimization of neuromorphic systems under hardware constraints.

3.2.3 Discussion and Conclusion

Watcher-based DSE reveals that some neurons can tolerate more aggressive bit removal without harming classification. The final design thus has mixed precision. Despite a huge combination space, watchers converge in a handful of steps. This method can be potentially adopted to perform DSE for other networks or approximate operators, balancing resource constraints with the acceptance of minimal error [7].

The experimental findings confirm that integrating watchers into the DSE framework effectively enhances exploration granularity without compromising accuracy. By introducing localized observation points, the system can selectively refine precision where necessary while maintaining lower bit-widths for parameters that exhibit higher resilience to quantization. This leads to a mixed-precision network configuration that remains functionally equivalent to the original model yet more compact in size.

Compared to the baseline IA-based approach, the watcher-augmented method

converges quickly, typically within a few iterations, demonstrating a favorable balance between exploration depth and computational efficiency. The use of interval-based monitoring ensures that each adjustment remains analytically traceable, preserving the conservative accuracy guarantees of the underlying IA model.

Overall, this methodology provides a solution for fast precision optimization in neuromorphic systems. Its adaptive and modular design enables an avenue for extension to other AxC scenarios, serving as a foundation for the broader discussion on general applicability and future research directions presented in the next section.

3.3 Overall Conclusions and Future Directions

This chapter presented two complementary methodologies that leverage IA to accelerate the DSE of AxC techniques applied to SNNs. The first methodology introduced a baseline IA-based analytical model capable of predicting the effects of precision reduction without re-simulating the entire network. The second methodology enhanced this framework through the introduction of watcher mechanisms, enabling fine-grained precision tuning at the level of individual neurons and parameters. Together, these approaches form a coherent exploration strategy that balances modeling accuracy with computational efficiency.

The baseline IA model demonstrated that the propagation of quantization-induced errors through neuron computations can be captured using interval-based formulations. This allowed for precise identification of safe truncation levels (around $k = 10$) that preserve the original classification accuracy of the network. By avoiding repeated inference runs for each configuration, the IA-based analysis significantly reduced exploration time, offering a practical solution for evaluating large search spaces in hardware-constrained environments.

The watcher-augmented approach extended the analytical framework by incorporating localized monitoring and adaptive feedback. This improvement enabled the system to automatically refine bit-width assignments per neuron or per connection, resulting in a mixed-precision configuration that further optimized memory and computation without sacrificing classification accuracy. The method's rapid convergence and scalability confirmed its effectiveness as a fine-grained precision exploration tool, particularly for architectures with heterogeneous sensitivity to quantization.

Collectively, the proposed techniques highlight the potential of IA-based exploration in bridging the gap between approximate algorithm design and practical hardware implementation. The analytical nature of these models eliminates the need for costly retraining or exhaustive simulation, providing early insights into the trade-offs between energy efficiency, resource utilization, and accuracy degradation. Such capabilities make these approaches suitable for deployment in hardware-aware design flows targeting neuromorphic and embedded computing platforms.

Future research can expand upon these contributions in several directions. First, integrating the watcher-based precision tuning into automated hardware synthesis tools could further accelerate the exploration-to-implementation pipeline. Second, extending the approach to larger and deeper neural architectures, including multi-layer SNNs or hybrid ANNs-SNNs models, would test its scalability and generality. Third, coupling interval-based analysis with other approximation techniques, such as approximate operators or voltage overscaling, could enable multi-dimensional exploration across accuracy, energy, and reliability trade-offs. Finally, embedding the watcher logic directly in neuromorphic hardware could facilitate real-time self-calibration, allowing systems to dynamically adjust precision in response to operational conditions or energy constraints.

In conclusion, the methodologies developed in this chapter suggest potential broader use for designing frameworks to perform analytical and adaptive DSE while applying AxC techniques to NNs such as SNNs. They represent an essential step toward efficient, energy-aware neuromorphic design, combining mathematical rigor with hardware practicality and paving the way for broader applications of IA-based exploration in emerging computing paradigms.

Chapter 4

Design Space Exploration Using Reinforcement Learning

This chapter presents an RL approach to multi-objective DSE for AxC techniques. The proposed methodology [6] selectively replaces precise arithmetic operations (additions and multiplications) with approximate ones in a CPU-based scenario. The objective is to balance accuracy, power consumption, and computation time, as well as to automate the selection of which variables in the code should be approximated.

4.1 Overview of the Proposed Approach

The idea is to generate an approximate version of a target application by selectively invoking approximate adders and multipliers from a known library whenever such approximation is expected to provide benefits in optimizing the considered objectives [6]. This work extends the mechanism proposed in [94] by:

- Identifying operations after which *variables* in the application code can be approximated without excessive loss in accuracy.
- Deciding which approximate *adder* and *multiplier* to use, each with different impacts on accuracy, power, and execution time trade-offs.
- Ensuring accuracy degradation remains under a certain threshold while maximizing power/time improvements.

Even with modest-sized operator libraries, the number of approximate configurations grows combinatorially with the number of decision sites due to independent choices of approximate adders/multipliers and per-variable approximation masks. This phenomenon is widely reported for exploring the design space after applying AxC techniques, across layers and operator libraries [43, 42, 38, 34]. Consequently,

exhaustive enumeration typically becomes infeasible beyond small instances, motivating a guided search policy that prioritizes promising regions subject to accuracy constraints.

Figure 4.1 illustrates the RL *environment*, where the RL *agent* chooses which approximate operators to use and which variables to approximate. The environment measures accuracy, power, and computation time for the resulting design [6].

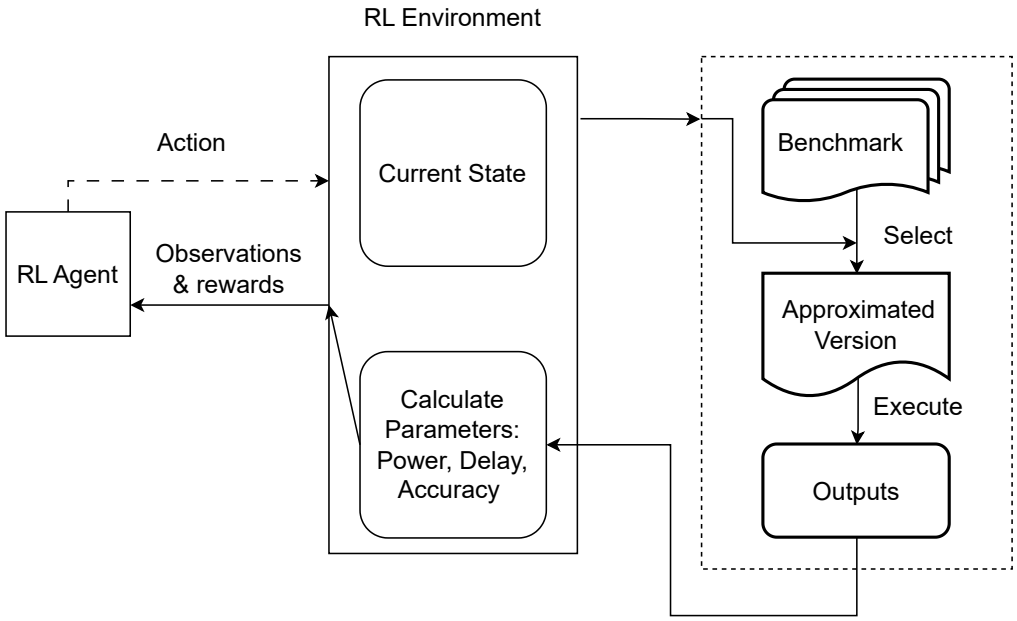

Figure 4.1: RL environment: at each step, it selects an approximate version of the benchmark and returns the measured accuracy, power, and time [6].

4.2 Reinforcement Learning Background and Algorithm Selection

In this section, an overview of the RL concept, definitions, algorithms, and the overall setup for the proposed approach are explained.

4.2.1 Reinforcement Learning Description

RL is a subfield of ML concerned with how agents ought to take actions in an environment to maximize cumulative reward. Unlike supervised learning, which learns from labeled input-output pairs, or unsupervised learning, which finds structure in unlabeled data, RL focuses on sequential decision-making, where an agent

learns by interacting with an environment and observing the outcomes of its actions [95].

At the core of RL lies the Markov Decision Process (MDP), which models the environment as a tuple $(\mathcal{S}, \mathcal{A}, \mathcal{P}, \mathcal{R}, \gamma)$, where:

- $\mathcal{S}$ is the set of all possible *states* the environment can be in.
- $\mathcal{A}$ is the set of all possible *actions* the agent can take.
- $\mathcal{P}(s'|s, a)$ is the *transition probability* of moving to state s' after taking action a in state s.
- $\mathcal{R}(s, a)$ is the *reward function* that maps a state-action pair to a scalar reward.
- $\gamma \in [0,1]$ is the *discount factor*, controlling the importance of future rewards.

The goal of an RL agent is to learn a *policy* $\pi : \mathcal{S} \to \mathcal{A}$ that maximizes the expected cumulative reward, or *return*, over time. Depending on the application, policies can be deterministic or stochastic.

The agent typically learns from experience through trial and error, adjusting its policy based on observed transitions and rewards. This interaction is formalized as a sequence of steps:

1. The agent observes the current state $s_t \in \mathcal{S}$.
2. It selects an action $a_t \in \mathcal{A}$ according to its policy π.
3. The environment responds with a new state s_{t+1} and a scalar reward r_t.
4. The agent updates its internal knowledge (e.g., value estimates or policy parameters).

RL methods are broadly categorized into:

- **Value-based methods**, which estimate value functions to derive policies (e.g., Q-Learning).
- **Policy-based methods**, which directly optimize the policy without estimating value functions.
- **Actor-Critic methods**, which combine both approaches to reduce variance and improve convergence.

In the context of system-level design and AxC, RL is particularly attractive for its ability to handle:

- **Multi-objective optimization**, where the agent balances competing metrics such as power, accuracy, and performance.
- **Non-differentiable and black-box environments**, typical in hardware-software systems where analytical gradients are unavailable.
- **Sparse feedback**, as the agent receives performance metrics only after testing a complete design configuration.

Recent studies have successfully applied RL to hardware optimization [96, 97], Neural Architecture Search (NAS) [98], and compiler optimization [99, 100], motivating its use for DSE in AxC. These properties make RL a suitable candidate for navigating large, discrete design spaces, potentially with improved sample efficiency compared to exhaustive search, depending on tuning and the benchmark.

In this work, RL is used to train an agent that selects AxC operators and identifies which variables in a codebase can be approximated. By observing the resulting changes in accuracy, power consumption, and execution time, the agent gradually learns to choose approximation configurations that can potentially deliver the most favorable trade-offs. The following subsection details the specific learning algorithms considered for this task.

4.2.2 Reinforcement Learning Algorithms

In RL, the choice of learning algorithm is critical to both the efficiency and effectiveness of the exploration process. Various RL algorithms have been proposed to solve MDPs, which can be broadly categorized into three classes: value-based methods, policy-based methods, and actor-critic methods [101].

Value-Based Methods

Value-based methods, such as Q-Learning and SARSA, estimate the expected return (value) of taking an action in a given state and then follow the greedy policy with respect to the learned value function. These methods are well-suited for problems with discrete and relatively low-dimensional state-action spaces [102, 103].

- **SARSA (State-Action-Reward-State-Action)** is an on-policy algorithm that updates its action-value function based on the action actually taken by the agent. While it is more conservative than Q-Learning, it often converges to safer policies in environments with high variability.
- **Q-Learning** is an off-policy algorithm that learns the optimal action-value function independent of the agent's actual actions, enabling more aggressive exploration and faster convergence in deterministic environments.

Policy-Based Methods

Policy-based methods directly learn the policy function that maps states to actions without estimating value functions. These methods are suitable for high-dimensional or continuous action spaces [104].

- **REINFORCE** is a basic policy gradient algorithm that updates the policy parameters in the direction that maximizes expected reward [105].
- These methods are particularly useful when the action space is continuous or when stochasticity in action selection is beneficial.
- However, they tend to suffer from high variance in the gradient estimate and require large sample sizes, making them less suitable for problems with constrained exploration budgets like DSE.

Actor-Critic Methods

Actor-critic methods combine value-based and policy-based methods. The *actor* updates the policy based on feedback from the *critic*, which evaluates the current policy by estimating the value function [106].

- Examples include PPO (Proximal Policy Optimization) and A2C (Advantage Actor-Critic) [107, 108].
- These methods are powerful and widely used in modern RL applications, including games and robotic control.
- Their complexity and computational overhead make them less ideal for tabular or small-scale DSE tasks where simplicity and interpretability are preferred.

Given the nature of the design space, discrete actions, finite state space, limited computational budget, and the need for explainability, value-based methods offer the most promising solutions. In particular, Q-Learning is chosen for its simplicity, proven convergence in tabular domains, and successful application in other AxC contexts [45, 67, 46], as detailed in the next subsection.

Q-Learning

The DSE problem considered in this work involves discrete and finite decisions: selecting AxC operators from a fixed library and choosing which program variables to approximate. This naturally formulates an MDP with a finite state and action space. In such scenarios, *Q-Learning* [101] is a well-established model-free RL algorithm suitable for learning optimal policies through value iteration. It does

not require an explicit model of the environment, which is advantageous in this context where the approximation-induced behavioral model is complex and non-differentiable.

