RESOURCE-AWARE OPTIMIZATION TECHNIQUES FOR MACHINE LEARNING INFERENCE ON HETEROGENEOUS EMBEDDED SYSTEMS

by

Ourania Spantidi

M.S., Southern Illinois University, 2018

A Dissertation
Submitted in Partial Fulfillment of the Requirements for the
Doctor of Philosophy Degree

School of Electrical, Computer and Biomedical Engineering
in the Graduate School
Southern Illinois University Carbondale
May 2023

DISSERTATION APPROVAL

RESOURCE-AWARE OPTIMIZATION TECHNIQUES FOR MACHINE LEARNING

INFERENCE ON HETEROGENEOUS EMBEDDED SYSTEMS

by

Ourania Spantidi

A Dissertation Submitted in Partial

Fulfillment of the Requirements

for the Degree of

Doctor of Philosophy

in the field of Electrical and Computer Engineering

Graduate School
Southern Illinois University Carbondale
March 21, 2023

AN ABSTRACT OF THE DISSERTATION OF

Ourania Spantidi, for the Doctor of Philosophy degree in Electrical and Computer Engineering, presented on March 21, 2023, at Southern Illinois University Carbondale.

TITLE: RESOURCE-AWARE OPTIMIZATION TECHNIQUES FOR MACHINE LEARNING INFERENCE ON HETEROGENEOUS EMBEDDED SYSTEMS

MAJOR PROFESSOR: Dr. I. Anagnostopoulos

With the increasing adoption of Deep Neural Networks (DNNs) in modern applications, there has been a proliferation of computationally and power-hungry workloads, which has necessitated the use of embedded systems with more sophisticated, heterogeneous approaches to accommodate these requirements. One of the major solutions to tackle these challenges has been the development of domain-specific accelerators, which are highly optimized for the computationally intensive tasks associated with DNNs. These accelerators are designed to take advantage of the unique properties of DNNs, such as parallelism and data locality, to achieve high throughput and energy efficiency. Domain-specific accelerators have been shown to provide significant improvements in performance and energy efficiency compared to traditional general-purpose processors and are becoming increasingly popular in a range of applications such as computer vision and speech recognition. However, designing these architectures and managing their resources can be challenging, as it requires a deep understanding of the workload and the system's unique properties. Achieving a favorable balance between performance and power consumption is not always straightforward and requires careful design decisions to fully exploit the benefits of the underlying hardware. This dissertation aims to address these challenges by presenting solutions that enable low energy consumption without compromising performance for heterogeneous embedded systems. Specifically, this dissertation will focus on three topics: (i) the utilization of approximate computing concepts and approximate accelerators for energy-efficient DNN inference, (ii) the integration of formal properties in the systematic employment of approximate computing concepts, and (iii) resource management techniques on heterogeneous embedded systems. In summary, this dissertation provides a comprehensive study of solutions that can improve the energy efficiency of heterogeneous embedded systems, enabling them to perform computationally intensive tasks associated with modern applications that incorporate DNNs without compromising on performance. The results of this dissertation demonstrate the effectiveness of the proposed solutions and their potential for wide-ranging practical applications.

DEDICATION

To my late grandfather Nikos, whose eyes would always light up with excitement whenever I spoke about my life and studies in the USA – a country he loved dearly.

To my late mother-in-law Lena, whose radiant spirit, generosity, and endless love touched my life in countless ways. She will always be remembered as a shining example of what it means to be a caring and thoughtful person.

They will forever be missed, especially on the day of my graduation and the start of my career, which I dedicate to them.

ACKNOWLEDGMENTS

In the Acknowledgment section of every dissertation, it is common for authors to express their gratitude to their advisors for their guidance and support. However, in my case, I feel that words cannot fully convey the extent of the confidence, knowledge, and skills (both hard and soft) that I have gained under the guidance of my advisor, Dr. Iraklis Anagnostopoulos. Dr. Anagnostopoulos' belief in my abilities has given me the confidence to take on challenges and pursue my dreams. I am grateful for his selfless dedication, patience, and encouragement, which have played an instrumental role throughout my Ph.D. journey. From long nights spent at the lab for paper submissions, research discussions on future prospects, and attending conferences together, to invaluable support during job interviews, Dr. Anagnostopoulos has been a constant source of help and motivation. Thank you, Dr. Anagnostopoulos, for being a perfect advisor and a great mentor.

There is one person who played an important role throughout all the years I spent in Carbondale, and that is Dr. Spyros Tragoudas. His support and guidance have been invaluable assets to me. His care and kindness extended beyond academics, and I am thankful for the many times he has gone above and beyond to support me.