The choice of Q-Learning over other RL algorithms is driven by several factors:

- The **action space is discrete**, involving the selection of arithmetic operators and toggling of approximation flags. Q-Learning is highly effective in such tabular settings without requiring function approximation.
- The **state space, though large, is finite and structured**, making value-based learning more interpretable and easier to implement compared to policy-gradient or deep RL methods.
- Q-Learning offers a sample-efficient and computationally lightweight approach, suitable for DSE loops with bounded training steps and constrained simulation time.
- Previous studies in resource-constrained DSE tasks, such as [109], have successfully employed Q-Learning for related optimization problems with similar design space characteristics, which supports the choice as reasonable for the present setting.

Convergence Considerations: In theory, Q-Learning is guaranteed to converge to the optimal policy under the following standard assumptions [110, 111]:

- The MDP has finite states and actions.
- The learning rate α_t satisfies the Robbins-Monro conditions: $\sum_t \alpha_t = \infty$ and $\sum_t \alpha_t^2 < \infty$.
- All state-action pairs are visited infinitely often.
- The reward function is bounded.

While the environment satisfies several of these conditions, namely, a finite MDP and bounded rewards, the proposed method does not seek to fulfill the requirement of infinite exploration due to the practical constraint of a finite exploration budget (up to 10000 steps). Moreover, learning rates and reward shaping are tailored to accelerate practical convergence rather than to fulfill theoretical guarantees.

Therefore, in this work, no claims of theoretical convergence to the globally optimal policy can be made. Instead, this work relies on empirical convergence assessed by:

- Stabilization of the agent's cumulative reward,

- meeting the selected thresholds in observed runs for power and execution time objectives while maintaining accuracy within user-defined bounds,
- Observed saturation in performance improvement across additional training steps.

This empirical approach is common in applied RL for system-level design tasks, where exhaustive state-action visitation is infeasible, but near-optimal trade-offs are sufficient and desirable. Future work may explore deep RL methods with function approximation to handle larger design spaces or establish probabilistic guarantees for convergence under partial exploration.

4.2.3 Reinforcement Learning Instrumentation

In a typical RL framework [112], an *agent* interacts iteratively with an *environment* by performing actions and receiving observations (states) and corresponding rewards. In this work, the RL framework specifically targets the automated DSE of AxC techniques, aiming at balancing accuracy degradation with power and execution time improvements [6]. The detailed RL setup comprises four essential components: the *environment*, the *state*, the *actions*, and the *reward function*. Each component is precisely defined below.

Environment Definition

The environment encapsulates the entire optimization problem context and responds dynamically to the agent's actions. Formally, the environment is defined as follows:

$$environment = \{\, adder,\ multiplier,\ variables_{approx},\ \Delta acc,\ \Delta power,\ \Delta time \} \quad (4.1)$$

The parameters in Equation 4.1 represent:

- **Adder, Multiplier**: Integer indices representing selected AxC operators from a predefined library of adders and multipliers (detailed in Tables 4.1 and 4.2). These are sorted based on their accuracy degradation (MRED).
- **Variables to Approximate** ($variables_{approx}$): A binary vector indicating whether each variable in the target program is approximated (`1`) or kept precise (`0`). Formally, $variables_{approx} = \{v_0, v_1, \ldots, v_{N-1} \mid v_i \in \{0,1\}\}$.

- **Accuracy degradation (Δacc)**: Measured as the MAE between the outputs of the precise and approximate versions of the application, computed as follows:

$$MAE = \frac{1}{N} \sum_{i=0}^{N-1} |exactOutput_i - approxOutput_i| \tag{4.2}$$

  where N is the total number of outputs.

- **Power and Time Improvements ($\Delta power, \Delta time$)**: These quantify the reduction in power consumption and execution time, respectively, comparing the approximate execution against the precise baseline:

$$\Delta power = power_{precise} - power_{approx} \tag{4.3}$$

$$\Delta time = time_{precise} - time_{approx} \tag{4.4}$$

This environment is illustrated in Figure 4.1, showing the interaction cycle clearly.

State Representation

The *state* in this RL setup represents the current configuration of the approximate design, along with the associated evaluation metrics:

$$state = (adder, multiplier, variables_{approx}, \Delta acc, \Delta power, \Delta time) \tag{4.5}$$

For instance, a typical state can be represented as $(4, 5, (1,0,\ldots,0), 30, 100, 200)$, indicating that approximate adder number 4 and multiplier number 5 are used, only the first variable is approximated, and the observed changes are 30 units of accuracy degradation, 100 mW power reduction, and 200 ns time improvement.

Action Space

The action defines the decision taken by the RL agent at each iteration, modifying the design configuration to explore new trade-offs. Specifically, the action space consists of three types of actions:

1. Selecting a different approximate adder from the library.
2. Selecting a different approximate multiplier from the library.
3. Toggling the approximation status (precise to approximate or vice versa) of one of the program variables.

The action space is discrete and finite, corresponding directly to the available operators and variables in the application.

Reward Function

The reward function guides the agent toward favorable configurations, balancing power/time improvements and accuracy constraints. Algorithm 2 formally details the reward assignment logic:

Input: R (Maximum cumulative reward), R_{cum} (Cumulative reward), *state* (adder, multiplier, variables, $\Delta acc, \Delta power, \Delta time$)

```
1  if Δacc ≤ acc_th then
2  |  if (adder=max AND multiplier=max AND all variables approximated)
   |   then
3  |  |  reward = R
4  |  |  terminate = True
5  |  end
6  |  else
7  |  |  if (Δpower ≥ p_th AND Δtime ≥ t_th) then
8  |  |  |  reward = 1
9  |  |  end
10 |  |  else
11 |  |  |  reward = -1
12 |  |  end
13 |  end
14 end
15 else
16 |  reward = -R
17 end
18 R_cum += reward
```

Algorithm 2: RL reward assignment at step i

This function is constructed to encourage the agent to select configurations with meaningful power and performance improvements while strictly penalizing designs exceeding the defined accuracy threshold (acc_{th}). Positive rewards incentivize beneficial changes, while penalties guide the agent away from suboptimal regions of the design space.

Algorithm 2 analyzes the current state. A further approximation can be introduced if the accuracy loss is below the tolerable accuracy loss threshold for the benchmark (line 4). This acceptable threshold is an exploration parameter and can be adapted to the case. If the most-approximated adder and multiplier are in use and all variables are approximated, the maximum reward possible is given, and the "terminate" flag is set to true (lines 5 to 8). Otherwise (lines 9 to 16), the gain in power consumption and computation time must be above a threshold to obtain a positive reward (+1). Otherwise, the agent gets a negative reward (-1). Line 19

gives the maximum negative reward when the accuracy loss exceeds its threshold. Finally, the cumulative reward is updated (line 21).

Learning Algorithm: Q-Learning

In this work, the Q-Learning algorithm is chosen for its simplicity and effectiveness in discrete, finite action spaces, as detailed in Section 4.2.2. The Q-function update is defined as follows [101, 113, 114]:

$$Q(s,a) \leftarrow Q(s,a) + \alpha \left[r + \gamma \max_{a'} Q(s',a') - Q(s,a) \right] \tag{4.6}$$

where:

- $Q(s,a)$ is the current estimated value of taking action a in state s.
- α is the learning rate, controlling the impact of new experiences.
- γ is the discount factor, influencing the importance of future rewards.
- r is the immediate reward received after taking action a.
- s' and a' are the subsequent state and the best subsequent action, respectively.

Through repeated interactions, the Q-function converges towards the optimal policy that balances accuracy, power, and execution time effectively.

By combining the comprehensive RL setup described above with systematic code instrumentation and evaluation of each configuration's performance metrics, the proposed methodology automates the exploration of complex approximation design spaces.

4.3 Approximate Components Database and Instrumentation

The selection and accurate characterization of AxC operators is fundamental to the success of the proposed RL-based DSE methodology. To this end, in this work `EvoApproxLib` [4, 5] is utilized. `EvoApproxLib` is an extensively employed open-source library providing hardware-oriented AxC circuits. Specifically, `EvoApproxLib` provides configurable implementations of approximate adders and multipliers, characterized by accuracy, power consumption, and execution time.

The library contains C models and synthesis data for 8-bit and 16-bit adders and 8-bit and 32-bit multipliers, generated through EAs that aim to optimize specific objectives, such as energy and area consumption, under error constraints. These AxC units serve as the building blocks for the approximate version of the targeted

benchmarks. The selected operators are summarized comprehensively in Tables 4.1 and 4.2, sorted by their MRED, a critical metric that quantifies computation accuracy loss.

For 32-bit multipliers, due to the absence of explicitly accurate implementations within `EvoApproxLib`, the corresponding parameters are derived by extrapolating the power and execution time reduction factors from the 8-bit multipliers. Specifically, the power and timing data for precise and minimally approximate 32-bit multipliers are directly computed by applying scaling factors obtained from equivalent operations in the 8-bit domain, ensuring consistency in the characterization approach and enabling reliable DSE outcomes.

Table 4.1 lists the selected approximate adders along with their key characteristics, namely, the type identifier, computation accuracy expressed as MRED, power consumption, and computation time. Similarly, Table 4.2 provides detailed characteristics for selected multipliers, facilitating systematic exploration and accurate trade-off analysis during the RL-guided exploration.

Incorporating these operators into the target application requires systematic code instrumentation, a crucial process that enables automated exploration by the RL agent. Instrumentation involves replacing all arithmetic operations (additions and multiplications) on variables selected for approximation with their corresponding approximate versions from the library. Specifically, the RL agent dynamically chooses the indices of the AxC operators from the pre-characterized library (Tables 4.1 and 4.2) and identifies the variables in the target codebase to approximate. This process is achieved through software-level instrumentation using conditional compilation or runtime selection based on the RL agent's decisions.

The instrumentation ensures:

- Precise baseline execution for each benchmark, providing a reference for evaluating approximation-induced trade-offs.
- Execution of the approximate versions of benchmarks, configured according to the RL agent's current policy decisions (selected arithmetic operators and variables).
- Accurate measurement of the output values for both precise and approximate executions, enabling rigorous computation of accuracy degradation metrics, primarily the Mean Absolute Error (MAE) as defined in Equation (4.2).
- Accurate estimation of power consumption and execution time by referencing the pre-characterized values from the library.

By systematically integrating `EvoApproxLib`'s characterized AxC components through instrumentation, the RL agent is enabled to navigate effectively through a complex and discrete design space, exploring potential trade-offs among computation accuracy, power efficiency, and computational performance. The rigorous

definition, instrumentation, and characterization process described herein provides a structured foundation to support reproducible evaluation of the explored configurations.

Table 4.1: Selected adders from EvoApproxLib [4, 5, 6].

Operator	Type	MRED	Power (mW)	Time (ns)
8-bit adder	1HG	0	0.033	0.63
8-bit adder	6PT	0.14	0.029	0.55
8-bit adder	6R6	2.93	0.012	0.27
8-bit adder	0TP	6.16	0.0095	0.24
8-bit adder	00M	14.58	0.0046	0.17
8-bit adder	02Y	24.87	0.0015	0.11
16-bit adder	1A5	0	0.072	1.28
16-bit adder	0GN	0.005	0.057	1.04
16-bit adder	0BC	0.018	0.051	0.95
16-bit adder	0HE	0.16	0.036	0.68
16-bit adder	0SL	9.54	0.011	0.27
16-bit adder	067	22.35	0.0041	0.20

Table 4.2: Selected multipliers from EvoApproxLib [4, 5, 6].

Operator	Type	MRED	Power (mW)	Time (ns)
8-bit multiplier	1JJQ	0	0.391	1.43
8-bit multiplier	4X5	0.033	0.380	1.40
8-bit multiplier	GTR	1.23	0.303	1.46
8-bit multiplier	L93	4.52	0.178	1.11
8-bit multiplier	18UH	17.98	0.062	0.90
8-bit multiplier	17MJ	53.17	0.0041	0.11
32-bit multiplier	precise	0	10.76	4.565
32-bit multiplier	000	0.00	10.46	4.470
32-bit multiplier	018	0.01	4.32	3.220
32-bit multiplier	043	1.45	1.63	2.440
32-bit multiplier	053	10.59	1.05	2.030
32-bit multiplier	067	41.25	0.51	1.750

4.4 Experimental Setup

The experiments conducted in this study [6] employ a structured setup to evaluate the feasibility and applicability of the proposed RL-based DSE approach and

characterize its behavior on the selected benchmarks. The experimental evaluations utilize two fundamental computational kernels widely recognized for their significance and frequent use in real-world engineering applications, specifically Matrix Multiplication and the Finite Impulse Response (FIR) filter. The employed tools, input generation procedure, and benchmark configurations are detailed in this section.

4.4.1 Tools for Reinforcement Learning Implementation

To implement the RL environment, the `Gymnasium` library [115], is utilized, which is a maintained and enhanced fork of OpenAI Gym [116]. `Gymnasium` is an open-source Python toolkit specifically designed to provide a standardized and reproducible RL environment, with the aim of developing, testing, and comparing RL algorithms. It provides a standardized API to create and interact with custom environments, thus enabling systematic experimentation and reproducible results. The key advantage of employing `Gymnasium` lies in its modularity and extensive community support, facilitating integration with various RL algorithms and tools for debugging and performance analysis.

4.4.2 Benchmarks

Regarding benchmarks, two application kernels were selected: Matrix Multiplication and the FIR filter. Each of these benchmarks is carefully chosen due to its fundamental role and wide applicability in software and hardware domains, especially in applications relevant to AxC. For each kernel, two representative problem sizes are evaluated to support an initial assessment of scalability and robustness of the proposed exploration strategy under distinct computational characteristics.

Matrix Multiplication is a cornerstone operation extensively utilized in various computational domains, including scientific computing, graphics processing, ML, and especially NNs [117, 118]. Its computational intensity, combined with repetitive, structured arithmetic operations, makes matrix multiplication an ideal candidate for exploring the application of AxC techniques. Approximation can substantially reduce computation time and power consumption in matrix-heavy workloads, a vital consideration in energy-constrained and embedded systems.

To evaluate Matrix Multiplication applications in this work, benchmarks with two representative sizes are used:

- 10×10 **matrices**: providing insights into small-scale computational kernels common in embedded and real-time processing applications.
- 50×50 **matrices**: representing moderate complexity workloads found in intermediate computational layers of various data-processing and NN inference tasks.

These two distinct sizes of the same benchmark are selected to provide a comparative understanding of the scalability of the proposed method.

The **FIR filter** is widely employed in Digital Signal Processing (DSP) applications, including audio processing, image processing, and telecommunications. FIR filters are characterized by their feed-forward structure, inherent numerical stability, deterministic latency, exact linear-phase response, and robustness to coefficient quantization, which makes them particularly suitable for real-time signal processing and hardware-oriented implementations under finite-precision constraints [119, 120, 121]. Applying AxC techniques when implementing FIR filters may reduce resource utilization without substantially degrading the quality of the filtered outputs, especially in noise-tolerant applications such as audio and image enhancement.

For FIR filter benchmarks in this work, a 32-tap FIR filter (a filter with the length of 32) evaluated with two input sizes is used:

- **100-sample filter**: representative of moderate complexity filters employed in real-time audio and small-scale sensor-data processing.
- **200-sample filter**: capturing more demanding filtering tasks, typical in applications requiring higher frequency resolution or improved noise suppression capabilities.

White noise inputs are employed to thoroughly examine the robustness of the FIR filter implementations against arbitrary input signals.

The experimental setup adopted in this chapter is designed to evaluate the feasibility and effectiveness of the proposed RL-based DSE methodology rather than to provide an exhaustive performance comparison across all possible approximation strategies. Therefore, the experimental evaluation does not include a comparison against a baseline configuration in which a single approximate adder and multiplier are uniformly applied throughout the entire benchmark. Such fixed-approximation scenarios have been extensively analyzed in prior AxC studies and do not involve any form of DSE. Since the focus of this work is on evaluating the ability of the proposed RL-based framework to autonomously discover heterogeneous approximation configurations, comparisons against trivial fixed baselines were not considered in this study.

Despite selecting widely applicable and representative kernels, it is acknowledged that the benchmark suite currently includes only a limited number of examples, particularly just one type per use-case category (matrix operation and signal processing). This limited benchmark set does not allow full generalization to all application domains; instead, it provides a controlled and well-understood experimental context for validating the proposed methodology and analyzing its behavior under distinct computational characteristics. Also, it is worth noting that even in the case that the results demonstrate the feasibility of the proposed RL-based

DSE approach within these selected cases, generalizing these results to all possible application domains requires careful consideration and further experimentation. Extending the benchmark suite to cover a broader range of representative applications (e.g., larger-scale NNs, multimedia encoding algorithms, or diverse DSP applications) remains essential future work to comprehensively validate the general applicability and scalability of the proposed methodology.

Nevertheless, the current choice of benchmarks aims to represent foundational computational tasks whose performance characteristics, while applying AxC techniques at the hardware or software level, are well-documented and understood. They serve as an effective baseline for initial evaluation and validation of the proposed RL-based DSE framework, demonstrating potential benefits and facilitating focused comparisons with existing AxC techniques.

4.4.3 Approximate Operators Library

The AxC operations (additions and multiplications) from the `EvoApproxLib` library [4, 5] are selectively integrated into the benchmarks as directed by the RL agent. The selected AxC operators from `EvoApproxLib` are detailed in Tables 4.1 and 4.2. All benchmarks underwent systematic instrumentation to facilitate accurate measurement of performance, including power consumption, execution time, and accuracy degradation. Power and time improvements ($\Delta power$ and $\Delta time$) are determined relative to precise baseline executions.

4.4.4 Selecting Thresholds and Metrics

The thresholds employed in the reward function were determined based on systematic considerations and empirical observations derived from preliminary experimental evaluations tailored explicitly for the benchmarks considered in this study. Specifically, the power (p_{th}) and computation time (t_{th}) thresholds were set at 50% of their precise baseline values. The appropriateness of these thresholds can vary significantly across different benchmarks and optimization goals. Consequently, setting these thresholds should always depend explicitly on the characteristics of the specific benchmark being analyzed and the optimization goals.

The accuracy degradation threshold (acc_{th}) was set to 0.4 times the average precise output. This threshold was established through preliminary experiments and systematic empirical analyses of the results for the chosen benchmarks. Specifically, after conducting extensive preliminary tests involving numerous trials using available approximations, it was concluded that this threshold maintains acceptable accuracy levels for the examined benchmarks. However, this threshold value cannot universally apply to all benchmarks or approximation techniques. In practice, the acceptable accuracy threshold must be specified by the user based on the target application tolerance to approximation-induced errors prior to initiating the

DSE. Consequently, adaptive thresholding schemes or more sophisticated accuracy metrics, tailored explicitly for particular application domains, should be considered in future studies.

Selecting the Maximum Number of Reinforcement Learning Steps

The maximum number of exploration steps for the Q-learning algorithm was empirically set to 10000 based on systematic trial-and-error evaluations. While an exhaustive search theoretically provides an absolute upper bound on complexity, it rapidly becomes infeasible due to the complexity growth associated with the combinatorial exploration of all AxC operators and variable approximation choices. For instance, an exhaustive search, even for modest problem sizes, would require millions of evaluations, which may exceed computational feasibility. Thus, the selected 10000-step limit offers a pragmatic compromise between the comprehensiveness of exploration and practical computational constraints.

Selecting the Error Metrics

The primary accuracy metric employed in this study is the Mean Absolute Error (MAE), as defined in Equation (4.2). MAE was chosen for its simplicity, computational efficiency, and widespread use as an initial evaluation metric in AxC literature [68, 77, 122, 84]. However, it is essential to acknowledge that MAE might not comprehensively represent the quality implications for all real-world end applications, particularly where subjective or perceptual qualities significantly influence user experience (e.g., multimedia applications). Therefore, while MAE provides an efficient and standardized metric for preliminary evaluations, future studies should incorporate additional, domain-specific accuracy measures to enrich the assessment of approximation impacts.

Overall, this experimental setup is meticulously structured, offering a foundation for systematically assessing and validating the performance and effectiveness of the proposed RL-based methodology in the domain of AxC.

4.5 Experimental Results and Analysis

The experiments were conducted using the proposed RL-based DSE framework, targeting the selected benchmarks: Matrix Multiplication (sizes 10×10 and 50×50) and the FIR filter (with 100 and 200 samples). The exploration process required 2000 and 4000 steps to achieve stable solutions for Matrix Multiplications of sizes 10×10 and 50×50, respectively. In addition, the exploration process required 500 and 1240 steps for the FIR filters with 100 and 200 samples, respectively. These results reflect varied complexities across benchmarks [6].

4.5.1 Exploration Outcomes

The complete exploration outcomes for each benchmark, including power consumption, computation time, accuracy degradation (quantified by Mean Absolute Error (MAE)), and selected approximate operators, are detailed in Table 4.3. The reported numerical values represent the difference (Δ) between the parameter obtained from the precise baseline and the one obtained after approximation at the final exploration step. Specifically:

- **Min**: The minimum observed Δ value during exploration.
- **Solution**: The final configuration selected by the RL agent as the optimal trade-off.
- **Max**: The maximum observed Δ value during exploration, indicating the extreme approximation scenario explored.

The accuracy degradation values indicate the extent to which computational accuracy is compromised using MAE (see Equation (4.2)). It is crucial to note that while the MAE provides a standardized accuracy measure, its implications for real-world applications must be carefully evaluated based on the specific application domain and error-tolerance criteria.

4.5.2 Evolution of Performance Metrics

Figure 4.2 and Figure 4.3 illustrate the evolution of performance metrics (power, computation time, and accuracy) over the exploration steps for Matrix Multiplication (10×10) and FIR filter (100 samples), respectively. Each data point corresponds to a particular configuration tested by the agent. An apparent convergence trend towards solutions that effectively balance accuracy against performance gains can be observed in Matrix Multiplication. Conversely, FIR exploration exhibits more fluctuation, indicating challenges related to reward shaping or hyperparameter tuning, which highlights the need for further method refinements in complex scenarios.

4.5.3 Reward Evolution Analysis

The evolution of the reward throughout the exploration provides insights into the learning efficiency of the agent. As depicted in Figure 4.4, the reward evolution for the Matrix Multiplication benchmark demonstrates consistent reward improvements, indicating successful learning behavior. Conversely, the reward evolution for the FIR filter reveals irregularities in the reward progression, indicating sensitivity to the agent's hyperparameters and reward shaping strategies, and thus warrants further optimization efforts.

Table 4.3: Exploration results for power, time, and accuracy across benchmarks [6].

Benchmarks	**Matrix Mult.**		**FIR**	
	10×10	*50×50*	*100-sample*	*200-sample*
Δ Power (mW)				
min	15	0.55	529.515	1059.345
solution	415.3	753.72	10850.855	1237.247
max	418.4	1552.017	17344.390	34699.1
Δ Time (ns)				
min	50	-90	563.135	1126.605
solution	1780	1460.8	2664.385	3951.525
max	1840	5707.6	6547.495	13098.89
Accuracy (MAE)				
min	0.02	0	1096.03	395.74
solution	19.95	0.736	1096.03	27580.345
max	204.71	26.7964	31671.43	27580.35
Configuration				
Adder	00M	6R6	0GN	067
Multiplier	17MJ	L93	043	018

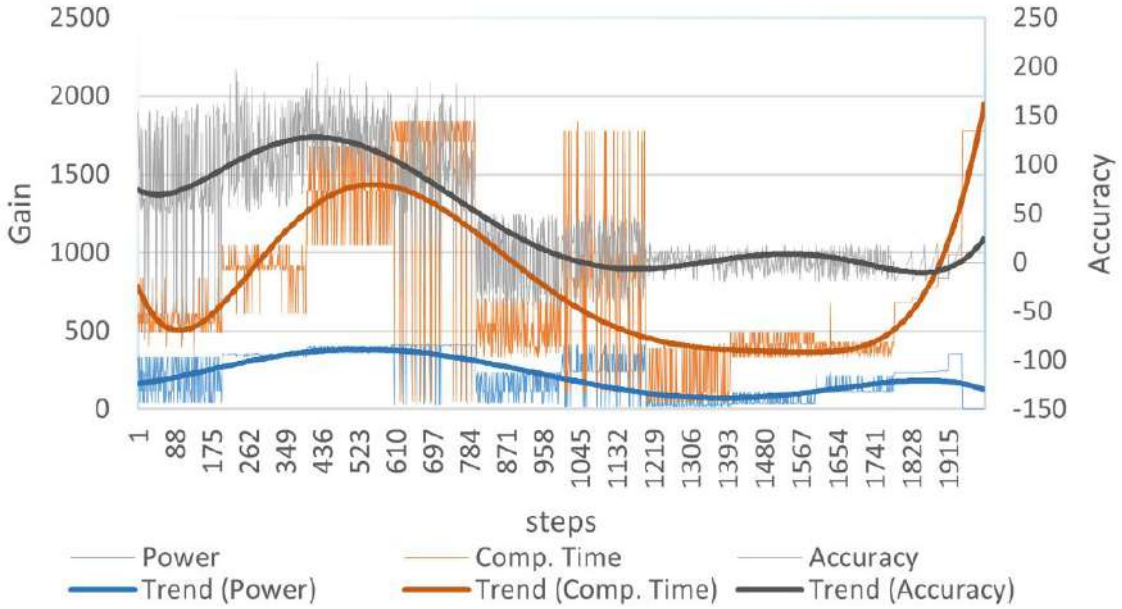

Figure 4.2: Exploration outcomes over time for Matrix Multiplication (10×10) [6].