I would also like to thank the rest of my dissertation committee. Dr. Haibo Wang and Dr. Chao Lu have not only been some of the best professors I have had, but also incredible mentors who have supported me through thick and thin. And to Dr. Dimitrios Soudris, who agreed to participate in my committee despite the distance between us, I am forever grateful. Your willingness to lend your expertise and guidance has helped make this dissertation a reality. Thank you all from the bottom of my heart.

I also want to express my deep appreciation for the support and guidance I received from Richard Smith during my time in the IT Security group. He has always been understanding of my time off requests to attend conferences and visit my family in Europe and has accommodated my schedule accordingly. Being a part of IT Security was a great experience, and I thoroughly enjoyed our meetings and the work we did together. Thank you all for making my time during my assistantship so fulfilling.

As I prepare to conclude my journey here in southern Illinois, I want to take a moment to express my heartfelt appreciation to everyone who has played a part in my life over the past 6.5 years. To my fellow members of the Greek community, thank you for giving me a sense of belonging and thank you for easing my homesickness. And to all the amazing people I have met along the way, thank you for the laughter, the conversations, and the unforgettable experiences. I will always cherish the friendships that were formed here at SIU, and I will forever be grateful for the impact you have had on my life.

To my dear friends back in Europe who have supported me from afar and remained an important part of my life, thank you for your love and friendship. Though we may be far apart, you are always close in my heart.

I would not be where I am today without the support of my family. My parents have been providing me with endless love, encouragement, and support throughout my entire Ph.D. journey. My father Giorgos is my biggest cheerleader, followed by my dear grandma Chrysi, and my mother Ergina always made sure to send care packages year-round. They have always been there for me, whether it was to lend a listening ear, help with a problem, or provide a shoulder to cry on. I also want to express my appreciation to my brother Dimitris. His belief in me has been a driving force throughout my life, and I couldn't be more proud of him for all the amazing things he has accomplished.

Of course, the past 6.5 years would have been a total nightmare if it wasn't for my partner Vasilis. Vasilis has always believed in me, even when I struggled to believe in myself, and has been there to celebrate my successes and lift me up through my failures. Thank you for being in my life. I look forward to all the adventures that lie ahead for us.

To new beginnings!

PREFACE

This work has been supported in part by grant NSF IIP 1361847 from the NSF I/UCRC for Embedded systems at SIUC. Any opinions, findings, and conclusions or recommendations expressed in this material are those of the authors and do not necessarily reflect the views of the National Science Foundation.

TABLE OF CONTENTS

LIST OF FIGURES

LIST OF TABLES

CHAPTER 1
INTRODUCTION

Deep neural networks (DNNs) have revolutionized the field of artificial intelligence by enabling machines to learn from data and make intelligent decisions. DNNs are composed of multiple layers of interconnected nodes that can learn to identify patterns and features in the input data. Nowadays, modern applications are very likely to be based on image classification tasks, they might require depth and hand pose estimation on top of image classification [10], and overall they could potentially require tasks such as speech and image recognition among others [11]. All of these tasks can be addressed through DNN deployment on target devices. As the demand for intelligent and autonomous devices grows, developing efficient DNN architectures that can run on embedded devices has become an essential area of research.

1.1 MODERN EMBEDDED SYSTEMS

Embedded systems comprise a variety of different resources in terms of computational capabilities. The deployment of DNN applications can be very intense for embedded devices due to their high computational demands, and hardware accelerators have been developed to address this issue. These accelerators are becoming increasingly popular in a range of applications, playing a key role in enabling the deployment of DNNs in resource-constrained environments such as embedded systems [12]. Alongside such accelerators, Neural Processing Units (NPUs) prevail as a solution to the accuracy throughput trade-off. NPUs have been established as the tailored computing architecture paradigm to accelerate DNNs and as Facebook reports [13], we are “reaching an inflection point” towards the integration on embedded devices.

The heart of any NPU is a massive array of multiply-accumulate (MAC) units that profoundly accelerate the computations required by various phases throughout the execution of DNN inference. For instance, the Google Edge TPU [11] consists of 4K MAC units and employs a systolic MAC array. Other representative examples of accelerators include the neural compute engines integrated into Apple's M1 SoC and Intel's Nervana, while other platforms comprise multiple NPUs as a way to serve multiple DNNs at the same time [11]. However, performing such a high number of MAC operations per cycle results in very high power consumption [14] and excessive on-chip temperatures and localized hot spots that eventually form thermal bottlenecks [15].

1.2 MOTIVATION

This dissertation is focusing on energy-efficient embedded systems through resource management techniques. This section will provide some motivational examples and observations that have led to the need for these methodologies.

1.2.1 Utilizing Approximate Computing for efficient DNN inference

Approximate computing is a computing paradigm that aims to build energy-efficient systems, by exploiting the inherent error tolerance that characterizes DNNs. Not all computations require absolute precision, and by allowing some degree of imprecision or error, significant gains in performance and energy consumption can be achieved [16]. With the increasing demand for energy-efficient and high-performance computing, approximate computing has gained significant attention and is considered a promising solution for many real-world applications.

1.2.1.1 Approximate circuits

Specifically, there have been approximate computing efforts that have targeted the multiplication operation and thus have led to the design of approximate multipliers [1,3,17]. Most of these approaches employ a systolic $M \times M$ MAC array (similar to the one used on the Google TPU [11]), and replace the accurate (exact) MAC units with approximate ones [18], altering the circuit to accommodate the inclusion of an

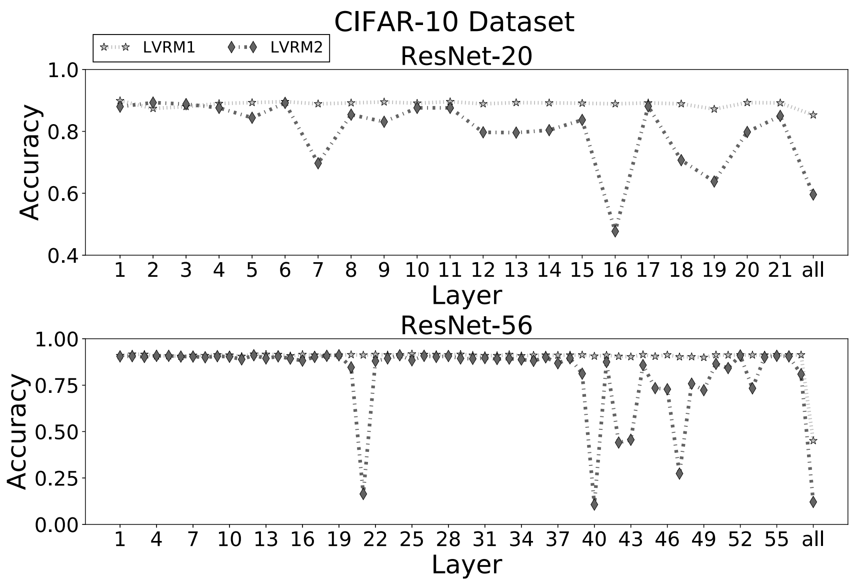

Figure 1.1: LVRM example on DNN multiplications [1]

approximate multiplier.

The final goal of approximate computing techniques remains the achievement of high savings in terms of energy with minimal accuracy loss during DNN inference. However, an approximate multiplier might inflict grave performance degradation on a given DNN if all the multiplications get assigned to it. Some layers of a DNN might be proven to be more "sensitive" than others, and in such cases, they should not be targets of approximate operations. On the other hand, other layers of the same DNN might be able to tolerate said operations to the point of being completely mapped to approximation without lowering the inference accuracy.

Specifically, Figure 1.1 demonstrates an example of DNN error sensitivity on two different DNNs: ResNet-20 and ResNet-56 on the CIFAR-10 dataset. The depicted example utilizes the Low-Variance Reconfigurable Multiplier (LVRM) presented in [1]. It comprises two approximate modes: LVRM1 which introduces low error and LVRM2 which introduces higher error. For each of the shown DNNs, one by one all the layers are being mapped entirely to one multiplier mode while the rest remain mapped to the exact multiplier. The DNN is then executed for inference and the accuracy is being monitored and shown in the y-axis. It is evident that not all layers behave well under aggressive approximation, for instance, layer 40 of ResNet-56 drops almost 80% from the baseline. Therefore, to keep the DNN accuracy as close to the baseline as possible while reaping the benefits of approximate multiplier utilization, further exploration of the error resilience of each DNN layer is needed.

Observation: It is not a straightforward task to find multiplication-to-approximation assignments in DNNs since they vary in layer sensitivity.