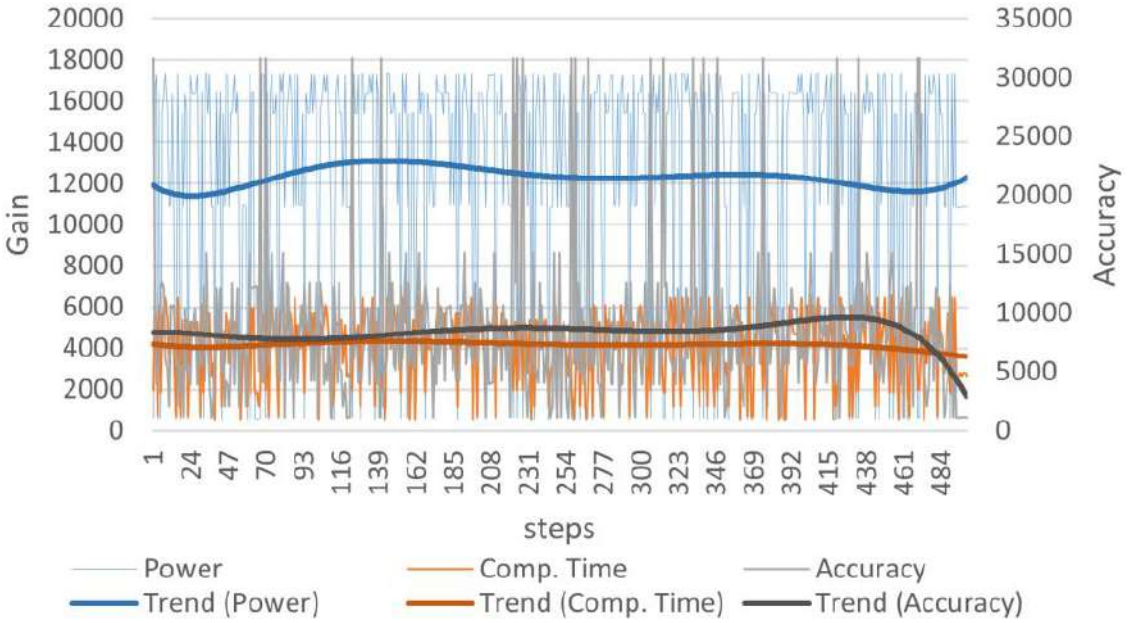

Figure 4.3: Exploration outcomes over time for FIR (100 samples) [6].

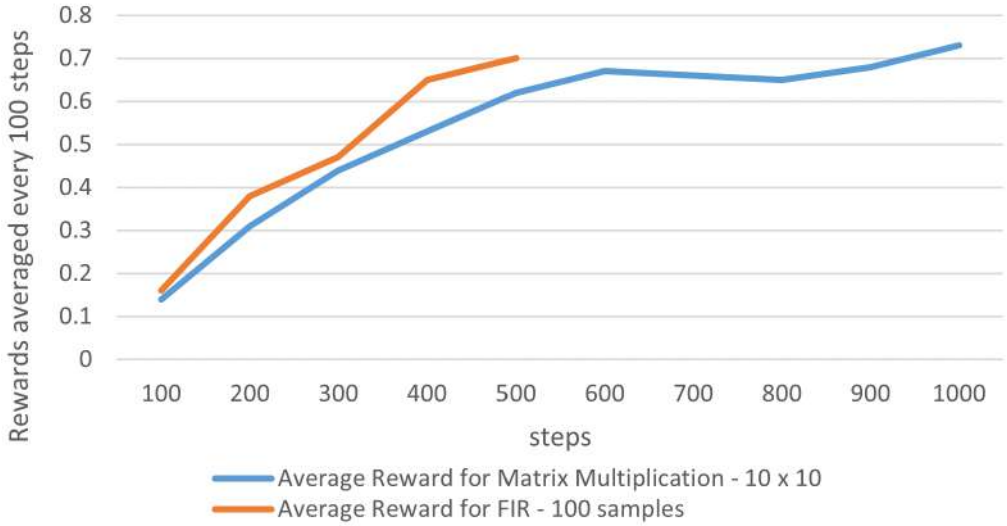

Figure 4.4: Average reward evolution for Matrix Multiplication (10×10) and FIR (100 samples) [6].

4.5.4 Design Space Size and Complexity Considerations

Regarding the comparison of the complexity of exhaustive search against the proposed method, it should be noted that despite selecting a limited number of adders and multipliers per exploration step (five approximate operators for each in

this case), the total size of the design space becomes considerably large when considering combinatorial interactions among arithmetic operators, variable approximations, and multiple precision levels. An exhaustive approach quickly becomes computationally prohibitive, typically exceeding millions of evaluations even for moderate problem sizes. The selected RL-based DSE attempts to reduce complexity and efficiently navigate the design space, and struggles to achieve practical convergence towards near-optimal configurations within feasible computational budgets.

Exhaustive per-site design space (independent of the RL policy).

Let A and M denote the numbers of additions and multiplications exercised by the benchmark, respectively, and let S_{add} and S_{mul} be the numbers of available implementations per operator type (precise plus approximate variants in Tables 4.1 and 4.2). The total number of distinct configurations an exhaustive search must enumerate is:

$$|\boldsymbol{\mathcal{D}}_{\text{exhaustive}}| = (\boldsymbol{S_{\text{add}}})^{\boldsymbol{A}} \times (\boldsymbol{S_{\text{mul}}})^{\boldsymbol{M}} \tag{4.7}$$

In this work, the setup is $S_{\text{add}} = S_{\text{mul}} = 6$, hence $|\mathcal{D}_{\text{exhaustive}}| = 6^{(A+M)}$. For the naive $n \times n$ matrix multiplication kernel one has $M = n^3$ multiplications and $A = n^2(n-1)$ additions; e.g., at $n = 50$ this yields $|\mathcal{D}_{\text{exhaustive}}| = 6^{125000+122500} = 6^{247500}$. For an L-tap FIR filter applied to N input samples, where L is the filter length (number of taps) and N is the number of input samples, $M = L \times N$ and $A = L \times N$, leading to $|\mathcal{D}_{\text{exhaustive}}| = 6^{(2L)\times N}$. For example, for $L = 32$ and $N = 100$, $|\mathcal{D}_{\text{exhaustive}}| = 6^{64\times 100} = 6^{6400}$. These counts capture only per-site operator choices; incorporating additional dimensions (e.g., variable-level masks or precision schemes) would further enlarge the search space.

In conclusion, the presented results suggest the effectiveness of the proposed RL-based DSE framework in navigating trade-offs among accuracy, power consumption, and computation time, specifically for the Matrix Multiplication benchmarks. Considering the FIR benchmarks highlights areas that require further method refinement and exploration for enhanced generalization and robustness.

4.5.5 Comparisons with Existing Methods

Several approaches exist for performing DSE on approximate designs, ranging from heuristic and EAs to modern ML-based solutions. Traditional heuristics, such as GAs or simulated annealing [44, 123, 70, 36, 71, 72], systematically evaluate sets of approximate operators while often requiring extensive computation time and parameter tuning. Recently, ML-based DSE approaches, including RL, have shown promising efficiency for performing DSE with the aim of resource allocation [96] and speeding up High-Level Synthesis (HLS) exploration [97].

Among ML-based DSE methods, RL is prominently featured in several notable studies [45, 67, 46], although alternative learning strategies such as MBO [68], modified MCTS [69], and general heuristic-based ML frameworks [38, 32] have also been explored. The effectiveness of these methods depends strongly on their target hardware platforms, optimization objectives, and the complexity of the design space.

Compared to the method proposed by Hashemi et al. [45], which applies RL to optimize parameters for iris scanning on FPGAs, the proposed approach targets general-purpose CPU scenarios, employing operator-level approximation. While their framework focuses on parameter-level tuning, this study systematically explores heterogeneous arithmetic configurations, enabling broader applicability across various computational kernels.

The approach by Hoffmann et al. [67] employs RL to dynamically adjust software-level parameters, such as loop perforation and data structure approximations, for heterogeneous platforms. In contrast, the proposed framework operates directly on hardware-level arithmetic operators, providing finer control for approximation and directly influencing power and timing characteristics of the executed kernels.

Similarly, Elthakeb et al. [46] proposed an RL-based quantization framework for DNN inference. Their solution is domain-specific, focusing on DNN operations, and primarily targets quantization levels affecting accuracy-energy trade-offs. By contrast, the proposed methodology supports a different category of computation-intensive workloads, including signal processing and matrix operations, through general-purpose arithmetic operation approximation.

Recent heuristic-based approaches, such as AutoAx [38], LDAX [32], and modified MCTS techniques [69], mainly target accelerator design and emphasize area or energy efficiency. These studies, explore approximation techniques using approximate addition and multiplication operations similar to this study. However, these studies often neglect explicit timing constraints or multi-objective trade-offs for more than two objectives. The proposed RL-based exploration methodology in this study explicitly integrates accuracy, power, and execution time objectives into a single reward loop, allowing the agent to discover configurations that best satisfy multi-objective requirements.

The complexity of the design space is another distinguishing factor. Heuristic or EAs typically require a large number of evaluations to cover all of the combinations of operator, variable, and precision-level choices. The proposed method leverages the inherent learning capability of RL to reduce the number of evaluations needed while maintaining near-optimal trade-offs. Nevertheless, as observed in the FIR filter benchmark results, the reward fluctuations suggest that further refinement of the reward-shaping strategy and hyperparameter tuning would enhance robustness for more complex exploration scenarios.

The selection of RL as the primary search strategy, rather than EAs such as

GAs, is motivated by the cost structure and scalability limits of search-based DSE in combinatorial, black-box approximation spaces. As discussed in Chapter 2, Section 2.2.2, EA-driven exploration typically requires a large number of full design evaluations and careful hyperparameter tuning to achieve competitive solutions, which becomes prohibitive as the number of approximation decision sites and operator choices increases. In contrast, RL explicitly learns a policy that guides subsequent sampling toward promising regions of the design space, thereby reducing repeated evaluations while maintaining the ability to operate under discrete actions, non-differentiable objectives, and strict accuracy constraints.

In summary, the proposed RL-driven DSE methodology is consistent with recent learning-based exploration solutions while enabling the exploration of approximated designs and multi-objective optimization. Future research should focus on refining the reward formulation, expanding benchmark diversity, and improving parameter adaptation to increase the likelihood of generalizing the method across different computational domains and to improve convergence stability in larger, more heterogeneous design spaces.

4.6 Discussion and Implications

The results presented in this chapter highlight the strengths and limitations of employing a RL-based methodology for multi-objective DSE in AxC contexts. The exploration outcomes across different benchmarks demonstrate that the proposed framework can identify configurations that effectively reduce power consumption and computation time, while controlling accuracy degradation within predefined thresholds. Specifically, for matrix multiplication benchmarks, the agent exhibited a consistent learning behavior and was able to converge towards high-quality solutions. However, in more complex or less predictable benchmarks such as the FIR filter, the exploration process revealed irregular reward patterns, suggesting that further refinements in reward shaping or agent configuration are necessary to improve convergence and robustness.

A critical factor influencing the success of the exploration is the setting of design constraints, especially the thresholds acc_{th}, p_{th}, and t_{th} that are encoded into the reward function. These thresholds are not universal; they were empirically determined based on preliminary evaluations of the studied benchmarks and tailored to the expected trade-offs between accuracy and performance. For instance, the selected accuracy threshold (acc_{th}), although fixed for experimental consistency, was established through iterative trials to ensure that the final output remained within an acceptable error range for the chosen benchmarks. As discussed in Section 4.4.4, this threshold may not be appropriate for other applications, and thus, the framework is designed to allow user-specified constraints for each exploration run. This flexibility ensures that the DSE process is adaptable to application-specific

requirements and error tolerances.

The heterogeneous behavior observed across benchmarks also emphasizes the significance of benchmark characteristics in shaping the agent's learning trajectory. Benchmarks with structured data and predictable error propagation, such as matrix multiplication, provide a more favorable environment for the RL agent to exploit the reward signals effectively. In contrast, benchmarks with more complex or noisy behavior may require refined agent architectures, advanced reward normalization strategies, or integration of prior domain knowledge to ensure stable exploration.

The complexity of the design space also plays a central role in determining the practical utility of the proposed approach. As discussed in Section 4.5, even with a constrained selection of arithmetic operators, the number of possible configurations grows rapidly due to combinatorial interactions among operator choices, input precisions, and error tolerances. Performing an exhaustive search in such spaces becomes computationally infeasible, particularly for real-world applications. The RL approach, despite its sensitivity to hyperparameter settings, offers a viable alternative by guiding the exploration towards promising regions of the design space without evaluating every possible configuration.

Looking forward, several avenues exist for enhancing the proposed framework. Future work may focus on extending the reward function to incorporate more diverse accuracy metrics or domain-specific quality indicators, enabling broader applicability to domains such as image processing, signal analysis, or ML inference. Additionally, incorporating domain-specific insights, such as error resilience patterns or sensitivity analysis of variables, could reduce search overhead and improve convergence behavior. Another promising direction is to scale the framework to support larger programs and a broader range of approximation opportunities, including approximate memory accesses, control flow simplifications, or dynamic bitwidth tuning. In general, future work will expand the evaluation to a broader set of applications to further confirm the scalability and general applicability of the proposed RL-based exploration framework.

In summary, the findings presented in this chapter indicate that RL constitutes a promising tool for navigating the multi-objective trade-offs inherent in AxC applications. While the framework demonstrates encouraging results for selected benchmarks, further research is necessary to generalize its applicability, improve its scalability, and increase its robustness to variations in application behavior and design constraints.

Chapter 5

Conclusion and Future Work

This thesis presented methodologies for performing DSE to apply AxC techniques in computationally intensive designs, with a focus on accuracy, power consumption, and computational time trade-offs. Two distinct approaches were explored: an IA-based DSE approach for exploring the effect of approximation on an SNN and a RL-based DSE approach for exploring approximation opportunities in general computation benchmarks such as Matrix multiplication and FIR filters.

The IA-based methodology introduced in Chapter 3 provides a framework for analysis and exploration, at least faster than brute-force evaluation in the evaluated setups, to estimate the impact of precision reduction on an SNN accuracy. A baseline IA model identified optimal truncation thresholds for the network parameters while maintaining user-defined accuracy degradation limits, striving to reduce the complexity of brute-force approaches. Further improvements introduced watchers, enabling fine-grained, neuron-level precision tuning. This enhanced IA-based model revealed opportunities to further reduce precision in select neurons without compromising network classification accuracy. Experimental evaluations showed consistent exploration behavior in the studied cases, suggesting that selective precision adjustments could achieve considerable memory compression and computational efficiency.