Many error metrics have been employed in the approximate computing literature to evaluate the impact of approximate circuits on DNN accuracy. There are general error metrics such as the *error rate*, *worst-case and average Hamming distance*, which are mainly used when all bits are considered to have the same significance [19]. However, when it comes to approximate multipliers, each bit has a different weight, and arithmetic error metrics are utilized instead, e.g. the *worst-case arithmetic error*, *mean absolute error*, *mean squared error*, *relative worst-case error* and *mean relative error*. Normalized error metrics are used when the bit-width increases. A common way to categorize approximate multipliers is based on their Mean Relative Error (MRE) which is derived from the Mean Absolute Error (MAE) metric. Specifically, for n samples, the MAE is calculated as the sum of the absolute differences in magnitude between the original and approximate circuits, averaged over all inputs [19]. Specifically, when we replace the exact multiplication with an approximate one during DNN inference, the MAE would be acquired like so:

$$MAE = \frac{1}{n}\sum_{\forall x} |accurate(x) - approximate(x)| \tag{1.1}$$

and the MRE value would be calculated by replacing the error distance in Equation 1.1 with the relative error distance:

$$MRE = \frac{1}{n} \sum_{\forall x} \frac{|accurate(x) - approximate(x)|}{accurate(x)} \tag{1.2}$$

In both equations 1.1, 1.2, $accurate(x)$ represents the exact multiplication x between a weight and an activation, while $approximate(x)$ represents the approximate one, for a total of n operations.

As an example, the EvoApprox library [20] contains multiple static approximate multipliers of varying MRE, which ranges from 0.01% up to 99.22%. Note that, an MRE value of 99.22% means a very high approximation and the respective multiplier would not be able to be applied to an entire DNN without resulting in unacceptable accuracy results. Therefore only a layer-wise mapping technique would be able to utilize such high MRE multipliers. Note that, a high MRE percentage results in most cases in high savings in terms of energy.

Observation: A static multiplier error does not allow for very flexible approximation applications.

Current techniques for designing approximate multipliers mainly focus on introducing low-error while also applying retraining to recover the accuracy loss [21]. However, this is not always feasible due to proprietary datasets and models [5, 18]. Moreover, retraining requires simulating the approximate multiplier, precluding the exploitation of hardware optimizations (e.g., AVX instructions) and exponentially increases the training time, especially for deeper networks [5]. Since we target embedded devices, training is also a time-consuming and very computationally-intensive procedure for their limited resources. Additionally, retraining might not even be an option since some datasets can be proprietary [4].

Observation: Retraining is very useful in regaining some of the lost accuracy inflicted by approximation but should be avoided.

1.2.1.2 Approximate hardware

Heterogeneous DNN accelerators (HDAs) have recently prevailed as a way to balance the latency/power dilemma imposed by modern DNNs and technology scaling during inference. Heterogeneity can be introduced by integrating multiple smaller sub-NPUs, each of which organizes the MAC units in different topologies, thus taking advantage of the distinct computing requirements of different NNs. Generally, such HDAs are application-specific, and as the chip resources are limited and split between different sub-accelerators, each sub-NPU may be less powerful compared to a homogeneous one [10]. Other works reduce the number of bits handled by the MAC units, usually by clock-gating the respective bits [22]. As a result, energy savings are obtained due to the reduced switching activity. However, NPU scaling is restricted since such approaches do not reduce the area of MAC units. The reduction of utilized bits might also result in significant accuracy loss due to the reduction in the numerical representation of the MAC inputs (e.g., weights, activations) if no additional action is taken [14].

Nowadays, many applications are built upon very complex functionalities, requiring the execution of multiple DNNs [10, 13]. Diverse tasks require combinations of various DNNs, each one comprising different amounts of operations and layers, leading to the construction of heterogeneous multi-DNN workloads [10]. when an application relies on multiple DNNs to achieve a target performance, there are many challenges inflicted on end devices, especially when they need to operate within constrained power budgets. For instance, in an IoT application where smart glasses require the continuous DNN application on camera-sensed video, to keep a certain performance threshold the battery could be drained even in less than 2 hours [23]. Therefore, there is a need to achieve major improvements in energy efficiency without any compromise in the final performance.

Observation: There is a need for a framework that enables the efficient execution of diverse DNN workloads on HDAs.

1.2.2 Formal Property Integration in Approximate Computing techniques

As mentioned in Section 1.2.1, approximate computing [24] has emerged as the dominant paradigm that trades quality loss for energy and performance gains. Approximate accelerators have been proposed following two main architectural designs: (i) tile-based MAC units, and (ii) MAC units with reconfigurable approximate multipliers. Regarding the first method, instead of having a mesh of MAC units, the DNN accelerator is organized as a mesh of tiles. Each tile hosts a combination of exact and multiple static approximate multipliers [5]. Then, based on the DNN and the system constraints, an analysis is performed to map the DNN layers on the appropriate multiplier (exact or approximate), while the rest are power-gated. Since each layer of a DNN has different error sensitivity, the layer-to-static multiplier mapping is not a trivial problem; the design space is so big that exhaustive exploration is prohibited [5]. Additionally, such a coarse-grain *layer-based* mapping of approximation limits the energy gains [1]. The second method is more fine-grain, flexible, and focuses on the design of reconfigurable approximate multipliers [1,3]. Such multipliers sacrifice area to support multiple approximation modes with different introduced errors and are able to change the level of introduced error at run-time, based on the weight values of each layer, to offer more fine-grain control. However, the main problem regarding their utilization lies in how to decide which approximation mode of the multiplier will be triggered for the different weight values in order to keep DNN accuracy within specific thresholds.