Chapter 4 described an RL-based exploration framework designed to automate the multi-objective selection of approximate arithmetic operations in software applications. Using an RL agent, the approach traded off the computation accuracy with power consumption and execution time. Benchmarks such as matrix multiplication and FIR filter provided an initial validation of the feasibility of using the proposed DSE methodology to find acceptable improvements in computational efficiency and energy consumption within acceptable accuracy constraints. This RL-based DSE approach provides a framework for systematically exploring large design spaces, reducing manual tuning efforts in the evaluated cases.

Both methodologies share a common goal: efficiently navigating the design spaces comprised of approximated designs to identify the most suitable designs that

may maximize the benefits over optimization goals, for example, reducing power consumption and the execution time simultaneously while maintaining accuracy degradation within predefined acceptable thresholds. The two methodologies may be complementary, and future work could investigate integration of such methods, combining IA capability in local error estimation with RL global decision-making capabilities. Such integrated approaches could further refine accuracy-aware DSE, opening new avenues for optimizing complex AxC applications.

Future work should explore the scalability of the proposed methodologies to larger and more diverse application domains. For the IA-based approach, extending the error-propagation model to deeper or other types of NN architectures may yield practical benefits. For the RL-based methodology, enhancing reward functions, integrating more sophisticated learning algorithms, or combining RL with domain-specific heuristics could lead to more efficient exploration in larger software systems.

Overall, the contributions presented in this thesis strive to advance the state-of-the-art in DSE methodologies for AxC applications, providing foundations for further research and practical applications in energy-efficient and performance-critical computing systems.

List of Publications

The following publications have resulted from the research work presented in this thesis.

Journal and Conference Papers

1. Saeedi, S., Piri, A., Deveautour, B., O'connor, I., Bosio, A., Savino, A. and Di Carlo, S., 2024. A Survey on Design Space Exploration Approaches for Approximate Computing Systems. Electronics, 13(22), p.4442. [**Chapter 2**]

2. Saeedi, S., Carpegna, A., Savino, A. and Di Carlo, S., 2022, September. Prediction of the impact of approximate computing on spiking neural networks via interval arithmetic. In 2022 IEEE 23rd Latin American Test Symposium (LATS) (pp. 1-6). IEEE. [**Chapter 3**]

3. Saeedi, S., Carpegna, A., Savino, A. and Di Carlo, S., 2024, October. Fast Exploration of the Impact of Precision Reduction on Spiking Neural Networks. In 2024 IEEE International Conference on Design, Test and Technology of Integrated Systems (DTTIS) (pp. 1-6). IEEE. [**Chapter 3**]

4. Saeedi, S., Savino, A. and Di Carlo, S., 2023, June. Design space exploration of approximate computing techniques with a reinforcement learning approach. In 2023 53rd Annual IEEE/IFIP International Conference on Dependable Systems and Networks Workshops (DSN-W) (pp. 167-170). IEEE. [**Chapter 4**]

Other Publications

The following publications are not directly included in the thesis chapters but reflect other research contributions by the author.

1. Piri, A., Saeedi, S., Barbareschi, M., Deveautour, B., Di Carlo, S., O'Connor, I., Savino, A., Traiola, M. and Bosio, A., 2022, May. Input-aware approximate

computing. In 2022 IEEE International Conference on Automation, Quality and Testing, Robotics (AQTR) (pp. 1-6). IEEE.

2. Traiola, M., Pappalardo, S., Piri, A., Ruospo, A., Deveautour, B., Sanchez, E., Bosio, A., Saeedi, S., Carpegna, A., Göğebakan, A.B. and Magliano, E., 2024, May. Approximate Fault-Tolerant Neural Network Systems. In 2024 IEEE European Test Symposium (ETS) (pp. 1-10). IEEE.

Appendix A

Comprehensive tables for Related Work

This appendix consolidates all detailed comparative tables presented to categorize the related work as referenced in Chapter 2. The tables retain their original numbering and content for completeness.

Table A.1: Research works using ML-based search algorithms to perform the DSE [2].

Year	Ref.	Target Hardware	Use Case Domains	Benchmarks	Search Algorithm
2018	[45]	FPGA	Image processing	Iris scanning	RL
2021	[68]	FPGA	Image processing	Kernel-based Gaussian blur filter	MBO
2019	[38]	Accelerator (ASIC)	Image processing	Sobel and Gaussian blur filters (one with fixed coefficients, and one with a 3×3 kernel)	ML-based heuristic
2023	[69]	Accelerator (ASIC)	Signal and image processing and general arithmetic	Adder Tree, RGB2gray, FIR, and Gaussian blur filters	Artificial Intelligence (AI)-based heuristic: modified MCTS

Continued on next page

Table A.1 (Continued)

Year	Ref.	Target Hardware	Use Case Domains	Benchmarks	Search Algorithm
2021	[32]	FPGA, ASIC, general-purpose computing system	Image processing	RGB2GRAY, Ternary sum, FIR, image sharpening, and Gaussian blur filters	ML
2015	[67]	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	[46]	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	[6]	General-purpose CPU	ML, digital signal processing, and image processing	Matrix multiplication and FIR filter	RL

Table A.2: Research works using EAs as a search algorithm to perform the DSE [2].

Year	Ref.	Target Hardware	Use Case Domains	Benchmarks	Search Algorithm
2021	[44, 123]	FPGA	Image Processing	Pixel-streaming pipeline	GA

Continued on next page

Table A.2 (Continued)

Year	Ref.	Target Hardware	Use Case Domains	Benchmarks	Search Algorithm
2023	[70]	FPGA-based approximate accelerator	High-Efficiency Video Coding (HEVC)	Multiplierless Multiple Constant Multiplier (MCM)	ES algorithm and NSGA-II
2022	[36]	Accelerator (FPGA and ASIC)	Image processing (JPEG compression)	Discrete Cosine Transform (DCT)	NSGA-II
2014	[71]	General-purpose CPU (implied)	Scientific computing, 3D gaming, 3D image rendering, signal, and image processing	Fast Fourier Transform (FFT), SOR, MC, SMM, LU, Zxing, JMEint, Imagefill, and raytracer	GA
2023	[72]	GPU	Image classification using CNNs	MobileNetV2 and ResNet50V2	NAS algorithms: EvoApproxNAS (based on NSGA-II)

Table A.3: Research works using custom search algorithms to perform the DSE [2].

Year	Ref.	Target Hardware	Use Case Domains	Benchmarks	Search Algorithm
2016	[43]	ASIC, and FPGA	HEVC	Sum of Absolute Differences (SAD)	Custom
2020	[33]	ASIC, and FPGA	Image classification using DNNs	ResNet-18/34/38/74, MobileNetV2, and Transformer-base/WikiText-103	Custom

Continued on next page

Table A.3 (Continued)

Year	Ref.	Target Hardware	Use Case Domains	Benchmarks	Search Algorithm
2016	[79]	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	[77]	ASIC (implied)	HEVC	SAD	Custom
2021	[80]	ASIC (implied)	Image processing	Sobel, FIR, and Gaussian blur filters, a ReLu Neuron, and Euclidean distance	Custom
2017	[122]	VLSI systems and HLS, aligning with ASIC design	Image processing	Average number calculator, inverse DCT calculator, Sobel, FIR, interpolation, and decimation filters	Custom
2020	[73]	HLS tools for accelerator design	Image processing	Sobel and Sharpen filters	TS with potential integration of GAs
2018	[78]	Hardware accelerator (ASIC implied)	ML, digital signal processing, and image processing	Matrix multiplication, Sobel filter, and DCT	Custom

Continued on next page

Table A.3 (Continued)

Year	Ref.	Target Hardware	Use Case Domains	Benchmarks	Search Algorithm
2016	[74]	ML accelerator (ASIC)	Image processing and Natural Language Processing (NLP)/text classification	Eye and face detection, optical digit, digit, webpage, and text classification	GD
2021	[124, 125]	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	[27]	NPU (ASIC)	Financial analysis, robotics, 3D gaming, image compression, signal, and image processing	Blackscholes, FFT, Inversek2j, Jmeint, JPEG, Sobel filter	Custom
2019	[75]	Not specified (implied general-purpose)	Image processing	Matrix multiplication and FIR filter	Custom

Continued on next page

Table A.3 (Continued)

Year	Ref.	Target Hardware	Use Case Domains	Benchmarks	Search Algorithm
2016	[76]	GPU	ML, signal processing (pattern recognition), image processing, medical imaging, scientific computing, and web mining	Backprop, Fastwalsh, Gaussian, Heartwall, Matrixmul, Particle filter, Similarity score, S.reduce, S.srad2, and String match (GPU). Bwaves, CactusADM, FMA3D, GemsFDTD, Soplex, and Swim (CPU)	Custom

Table A.4: Research works that perform DSE for approximate functions design space instead of a complete system [2].

Year	Ref.	Target Hardware	Use Case Domains	Benchmarks	Search Algorithm
2021	[81]	FPGA (also ASIC implied)	NA	apex2, b12, clip, duke2, and vg2 benchmarks from IWLS'93 Benchmark Set	NSGA-II
2014	[83]	FPGA and ASIC (implied)	NA	Ripple Carry, Carry Lookahead, and Kogge Stone Adders. Wallace, and Dadda Multipliers	Custom

Continued on next page

Table A.4 (Continued)

Year	Ref.	Target Hardware	Use Case Domains	Benchmarks	Search Algorithm
2021	[82]	FPGA and ASIC (designing fault-tolerant architectures)	Safety-critical applications: Quadruple Approximate Modular Redundancy (QAMR)	Generic combinational circuits	NSGA-II
2021	[84]	FPGA and ASIC (implied)	NA	Approximate adders, multipliers, divisor, barrel shifter, Sine, and Square	Heuristic
2022	[85]	FPGA	NA	Approximate adders, multipliers, decoders, and ALUs	NSGA-II

Table A.5: AxC techniques in research works using ML-based search algorithms to perform the DSE [2].

Year	Ref.	Target Hardware	Benchmarks	Software Techniques	Architectural Techniques	Hardware Techniques
2018	[45]	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

Continued on next page

Table A.5 (Continued)

Year	Ref.	Target Hard-ware	Benchmarks	Software Techniques	Architectural Techniques	Hardware Techniques
2021	[68]	FPGA	Kernel-based Gaussian blur filter	Approximation of the window size, mode, and stride length for convolution kernels	NA	Approximate multipliers and adders
2019	[38]	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	[69]	Accelerator (ASIC)	Adder Tree, RGB2gray, FIR, and Gaussian blur filters	NA	NA	Approximate adders and multipliers
2021	[32]	FPGA, ASIC, general-purpose computing system	RGB2GRAY, Ternary sum. FIR, image sharpening, and Gaussian blur filters	NA	NA	Approximate adders and multipliers

Continued on next page

Table A.5 (Continued)

Year	Ref.	Target Hardware	Benchmarks	Software Techniques	Architectural Techniques	Hardware Techniques
2015	[67]	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	[46]	General-purpose CPU and DNN accelerator	AlexNet, SimpleNet, LeNet, MobileNet, ResNet-20, 10-Layers, and VGG-11/16	DNN layer Quantization	NA	NA
2023	[6]	General-purpose CPU	Matrix multiplication and FIR filter	NA	NA	Approximate adders and multipliers

Table A.6: AxC techniques in research works using EAs as a search algorithm to perform the DSE [2].

Year	Ref.	Target Hardware	Benchmarks	Software Techniques	Architectural Techniques	Hardware Techniques
2021	[44, 123]	FPGA	Pixel-streaming pipeline	NA	NA	Sparse LUTs, precision scaling, approximate adders
2023	[70]	FPGA-based approximate accelerator	Multiplierless MCM	NA	NA	Approximate adders and multipliers
2022	[36]	Accelerator (FPGA and ASIC)	DCT	NA	NA	Approximate adders
2014	[71]	General-purpose CPU (implied)	FFT, SOR, MC, SMM, LU, Zxing, JMEint, Imagefill, and raytracer	Program static instructions	NA	NA

Continued on next page

Table A.6 (Continued)

Year	Ref.	Target Hardware	Benchmarks	Software Techniques	Architectural Techniques	Hardware Techniques
2023	[72]	GPU	MobileNetV2, and ResNet50V2	Approximate 8xN bit multipliers emulated using LUTs, approximate depthwise convolution, and quantization-aware training	NA	NA

Table A.7: AxC techniques in research works using custom search algorithms to perform the DSE [2].