Observation: There is no *systematic approach to explore different weight-to-approximation mappings* as previous works are based on hybrid methods [5,25] and require constant manual tuning [1,3].

Additionally, state-of-art works rely on the average accuracy of the entire DNN inference phase to evaluate the proposed frameworks. The introduction of error in computations does not affect all the batches of the dataset equally, creating, in many cases, great variations in the achieved accuracy. In modern applications, however, there can be certain quality of service requirements that need to be achieved over the entire dataset [26]. Specifically, in edge devices, the batch size on DNN inference is fairly small (e.g., 16 or even 1) and thus, large consistent drops in accuracy could gravely affect run-time performance.

Observation: Evaluating only on the average accuracy during DNN inference can be misleading.

As mentioned in Section 1.2.1.2, compressing DNNs to smaller model sizes reduces their energy consumption at the cost of accuracy degradation [27]. One commonly followed approach is quantization, where methods quantize DNN weights to lower bit-widths. Each DNN layer has varying sensitivity to lower bit-width quantization and thus, works that quantize all layers to the same bit-width [28, 29] can achieve non-efficient solutions. A way to combat this issue is to apply mixed precision quantization to both the weights and the activations. However, the search space for this problem is vast: for a DNN comprising L layers and 8 available bit-widths (1 to 8), the number of possible combinations is $8^{2\times L}$.

Observation: There is a need to efficiently explore the quantization search space.

Another way to compress a DNN model is pruning, where the non-zero parameters in the DNN are decreased through a variety of ways, deeming its layers sparse. To recover from the pruning-induced accuracy loss, state-of-art works fine-tune the generated models. The combination of both pruning and quantization has also been employed to generate compressed DNNs [30]. It is evident that the search space becomes even more abysmal when compared to only employing quantization as a compression technique. At the same time, all state-of-art works rely on retraining to recover some of the compression-induced accuracy loss.

Observation: There is still a need for an automated DNN compression framework that jointly considers pruning and quantization that does not rely on retraining or fine-tuning.

To address all of the aforementioned observations, we utilize Signal Temporal Logic (STL) and Parametric STL (PSTL), which are specification formalisms used to express the properties of a given system in a compact way. They have been used in testing and verification of cyber-physical systems [31], and we couple the concept of system exploration in the current context of approximate computing to propose energy-efficient automated frameworks.

1.2.3 Resource Management on Heterogeneous Embedded Systems

1.2.3.1 Clustered Chip Multiprocessors

Modern embedded systems have adopted the clustered Chip MultiProcessor (CMP) paradigm clustering together cores of the same type and sharing a last level cache and memory controller. Additionally, Dynamic Voltage and Frequency Scaling (DVFS) mechanisms are used to control and adjust power consumption. However, while this architecture allows for considerable performance gains for modern applications, it presents major challenges in balancing power-performance constraints [32].

Previous approaches try to satisfy power/temperature thresholds, without providing any insights on how they determined these values [9, 33]. However, these values may not be optimal as different applications have different effects on the system. Particularly, Thermal Design Power (TDP) is used as a power threshold during application execution. When applications exceed TDP, the resource manager limits the CPU speed and it can even deactivate specific cores, thus significantly impacting system throughput. However, TDP is not the maximum achievable power that ensures thermal safety. Hence, considering a single and constant power budget for clustered CMPs often leads to pessimistic decisions and performance losses [34]. Similarly, temperature guardbands are used for frequency control when the temperature reaches a critical value. While this mechanism provides a safety net, it greatly affects performance as the operating frequency will be conservatively bounded to the worst-case temperature. Furthermore, for

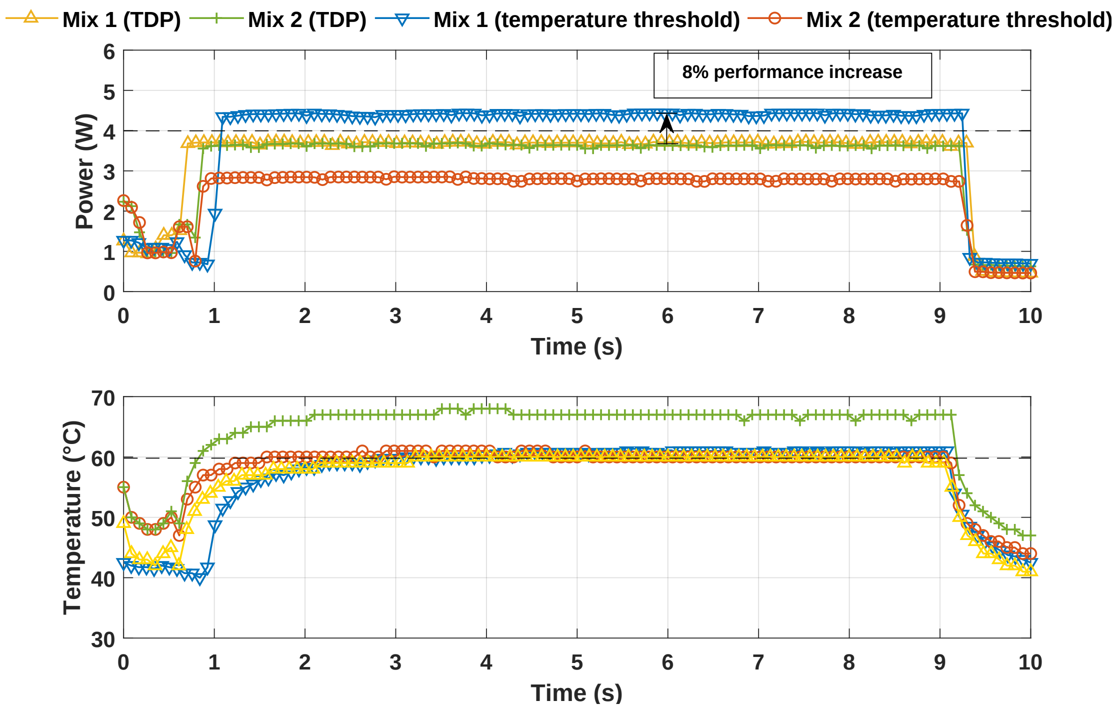

Figure 1.2: Motivational example on TDP thresholds [2]

embedded devices (e.g. mobile phones), due to lack of active cooling, the worst-case temperature is pessimistically estimated without considering the actual properties of the architecture and the workload [35].

Observation: Static TDP thresholds can lead to pessimistic system throughput.

As a motivation, Figure 1.2 presents the power and temperature for two application mixes running on the A15 cluster of the Odroid-XU3 platform. Each mix contains four concurrent executing applications that exert pressure on the memory subsystem in different ways and utilize all cores in the cluster. Additionally, for each mix, we considered two cases: (1) executing under TDP constraint keeping power under $4W$ following [9], and (2) executing under a CPU temperature threshold of $60°C$, which is used in modern mobile devices. While under TDP execution, both mixes satisfy the $4W$ constraint. However, Mix 2 had almost $10°C$ higher temperature than Mix 1. Thus, even though both mixes consumed the same power, the fact that Mix 2 had a greater impact on temperature, requires the TDP to be very pessimistic in order to guarantee thermal safety. In the case of temperature threshold ($60°C$), both mixes satisfy it. However, Mix 2 managed to execute on a higher frequency consuming also more power but without any significant impact on temperature. Additionally, it had an 8% increase in performance compared to the TDP scenario. Concluding, a single and constant power budget for clustered CMPs can lead to pessimistic decisions and performance losses, making necessity an investigation and in-depth analysis of more complex properties.

Observation: Choosing threshold values for thermal constraints is a difficult task that lies in the fact that (i) there is no *systematic approach* to investigate and accelerate an in-depth analysis of more complex properties, and (ii) *automatically* infer the ranges of the constraint parameters.

1.2.3.2 Heterogeneous Internet of Things Computing Systems

In the Internet-of-Things (IoT) domain, DNNs and specifically Convolutional Neural Networks (CNNs) are utilized in a plethora of applications such as smart cars [36] and smart cities [37]. Amongst different kinds of sensors in the IoT era, visual sensors (e.g., cameras) play a key role because they can capture rich and resourceful content, and thus help establish a new era, i.e., Internet of Video Things (IoVT) [38, 39]. Traditionally, IoT devices capture images and they transmit them to remote servers, where deep CNNs are used to process them and extract the required information. When the processing is finished, the server replies back to the IoT device sending the required information. However, this cloud-based model suffers from network overhead due to the vast amount of uploaded data. Moreover, the number of IoT devices increases exponentially and forecasts expect to overpass 25 billion devices by 2025.