Year	Ref.	Target Hardware	Benchmarks	Software Techniques	Architectural Techniques	Hardware Techniques
2016	[43]	ASIC and FPGA	SAD	NA	NA	Approximate adders and logic blocks
2020	[33]	ASIC and FPGA	ResNet-18/34/38/74, MobileNetV2, and Transformer-base/WikiText-103	Progressive Fractional Quantization (PFQ), Dynamic Fractional Quantization (DFQ)	NA	NA

Continued on next page

Table A.7 (Continued)

Year	Ref.	Target Hard-ware	Benchmarks	Software Techniques	Architectural Techniques	Hardware Techniques
2016	[79]	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	[77]	ASIC (implied)	SAD	NA	NA	Approximate adders
2021	[80]	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

Continued on next page

Table A.7 (Continued)

Year	Ref.	Target Hardware	Benchmarks	Software Techniques	Architectural Techniques	Hardware Techniques
2017	[122]	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	[73]	HLS tools for accelerator design	Sobel and Sharpen filters	NA	NA	Approximate adders and multipliers
2018	[78]	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	[74]	ML accelerator (ASIC)	Eye and face detection, optical digit, digit, webpage, and text classification	NA	NA	Clock overgating

Continued on next page

Table A.7 (Continued)

Year	Ref.	Target Hardware	Benchmarks	Software Techniques	Architectural Techniques	Hardware Techniques
2021	[124, 125]	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	[27]	NPU (ASIC)	Blackscholes, FFT, Inversek2j, Jmeint, JPEG, Sobel filter	NA	Use of NNs (NPU accelerator)	NA
2019	[75]	Not specified (implied general-purpose)	Matrix multiplication and FIR filter	NA	NA	Approximate adders and multipliers

Continued on next page

Table A.7 (Continued)

Year	Ref.	Target Hard-ware	Benchmarks	Software Techniques	Architectural Techniques	Hardware Techniques
2016	[76]	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	Rollback-Free Value Prediction (RFVP)

Table A.8: AxC techniques in research works that perform the DSE for approximate functions design space instead of a complete system [2].

Year	Ref.	Target Hardware	Benchmarks	Software Techniques	Architectural Techniques	Hardware Techniques
2021	[81]	FPGA (also ASIC implied)	apex2, b12, clip, duke2, and vg2 benchmarks from IWLS'93 Benchmark Set	NA	NA	Logic falsification
2014	[83]	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	[82]	FPGA and ASIC (designing fault-tolerant architectures)	Generic combinational circuits	NA	NA	Logic falsification
2021	[84]	FPGA and ASIC (implied)	Approximate adders, multipliers, divisor, barrel shifter, Sine, and Square	NA	NA	Approximation based on BMF for truth tables

Continued on next page

Table A.8 (Continued)

Year	Ref.	Target Hardware	Benchmarks	Software Techniques	Architectural Techniques	Hardware Techniques
2022	[85]	FPGA	Approximate adders, multipliers, decoders, and ALUs	NA	NA	Customized approximation of Boolean networks

Table A.9: Evaluated metrics in research works using ML-based search algorithms to perform the DSE [2].

Year	Ref.	Accu -racy	Error Metric(s)	Power or Energy	Time	Perfor - mance	Mem -ory	Area	Pareto Front
2018	[45]	Industry-level threshold for iris encoding	HD of any two images	Energy	Exec. time	NA	Mem. utilization	Logic utilization	Speedup vs. Avg. Error
2021	[68]	Application-level error	MAE	NA	NA	NA	NA	LUT count	LUT count vs. error
2019	[38]	Quality of Resultss (QORs)	SSIM	Energy	NA	NA	NA	Area	SSIM vs. area or energy
2023	[69]	Output accuracy	MRED, PSNR	NA	NA	NA	NA	Area	Area savings vs. error
2021	[32]	Output accuracy	MRED, PSNR	Power	NA	NA	NA	Area	NA

Continued on next page

Table A.9 (Continued)

Year	Ref.	Accu -racy	Error Metric(s)	Power or En- ergy	Time	Perfor - mance	Mem -ory	Area	Pareto Front
2015	[67]	Application accu- racy	Precision/Recall, PSNR, Swaption price, track quality, wire length, clustering quality	Energy effi- ciency	NA	Speedup	NA	NA	Energy vs. accuracy
2020	[46]	Relative DNN accu- racy	DNN quantiza- tion error	Energy	NA	Speedup	NA	NA	Relative accuracy vs. quanti- zation level
2023	[6]	Output accu- racy	MAE	Power	Computation time	NA	NA	NA	NA

Table A.10: Evaluated metrics in research works using EAs as a search algorithm to perform the DSE [2].

Year	Ref.	Accu -racy	Error Metric(s)	Power or En- ergy	Time	Perfor - mance	Mem -ory	Area	Pareto Front
2021	[44, 123]	Output image quality	ΔE in CIELAB color space	Power	NA	NA	NA	NA	Power vs. ΔE

Continued on next page

Table A.10 (Continued)

Year	Ref.	Accu -racy	Error Metric(s)	Power or En- ergy	Time	Perfor - mance	Mem -ory	Area	Pareto Front
2023	[70]	Output quality	PSNR	Power	NA	NA	NA	LUT count	Power vs. PSNR, LUT vs. PSNR
2022	[36]	Output image quality	Mean Structural SIMilarity (MSSIM) and Structural DISSIMi- larity (DSSIM)	Power	NA	NA	NA	Area	Area vs. DSSIM (for ASIC). LUT count vs. DSSIM (for FPGA). Power vs. DSSIM (for both).
2014	[71]	QORs	ER, average entry diff, MPD, normalized difference	Energy	NA	NA	NA	NA	NA
2023	[72]	CNN Top-1 Accu- racy	Classification accuracy (%)	Energy of mul- tipli- ca- tions	CNN train- ing time	NA	NA	NA	Energy vs. CNN accuracy

Table A.11: Evaluated metrics in research works using custom search algorithms to perform the DSE [2].

Year	Ref.	Accu -racy	Error Metric(s)	Power or En- ergy	Time	Perfor - mance	Mem -ory	Area	Pareto Front
2016	[43]	Output quality	BER (%)	Power	NA	Bit Rate	NA	Area	NA
2020	[33]	DNN accu- racy	Classification accuracy	Train -ing en- ergy	Train -ing la- tency	NA	NA	NA	NA
2016	[79]	DNN classifi- cation	Classification accuracy	Energy sav- ings	NA	NA	NA	NA	NA
2019	[77]	Output accura- cy/cod- ing effi- ciency	MAE, BD-PSNR	Power dissi- pa- tion	NA	NA	NA	Circuit Area	MAE vs. power and Circuit Area. Power and Area vs. BD- PSNR.
2021	[80]	Output accu- racy	MRED	Energy re- duc- tion	NA	Perfor -mance	NA	Area over- head	Power vs. output accuracy
2017	[122]	Comput -ation accu- racy	MAE	Power	NA	Perfor -mance	NA	Area	Area vs. MAE

Continued on next page

Table A.11 (Continued)

Year	Ref.	Accu -racy	Error Metric(s)	Power or En- ergy	Time	Perfor - mance	Mem -ory	Area	Pareto Front
2020	[73]	Output accu- racy	MED, PSNR	Power-Delay-Product (PDP)	NA	NA	NA	Area	Area and PDP vs. MED.
2018	[78]	Output accu- racy	MRED	Power	NA	NA	NA	NA	Power vs. error
2016	[74]	Output quality	Classific -ation accuracy	Energy	NA	NA	NA	NA	NA
2021	[124, 125]	DNN accu- racy	Classific -ation accuracy (%)	NA	Inference la- tency	Compute effi- ciency	NA	NA	NA
2016	[27]	Output quality	MRED, miss rate, image difference	Energy re- duc- tion	NA	Speedup	NA	NA	NA
2019	[75]	Output accu- racy	Normalized weighted error	NA	NA	NA	NA	NA	NA
2016	[76]	Output quality	MRED, Euclidean distance, mismatch rate, RMSE	Energy	Exec. time	Speedup	Mem. band- width	NA	Product of energy, execu- tion time, and error vs. predic- tor size

Table A.12: Evaluated metrics in research works that perform the DSE for approximate functions design space instead of a complete system [2].

Year	**Ref.**	**Accu -racy**	**Error Metric(s)**	**Power or Energy**	**Time**	**Perfor - mance**	**Mem -ory**	**Area**	**Pareto Front**
2021	[81]	Circuit output accuracy	BER	NA	NA	NA	NA	NA	Resources (Cubes and Literals) vs. BER
2014	[83]	Circuit output accuracy	Error magnitude/frequency	NA	NA	NA	NA	NA	Normalized Gate count vs. error frequency
2021	[82]	NA	NA	NA	Critical path delay	NA	NA	Area	Delay gain vs. area gain
2021	[84]	Circuit output accuracy	Normalized HD and MAE	Power	NA	NA	NA	Area	Power and Area utilization vs. MAE
2022	[85]	Accuracy at primary outputs	ER	NA	NA	NA	NA	LUT count	ER vs. LUT count

Bibliography

[1] Vasileios Leon, Muhammad Abdullah Hanif, Giorgos Armeniakos, Xun Jiao, Muhammad Shafique, Kiamal Pekmestzi, and Dimitrios Soudris. Approximate computing survey, part ii: Application-specific & architectural approximation techniques and applications. *arXiv preprint arXiv:2307.11128*, 2023.

[2] Sepide Saeedi, Ali Piri, Bastien Deveautour, Ian O'connor, Alberto Bosio, Alessandro Savino, and Stefano Di Carlo. A survey on design space exploration approaches for approximate computing systems. *Electronics*, 13(22):4442, 2024.

[3] Sepide Saeedi, Alessio Carpegna, Alessandro Savino, and Stefano Di Carlo. Prediction of the impact of approximate computing on spiking neural networks via interval arithmetic. In *2022 IEEE Latin-American Test Symposium (LATS)*, 2022.

[4] Vojtech Mrazek, Radek Hrbacek, Zdenek Vasicek, and Lukas Sekanina. Evoapprox8b: Library of approximate adders and multipliers for circuit design and benchmarking of approximation methods. In *Design, Automation & Test in Europe Conference & Exhibition (DATE), 2017*, pages 258–261, 2017.

[5] Milan Češka, Jiří Matyaš, Vojtech Mrazek, Lukas Sekanina, Zdenek Vasicek, and Tomas Vojnar. Approximating complex arithmetic circuits with formal error guarantees: 32-bit multipliers accomplished. In *2017 IEEE/ACM International Conference on Computer-Aided Design (ICCAD)*, pages 416–423, 2017.

[6] Sepide Saeedi, Alessandro Savino, and Stefano Di Calro. Design space exploration of approximate computing techniques with a reinforcement learning approach. In *2023 53rd Annual IEEE/IFIP International Conference on Dependable Systems and Networks Workshops (DSN-W)*, pages 167–170. IEEE, 2023.

[7] Sepide Saeedi, Alessio Carpegna, Alessandro Savino, and Stefano Di Carlo. Fast exploration of the impact of precision reduction on spiking neural networks. In *2024 IEEE International Conference on Design, Test and Technology of Integrated Systems (DTTIS)*, pages 1–6. IEEE, 2024.

[8] Sparsh Mittal. A survey of techniques for approximate computing. *ACM Computing Surveys (CSUR)*, 48(4):1–33, 2016.

[9] Nicola Jones. How to stop data centres from gobbling up the world's electricity. *Nature*, 561(7722):163–166, September 2018.

[10] R.H. Dennard, F.H. Gaensslen, Hwa-Nien Yu, V.L. Rideout, E. Bassous, and A.R. LeBlanc. Design of ion-implanted mosfet's with very small physical dimensions. *IEEE Journal of Solid-State Circuits*, 9(5):256–268, 1974.

[11] Honglan Jiang, Cong Liu, Leibo Liu, Fabrizio Lombardi, and Jie Han. A review, classification, and comparative evaluation of approximate arithmetic circuits. *J. Emerg. Technol. Comput. Syst.*, 13(4), aug 2017.

[12] Hadi Esmaeilzadeh, Emily Blem, Renee St. Amant, Karthikeyan Sankaralingam, and Doug Burger. Dark silicon and the end of multicore scaling. *IEEE Micro*, 32(3):122–134, 2012.

[13] John L. Hennessy and David A. Patterson. A new golden age for computer architecture. *Commun. ACM*, 62(2):48–60, January 2019.

[14] Alberto Bosio, Daniel Ménard, and Olivier Sentieys. General introduction. In Alberto Bosio, Daniel Ménard, and Olivier Sentieys, editors, *Approximate Computing Techniques: From Component- to Application-Level*, pages 1–10. Springer International Publishing, Cham, Switzerland, 2022.

[15] Swagath Venkataramani, Srimat T Chakradhar, Kaushik Roy, and Anand Raghunathan. Approximate computing and the quest for computing efficiency. In *Proceedings of the 52nd Annual Design Automation Conference*, pages 1–6, 2015.

[16] Vinay K Chippa, Srimat T Chakradhar, Kaushik Roy, and Anand Raghunathan. Analysis and characterization of inherent application resilience for approximate computing. In *Proceedings of the 50th Annual Design Automation Conference*, pages 1–9, 2013.

[17] Alberto Bosio, Daniel Ménard, and Olivier Sentieys, editors. *Approximate Computing Techniques: From Component- to Application-Level*. Springer International Publishing, Cham, Switzerland, 2022.

[18] Marcello Traiola, Arnaud Virazel, Patrick Girard, Mario Barbareschi, and Alberto Bosio. Testing approximate digital circuits: Challenges and opportunities. In *2018 IEEE 19th Latin-American Test Symposium (LATS)*, pages 1–6, 2018.

[19] Marcello Traiola, Alessandro Savino, Mario Barbareschi, Stefano Di Carlo, and Alberto Bosio. Predicting the impact of functional approximation: From component-to application-level. In *2018 IEEE 24th International Symposium on On-Line Testing And Robust System Design (IOLTS)*, pages 61–64. IEEE, 2018.

[20] Gennaro Rodrigues, Fernanda Lima Kastensmidt, and Alberto Bosio. Survey on approximate computing and its intrinsic fault tolerance. *Electronics*, 9(4), 2020.

[21] Shikai Li, Sunghyun Park, and Scott Mahlke. Sculptor: Flexible approximation with selective dynamic loop perforation. In *Proceedings of the 2018*

International Conference on Supercomputing, pages 341–351, 2018.

[22] Vassilis Vassiliadis, Konstantinos Parasyris, Charalambos Chalios, Christos D Antonopoulos, Spyros Lalis, Nikolaos Bellas, Hans Vandierendonck, and Dimitrios S Nikolopoulos. A programming model and runtime system for significance-aware energy-efficient computing. *ACM SIGPLAN Notices*, 50(8):275–276, 2015.

[23] Cindy Rubio-González, Cuong Nguyen, Hong Diep Nguyen, James Demmel, William Kahan, Koushik Sen, David H Bailey, Costin Iancu, and David Hough. Precimonious: Tuning assistant for floating-point precision. In *Proceedings of the international conference on high performance computing, networking, storage and analysis*, pages 1–12, 2013.

[24] Chih-Chieh Hsiao, Slo-Li Chu, and Chen-Yu Chen. Energy-aware hybrid precision selection framework for mobile gpus. *Computers & Graphics*, 37(5):431–444, 2013.

[25] Sharad Sinha and Wei Zhang. Low-power fpga design using memoization-based approximate computing. *IEEE Transactions on Very Large Scale Integration (VLSI) Systems*, 24(8):2665–2678, 2016.

[26] Qian Zhang, Ting Wang, Ye Tian, Feng Yuan, and Qiang Xu. Approxann: An approximate computing framework for artificial neural network. In *2015 Design, Automation & Test in Europe Conference & Exhibition (DATE)*, pages 701–706. IEEE, 2015.

[27] Divya Mahajan, Amir Yazdanbakhsh, Jongse Park, Bradley Thwaites, and Hadi Esmaeilzadeh. Towards statistical guarantees in controlling quality tradeoffs for approximate acceleration. *ACM SIGARCH Computer Architecture News*, 44(3):66–77, 2016.

[28] Debabrata Mohapatra, Vinay K. Chippa, Anand Raghunathan, and Kaushik Roy. Design of voltage-scalable meta-functions for approximate computing. In *2011 Design, Automation & Test in Europe*, pages 1–6, 2011.

[29] Lukas Sekanina, Zdenek Vasicek, and Vojtech Mrazek. Inexact arithmetic operators. In Alberto Bosio, Daniel Ménard, and Olivier Sentieys, editors, *Approximate Computing Techniques: From Component- to Application-Level*, pages 81–108. Springer International Publishing, Cham, Switzerland, 2022.

[30] Justine Bonnot, Daniel Ménard, and Karol Desnos. Analysis of the impact of approximate computing on the application quality. In Alberto Bosio, Daniel Ménard, and Olivier Sentieys, editors, *Approximate Computing Techniques: From Component- to Application-Level*, pages 145–176. Springer International Publishing, Cham, Switzerland, 2022.

[31] Justine Bonnot, Alexandre Mercat, Erwan Nogues, and Daniel Ménard. Approximate computing at the algorithmic level. In Alberto Bosio, Daniel Ménard, and Olivier Sentieys, editors, *Approximate Computing Techniques: From Component- to Application-Level*, pages 109–144. Springer International Publishing, Cham, Switzerland, 2022.

[32] Muhammad Awais, Hassan Ghasemzadeh Mohammadi, and Marco Platzner. Ldax: a learning-based fast design space exploration framework for approximate circuit synthesis. In *Proceedings of the 2021 on Great Lakes Symposium on VLSI*, pages 27–32, 2021.

[33] Yonggan Fu, Haoran You, Yang Zhao, Yue Wang, Chaojian Li, Kailash Gopalakrishnan, Zhangyang Wang, and Yingyan Lin. Fractrain: Fractionally squeezing bit savings both temporally and spatially for efficient dnn training. *Advances in Neural Information Processing Systems*, 33:12127–12139, 2020.

[34] Kumud Nepal, Yueting Li, R Iris Bahar, and Sherief Reda. Abacus: A technique for automated behavioral synthesis of approximate computing circuits. In *2014 design, automation & test in europe conference & exhibition (DATE)*, pages 1–6. IEEE, 2014.

[35] Kuan Wang, Zhijian Liu, Yujun Lin, Ji Lin, and Song Han. Haq: Hardware-aware automated quantization with mixed precision. In *Proceedings of the IEEE/CVF conference on computer vision and pattern recognition*, pages 8612–8620, 2019.

[36] Mario Barbareschi, Salvatore Barone, Alberto Bosio, Jie Han, and Marcello Traiola. A genetic-algorithm-based approach to the design of dct hardware accelerators. *ACM Journal on Emerging Technologies in Computing Systems (JETC)*, 18(3):1–25, 2022.

[37] Marcello Traiola, Alessandro Savino, and Stefano Di Carlo. Probabilistic estimation of the application-level impact of precision scaling in approximate computing applications. *Microelectronics Reliability*, 102:113309, 2019.

[38] Vojtech Mrazek, Muhammad Abdullah Hanif, Zdenek Vasicek, Lukas Sekanina, and Muhammad Shafique. 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*, pages 1–6, 2019.

[39] Andy D Pimentel. Exploring exploration: A tutorial introduction to embedded systems design space exploration. *IEEE Design & Test*, 34(1):77–90, 2016.

[40] Kurt Keutzer, A Richard Newton, Jan M Rabaey, and Alberto Sangiovanni-Vincentelli. System-level design: Orthogonalization of concerns and platform-based design. *IEEE transactions on computer-aided design of integrated circuits and systems*, 19(12):1523–1543, 2000.

[41] Pramudita Satria Palar, Yohanes Bimo Dwianto, Lavi Rizki Zuhal, Joseph Morlier, Koji Shimoyama, and Shigeru Obayashi. Multi-objective design space exploration using explainable surrogate models. *Structural and Multidisciplinary Optimization*, 67(3):38, 2024.

[42] Qiang Xu, Todd Mytkowicz, and Nam Sung Kim. Approximate computing: A survey. *IEEE Design & Test*, 33(1):8–22, 2015.

[43] Muhammad Shafique, Rehan Hafiz, Semeen Rehman, Walaa El-Harouni, and

Jörg Henkel. Cross-layer approximate computing: From logic to architectures. In *Proceedings of the 53rd Annual Design Automation Conference*, pages 1–6, 2016.

[44] Manu Manuel, Arne Kreddig, Simon Conrady, Nguyen Anh Vu Doan, and Walter Stechele. Model-based design space exploration for approximate image processing on fpga. In *2020 IEEE Nordic Circuits and Systems Conference (NorCAS)*, pages 1–7. IEEE, 2020.

[45] Soheil Hashemi, Hokchhay Tann, Francesco Buttafuoco, and Sherief Reda. Approximate computing for biometric security systems: A case study on iris scanning. In *2018 Design, Automation & Test in Europe Conference & Exhibition (DATE)*, pages 319–324. IEEE, 2018.

[46] Ahmed T Elthakeb, Prannoy Pilligundla, Fatemehsadat Mireshghallah, Amir Yazdanbakhsh, and Hadi Esmaeilzadeh. Releq: A reinforcement learning approach for automatic deep quantization of neural networks. *IEEE micro*, 40(5):37–45, 2020.

[47] Inigo Goiri, Ricardo Bianchini, Santosh Nagarakatte, and Thu D Nguyen. Approxhadoop: Bringing approximations to mapreduce frameworks. In *Proceedings of the Twentieth International Conference on Architectural Support for Programming Languages and Operating Systems*, pages 383–397, 2015.

[48] Arnab Raha, Swagath Venkataramani, Vijay Raghunathan, and Anand Raghunathan. Quality configurable reduce-and-rank for energy efficient approximate computing. In *2015 Design, Automation & Test in Europe Conference & Exhibition (DATE)*, pages 665–670. IEEE, 2015.

[49] Felipe Sampaio, Muhammad Shafique, Bruno Zatt, Sergio Bampi, and Jörg Henkel. Approximation-aware multi-level cells stt-ram cache architecture. In *2015 International Conference on Compilers, Architecture and Synthesis for Embedded Systems (CASES)*, pages 79–88. IEEE, 2015.

[50] Ye Tian, Qian Zhang, Ting Wang, Feng Yuan, and Qiang Xu. Approxma: Approximate memory access for dynamic precision scaling. In *Proceedings of the 25th edition on Great Lakes Symposium on VLSI*, pages 337–342, 2015.

[51] Ashish Ranjan, Swagath Venkataramani, Xuanyao Fong, Kaushik Roy, and Anand Raghunathan. Approximate storage for energy efficient spintronic memories. In *Proceedings of the 52nd Annual Design Automation Conference*, pages 1–6, 2015.

[52] Muhammad Shafique, Waqas Ahmad, Rehan Hafiz, and Jörg Henkel. A low latency generic accuracy configurable adder. In *Proceedings of the 52nd Annual Design Automation Conference*, DAC '15, New York, NY, USA, 2015. Association for Computing Machinery.

[53] Renée St. Amant, Amir Yazdanbakhsh, Jongse Park, Bradley Thwaites, Hadi Esmaeilzadeh, Arjang Hassibi, Luis Ceze, and Doug Burger. General-purpose code acceleration with limited-precision analog computation. *ACM SIGARCH Computer Architecture News*, 42(3):505–516, 2014.

[54] V. Inder Kumar and Santanu Kapat. Per-core configurable power supply for multi-core processors with ultra-fast dvs voltage transitions. In *2022 IEEE Applied Power Electronics Conference and Exposition (APEC)*, pages 1028–1034, 2022.

[55] Alireza Senobari, Jafar Vafaei, Omid Akbari, Christian Hochberger, and Muhammad Shafique. A quality-aware voltage overscaling framework to improve the energy efficiency and lifetime of tpus based on statistical error modeling. *IEEE Access*, 2024.

[56] Georgios Chatzitsompanis and Georgios Karakonstantis. On the facilitation of voltage over-scaling and minimization of timing errors in floating-point multipliers. In *2023 IEEE 29th International Symposium on On-Line Testing and Robust System Design (IOLTS)*, pages 1–7, 2023.

[57] Naeem Seliya, Taghi M Khoshgoftaar, and Jason Van Hulse. A study on the relationships of classifier performance metrics. In *2009 21st IEEE international conference on tools with artificial intelligence*, pages 59–66. IEEE, 2009.

[58] Benjamin Carrion Schafer and Zi Wang. High-level synthesis design space exploration: Past, present, and future. *IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems*, 39(10):2628–2639, 2019.

[59] Andy D. Pimentel. *Methodologies for Design Space Exploration*, pages 1–31. Springer Nature Singapore, Singapore, 2022.

[60] Alberto Sangiovanni-Vincentelli and Grant Martin. Platform-based design and software design methodology for embedded systems. *IEEE Design & Test of computers*, 18(6):23–33, 2001.

[61] Luiz Santos, Sandro Rigo, Rodolfo Azevedo, and Guido Araujo. *Electronic System Level Design*, pages 3–10. Springer Netherlands, Dordrecht, 2011.

[62] Matthias Gries. Methods for evaluating and covering the design space during early design development. *Integration*, 38(2):131–183, 2004.

[63] Amit Kumar Singh, Muhammad Shafique, Akash Kumar, and Jörg Henkel. Mapping on multi/many-core systems: Survey of current and emerging trends. In *Proceedings of the 50th Annual Design Automation Conference*, pages 1–10, 2013.

[64] Sean C. Smithson, Guang Yang, Warren J. Gross, and Brett H. Meyer. Neural networks designing neural networks: Multi-objective hyper-parameter optimization. In *2016 IEEE/ACM International Conference on Computer-Aided Design (ICCAD)*, pages 1–8, 2016.

[65] Etienne Dupuis, David Novo, Ian O'Connor, and Alberto Bosio. On the automatic exploration of weight sharing for deep neural network compression. In *2020 Design, Automation Test in Europe Conference Exhibition (DATE)*, pages 1319–1322, 2020.

[66] Alessandro Savino, Marcello Traiola, Stefano Di Carlo, and Alberto Bosio. Efficient neural network approximation via bayesian reasoning. In *2021 24th*

International Symposium on Design and Diagnostics of Electronic Circuits Systems (DDECS), pages 45–50, 2021.

[67] Henry Hoffmann. Jouleguard: Energy guarantees for approximate applications. In *Proceedings of the 25th Symposium on Operating Systems Principles*, pages 198–214, 2015.

[68] Salim Ullah, Siva Satyendra Sahoo, and Akash Kumar. Clapped: A design framework for implementing cross-layer approximation in fpga-based embedded systems. In *2021 58th ACM/IEEE Design Automation Conference (DAC)*, pages 475–480. IEEE, 2021.

[69] Muhammad Awais Rajput, Sultan Alyami, Qazi Arbab Ahmed, Hani Alshahrani, Yousef Asiri, and Asadullah Shaikh. Improved learning-based design space exploration for approximate instance generation. *IEEE Access*, 11:18291–18299, 2023.