Mobile processing, which emerges as an advantageous alternative to cloud-based IoT, brings computation where the data is produced and dictates that all the CNN inference is performed locally. It avoids unnecessary data transmission, alleviates network congestion, and drastically reduces the power that the device consumes on transferring data over the network, saving a significant amount of the power budget for on-device computation. Moreover, IoT devices are becoming more and more computationally capable, as they incorporate multiple CPUs, hardware accelerators such as GPUs, and often specialized NPUs on the same platform [40].

However, there are still strict platform-oriented limitations of resources and the CNN implementation on IoT devices remains a challenging task. The processing required by the CNN execution can result in high inference times, which may significantly affect the quality of user experience and it is unacceptable for latency-sensitive IoT applications. Additionally, the integration of multiple computing components can result in increased power consumption for already power-constrained IoT devices. The tight resource constraints of the IoT devices and the inherent computation complexity of CNNs create a challenging problem: achieve the lowest possible power footprint, while keeping throughput (inference time) acceptable for the end user.

Observation: There is a need to efficiently optimize CNN architectures on a resource constraint IoT device by effectively utilizing the available hardware resources.

1.3 DISSERTATION OUTLINE

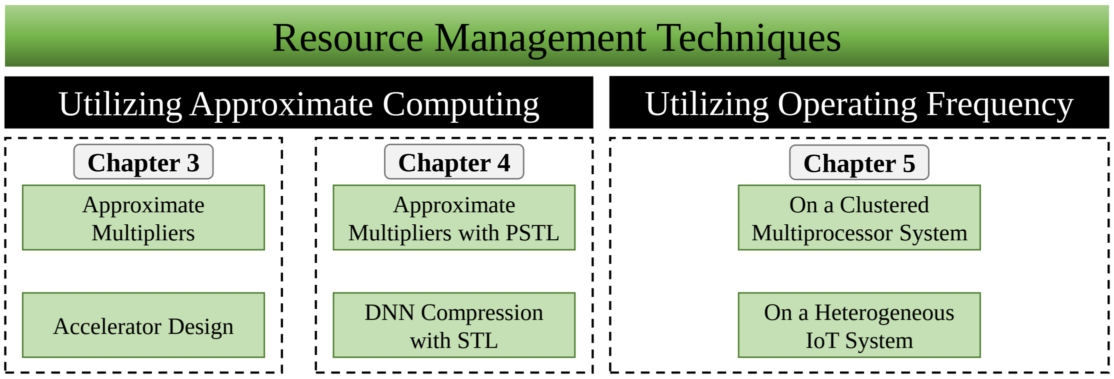
Resource Management Techniques
Utilizing Approximate Computing
Utilizing Operating Frequency
Chapter 3
Approximate Multipliers
Accelerator Design
Chapter 4
Approximate Multipliers with PSTL
DNN Compression with STL
Chapter 5
On a Clustered Multiprocessor System
On a Heterogeneous IoT System

Figure 1.3: The outline of this dissertation.

This dissertation is structured into five chapters following the Introduction chapter, each of which contributes to the field of approximate computing, resource management, and formal property integration for energy-efficient deep neural network (DNN) inference on heterogeneous embedded systems.

- Chapter 2 provides a comprehensive review of the prior art in approximate computing, resource management, and formal property integration for energy-efficient DNN inference. This chapter will explore the state-of-the-art techniques and methodologies used in the field, and identify the research gaps that this dissertation aims to fill.

- Chapter 3 presents the first contribution of this dissertation, which is an exploration of approximate computing techniques for energy-efficient DNN inference. The chapter investigates the trade-offs between accuracy and energy consumption in DNN inference and evaluates the effectiveness of various approximate computing techniques in achieving energy efficiency without sacrificing too much accuracy. Specifically, the focus will be on two different approaches, the proposal of an approximate multiplier and the proposal of a framework that explores approximate hardware designs.

- Chapter 4 builds on the previous chapter by proposing two formal property integration frameworks for approximate computing techniques. The chapter will explore how formal methods can be used to ensure the correctness and robustness of two different approximate computing approaches.

- Chapter 5 presents two resource management frameworks for heterogeneous embedded systems. Inspired by Chapter 4, this chapter will first propose a framework for clustered chip multiprocessors based on formal properties. Afterward, a framework that decides the optimal configuration of a heterogeneous device in terms of the CPU, GPU, and main memory frequency is proposed.