[70] Bharath Srinivas Prabakaran, Vojtech Mrazek, Zdenek Vasicek, Lukas Sekanina, and Muhammad Shafique. Xel-fpgas: An end-to-end automated exploration framework for approximate accelerators in fpga-based systems. In *2023 IEEE/ACM International Conference on Computer Aided Design (ICCAD)*, pages 1–9. IEEE, 2023.

[71] Jongse Park, Kangqi Ni, Xin Zhang, Hadi Esmaeilzadeh, and Mayur Naik. Expectation-oriented framework for automating approximate programming. In *Workshop on Approximate Computing Across the System Stack (WACAS)*, 2014.

[72] Michal Pinos, Vojtech Mrazek, Filip Vaverka, Zdenek Vasicek, and Lukas Sekanina. Acceleration techniques for automated design of approximate convolutional neural networks. *IEEE Journal on Emerging and Selected Topics in Circuits and Systems*, 13(1):212–224, 2023.

[73] Jorge Castro-Godínez, Julián Mateus-Vargas, Muhammad Shafique, and Jörg Henkel. 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*, pages 1–9, 2020.

[74] Younghoon Kim, Swagath Venkataramani, Kaushik Roy, and Anand Raghunathan. Designing approximate circuits using clock overgating. In *Proceedings of the 53rd Annual Design Automation Conference*, pages 1–6, 2016.

[75] Alessandro Savino, Michele Portolan, Regis Leveugle, and Stefano Di Carlo. Approximate computing design exploration through data lifetime metrics. In *2019 IEEE European Test Symposium (ETS)*, pages 1–7. IEEE, 2019.

[76] Amir Yazdanbakhsh, Gennady Pekhimenko, Bradley Thwaites, Hadi Esmaeilzadeh, Onur Mutlu, and Todd C Mowry. Rfvp: Rollback-free value prediction with safe-to-approximate loads. *ACM Transactions on Architecture and Code Optimization (TACO)*, 12(4):1–26, 2016.

[77] Guilherme Paim, Leandro Mateus Giacomini Rocha, Hussam Amrouch, Eduardo Antônio César da Costa, Sergio Bampi, and Jörg Henkel. A cross-layer gate-level-to-application co-simulation for design space exploration of approximate circuits in hevc video encoders. *IEEE Transactions on Circuits and Systems for Video Technology*, 30(10):3814–3828, 2019.

[78] Georgios Zervakis, Sotirios Xydis, Dimitrios Soudris, and Kiamal Pekmestzi. Multi-level approximate accelerator synthesis under voltage island constraints. *IEEE Transactions on Circuits and Systems II: Express Briefs*, 66(4):607–611, 2018.

[79] Shubham Jain, Swagath Venkataramani, and Anand Raghunathan. Approximation through logic isolation for the design of quality configurable circuits. In *2016 Design, Automation & Test in Europe Conference & Exhibition (DATE)*, pages 612–617. IEEE, 2016.

[80] Tanfer Alan, Andreas Gerstlauer, and Jörg Henkel. Cross-layer approximate hardware synthesis for runtime configurable accuracy. *IEEE Transactions on Very Large Scale Integration (VLSI) Systems*, 29(6):1231–1243, 2021.

[81] Jorge Echavarria, Stefan Wildermann, and Jürgen Teich. Approximate logic synthesis of very large boolean networks. In *2021 Design, Automation & Test in Europe Conference & Exhibition (DATE)*, pages 1552–1557. IEEE, 2021.

[82] Marcello Traiola, Jorge Echavarria, Alberto Bosio, Jürgen Teich, and Ian O'Connor. Design space exploration of approximation-based quadruple modular redundancy circuits. In *2021 IEEE/ACM International Conference On Computer Aided Design (ICCAD)*, pages 1–9. IEEE, 2021.

[83] Jin Miao, Andreas Gerstlauer, and Michael Orshansky. Multi-level approximate logic synthesis under general error constraints. In *2014 IEEE/ACM International Conference on Computer-Aided Design (ICCAD)*, pages 504–510. IEEE, 2014.

[84] Jingxiao Ma, Soheil Hashemi, and Sherief Reda. Approximate logic synthesis using boolean matrix factorization. *IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems*, 41(1):15–28, 2021.

[85] Jorge Echavarria, Oliver Keszocze, and Jürgen Teich. Probability-based dse of approximated lut-based fpga designs. In *2022 IEEE 15th Dallas Circuit And System Conference (DCAS)*, pages 1–5. IEEE, 2022.

[86] Alessio Carpegna, Alessandro Savino, and Stefano Di Carlo. Spiker: an fpga-optimized hardware accelerator for spiking neural networks. In *2022 IEEE Computer Society Annual Symposium on VLSI (ISVLSI)*, pages 14–19. IEEE, 2022.

[87] Marcel Stimberg, Romain Brette, and Dan FM Goodman. Brian 2, an intuitive and efficient neural simulator. *eLife*, 8:e47314, August 2019.

[88] Ieee standard for interval arithmetic. *IEEE Std 1788-2015*, pages 1–97, 2015.

[89] James Paul Turner and Thomas Nowotny. Arpra: An arbitrary precision range analysis library. *Frontiers in Neuroinformatics*, 15, 2021.

[90] Corinna Cortes Yann LeCunn and Christopher J.C Burges. The mnist database.

[91] David Heeger. Poisson model of spike generation. 10 2000.

[92] Marcello Traiola, Alessandro Savino, and Stefano Di Carlo. Probabilistic estimation of the application-level impact of precision scaling in approximate computing applications. *Microelectronics Reliability*, 102:113309, 2019.

[93] Alessio Carpegna, Alessandro Savino, and Stefano Di Carlo. Spiker: an fpga-optimized hardware accelerator for spiking neural networks. In IEEE, editor, *IEEE ISVLSI 2022*, 2022.

[94] Alessandro Savino, Michele Portolan, Regis Leveugle, and Stefano Di Carlo. Approximate computing design exploration through data lifetime metrics. In *2019 IEEE European Test Symposium (ETS)*, pages 1–7, 2019.

[95] CHCR Ribeiro. Reinforcement learning agents. *Artificial intelligence review*, 17(3):223–250, 2002.

[96] Nan Wu, Yuan Xie, and Cong Hao. Ironman: Gnn-assisted design space exploration in high-level synthesis via reinforcement learning. In *Proceedings of the 2021 Great Lakes Symposium on VLSI*, GLSVLSI '21, page 39–44, New York, NY, USA, 2021. Association for Computing Machinery.

[97] Quentin Gautier, Alric Althoff, Christopher L Crutchfield, and Ryan Kastner. Sherlock: A multi-objective design space exploration framework. *ACM Transactions on Design Automation of Electronic Systems (TODAES)*, 27(4):1–20, 2022.

[98] Yesmina Jaafra, Jean Luc Laurent, Aline Deruyver, and Mohamed Saber Naceur. Reinforcement learning for neural architecture search: A review. *Image and Vision Computing*, 89:57–66, 2019.

[99] Rahim Mammadli, Ali Jannesari, and Felix Wolf. Static neural compiler optimization via deep reinforcement learning. In *2020 IEEE/ACM 6th Workshop on the LLVM Compiler Infrastructure in HPC (LLVM-HPC) and Workshop on Hierarchical Parallelism for Exascale Computing (HiPar)*, pages 1–11. IEEE, 2020.

[100] Huanting Wang, Zhanyong Tang, Cheng Zhang, Jiaqi Zhao, Chris Cummins, Hugh Leather, and Zheng Wang. Automating reinforcement learning architecture design for code optimization. In *Proceedings of the 31st ACM SIGPLAN International Conference on Compiler Construction*, pages 129–143, 2022.

[101] Hongming Zhang and Tianyang Yu. Taxonomy of reinforcement learning algorithms. In *Deep reinforcement learning: Fundamentals, research and applications*, pages 125–133. Springer, 2020.

[102] Mohit Sewak. Temporal difference learning, sarsa, and q-learning: Some popular value approximation based reinforcement learning approaches. In *Deep reinforcement learning: Frontiers of artificial intelligence*, pages 51–63. Springer, 2019.

[103] Hafiq Anas, Wee Hong Ong, and Owais Ahmed Malik. Comparison of deep q-learning, q-learning and sarsa reinforced learning for robot local navigation. In *International Conference on Robot Intelligence Technology and Applications*, pages 443–454. Springer, 2021.

[104] Constantinos Daskalakis, Dylan J Foster, and Noah Golowich. Independent policy gradient methods for competitive reinforcement learning. *Advances in neural information processing systems*, 33:5527–5540, 2020.

[105] Richard S Sutton, David McAllester, Satinder Singh, and Yishay Mansour. Policy gradient methods for reinforcement learning with function approximation. *Advances in neural information processing systems*, 12, 1999.

[106] Ofir Nachum, Mohammad Norouzi, Kelvin Xu, and Dale Schuurmans. Bridging the gap between value and policy based reinforcement learning. *Advances in neural information processing systems*, 30, 2017.

[107] Wangshu Zhu and Andre Rosendo. A functional clipping approach for policy optimization algorithms. *IEEE Access*, 9:96056–96063, 2021.

[108] Alberto Del Rio, David Jimenez, and Javier Serrano. Comparative analysis of a3c and ppo algorithms in reinforcement learning: A survey on general environments. *IEEE Access*, 2024.

[109] Kaijie Feng, Xiaoya Fan, Jianfeng An, Xiping Wang, Kaiyue Di, Jiangfei Li, Minghao Lu, and Chuxi Li. Erdse: efficient reinforcement learning based design space exploration method for cnn accelerator on resource limited platform. *Graphics and Visual Computing*, 4:200024, 2021.

[110] Francisco S Melo. Convergence of q-learning: A simple proof. *Institute Of Systems and Robotics, Tech. Rep*, pages 1–4, 2001.

[111] Christopher JCH Watkins and Peter Dayan. Q-learning. *Machine learning*, 8(3):279–292, 1992.

[112] Leslie Pack Kaelbling, Michael L Littman, and Andrew W Moore. Reinforcement learning: A survey. *Journal of artificial intelligence research*, 4:237–285, 1996.

[113] Sean Meyn. The projected bellman equation in reinforcement learning. *IEEE Transactions on Automatic Control*, 69(12):8323–8337, 2024.

[114] George Cybenko, Robert Gray, and Katsuhiro Moizumi. Q-learning: A tutorial and extensions. *Mathematics of Neural Networks: Models, Algorithms and Applications*, pages 24–33, 1997.

[115] Sai Vineet Swaroop, Vishnu Deepak, Rohit Kumar Konala, Sai Kumar Vemula, and Deepak Prasad. Gymnasium. https://github.com/s-vineet/gymnasium, 2021. Accessed: 2023-04-07.

[116] Greg Brockman, Vicki Cheung, Ludwig Pettersson, Jonas Schneider, John Schulman, Jie Tang, and Wojciech Zaremba. Openai gym. *arXiv preprint arXiv:1606.01540*, 2016.

[117] Alhussein Fawzi, Matej Balog, Aja Huang, Thomas Hubert, Bernardino

Romera-Paredes, Mohammadamin Barekatain, Alexander Novikov, Francisco J R. Ruiz, Julian Schrittwieser, Grzegorz Swirszcz, et al. Discovering faster matrix multiplication algorithms with reinforcement learning. *Nature*, 610(7930):47–53, 2022.

[118] Yann LeCun, Yoshua Bengio, and Geoffrey Hinton. Deep learning. *nature*, 521(7553):436–444, 2015.

[119] Belle A Shenoi. *Introduction to digital signal processing and filter design*. John Wiley & Sons, 2005.

[120] Rizwana Mehboob, Shoab A. Khan, and Rabia Qamar. Fir filter design methodology for hardware optimized implementation. *IEEE Transactions on Consumer Electronics*, 55(3):1669–1673, 2009.

[121] Subhabrata Roy and Abhijit Chandra. A survey of fir filter design techniques: low-complexity, narrow transition-band and variable bandwidth. *Integration*, 77:193–204, 2021.

[122] Siyuan Xu and Benjamin Carrion Schafer. Exposing approximate computing optimizations at different levels: From behavioral to gate-level. *IEEE Transactions on Very Large Scale Integration (VLSI) Systems*, 25(11):3077–3088, 2017.

[123] Arne Kreddig, Simon Conrady, Manu Manuel, and Walter Stechele. A framework for hardware-accelerated design space exploration for approximate computing on fpga. In *2021 24th Euromicro Conference on Digital System Design (DSD)*, pages 1–8. IEEE, 2021.

[124] Swagath Venkataramani, Vijayalakshmi Srinivasan, Wei Wang, Sanchari Sen, Jintao Zhang, Ankur Agrawal, Monodeep Kar, Shubham Jain, Alberto Mannari, Hoang Tran, et al. Rapid: Ai accelerator for ultra-low precision training and inference. In *2021 ACM/IEEE 48th Annual International Symposium on Computer Architecture (ISCA)*, pages 153–166. IEEE, 2021.

[125] Swagath Venkataramani, Xiao Sun, Naigang Wang, Chia-Yu Chen, Jungwook Choi, Mingu Kang, Ankur Agarwal, Jinwook Oh, Shubham Jain, Tina Babinsky, et al. Efficient ai system design with cross-layer approximate computing. *Proceedings of the IEEE*, 108(12):2232–2250, 2020.

This Ph.D. thesis has been typeset by means of the TEX-system facilities. The typesetting engine was pdfLATEX. The document class was `toptesi`, by Claudio Beccari, with option `tipotesi=scudo`. This class is available in every up-to-date and complete TEX-system installation.