- In the end, Chapter 6 concludes this dissertation by summarizing the contributions of this work and discussing the limitations and future directions of the proposed techniques.

The main contributions of this dissertation are presented in Chapters 3 - 5, as shown in Figure 1.3.

CHAPTER 2
PRIOR ART AND NOVELTY

This section will present existing works in the related research domains outlined by this dissertation. This chapter aims to provide a comprehensive understanding of the state-of-the-art techniques and methodologies used in approximate computing and resource management techniques and identify the research gaps that this dissertation addresses. By analyzing the existing literature, this chapter will also outline the differentiators and novelty of the proposed work.

2.1 APPROXIMATE COMPUTING TECHNIQUES

2.1.1 Approximate Multipliers

Approximate DNN accelerators have attracted vast research interest. A significant fraction of DNN operations (about 90%) is attributed to GEMM operations, i.e., convolution and matrix multiplications [41]. Such computations rely on MAC units and the state-of-the-art approximates the multipliers to boost the energy efficiency of the overall accelerator. Cartesian Genetic Programming is used in [42, 43] to generate fixed approximate multipliers and replace the accurate ones, achieving high hardware gains for a minimal accuracy loss or even accuracy improvement. [21] introduced a multiplier-less artificial neuron that is based on additions only. Nevertheless, [21, 42, 43] require retraining, which as aforementioned is not always feasible.

In [5] the authors avoid retraining and use a layer-based approximation in which a different fixed approximate multiplier from [44] is used per layer. In addition, a weight-tuning method is employed targeting to reduce the introduced multiplication error. The work in [45] extends the approximate multipliers of [44] and shows that, in simple DNNs, high energy gains and minimal accuracy loss can be achieved, even without retraining. However, for more complex DNNs the energy gains are not maintained. Acknowledging the need for runtime reconfiguration, [25] generates approximate multipliers with dynamically reconfigurable accuracy and uses them to apply layer-wise approximation in DNNs by changing the multiplier's accuracy mode per layer. The work in [1] uses [25] to generate low variance approximate reconfigurable multipliers, and proposes a weight-oriented approximation for DNN inference. [46] employs a curable approximation in which the MAC's adder is split into low and high parts and the carry of the low part is accumulated by the neighboring MAC unit. The carry of the last MAC unit is not corrected. However, [46] is evaluated on the LeNet architecture, a very shallow architecture that cannot provide the number of operations recent DNNs do.

The work in [47] uses the Canonic Sign Digit Encoding to represent the weights and employs truncation as an approximation method. The architecture proposed in [48] uses Dynamic and Static Segmented approximate multipliers that support high and low precision for the size of the segment. A trainable input classifier is used to select the required precision per segment. [49] targets energy consumption minimization of MAC-based signal processing algorithms. [49] utilizes different fixed approximate multipliers in an interleaved manner to compensate for errors during the accumulate operations. Nevertheless, [48, 49] consider 16-bit inference and can be deemed inapplicable in modern DNN accelerators that use mainly 8-bit precision [11].

Contributions: (1) We design an approximate multiplier that approximates the generation of the partial products and comprises three operation modes: the Zero Error (ZE) mode (i.e., exact), the Positive Error (PE) mode, and the Negative Error (NE) mode.

(2) By approximating more or less partial products, we control the applied approximation and tune the accuracy-energy reduction trade-off as required.

(3) We present a methodology where for each NN filter, we exploit the two synergistic modes of the presented multiplier, and group all the weight values into two balanced summation sets, with the final goal being the convolution error converging to zero and consequently achieving high inference accuracy.

2.1.2 Approximate Hardware

Section 2.1.1 presented a few ways to integrate heterogeneity in NPUs through MAC units that contain approximate multipliers. The work in [18] introduces a control variate approximation to recover the accuracy loss due to approximate multiplications. The overall convolution error is estimated at run-time and it is finally accumulated in the output result. However, for the error accumulation, it requires an additional column of MAC units. The authors in [50] present an 8-bit quantization method where shift and add operations replace the multiplications between activations and weights, achieving greater speed and lower energy consumption.

Recently, research around the efficient deployment of DNNs on hardware accelerators has increased greatly. The framework in [51] jointly explores the space of neural architecture, hardware implementation, and quantization, but through a compute-intensive, resource- and time-consuming process. The cross-layer framework in [52] explores device, circuit, and architecture design space while considering the device's variation to a couple of robust NNs with efficient hardware design. The work presented in [53], introduces an analytical model that estimates execution time, energy efficiency, and hardware cost of dataflows. The authors in [10] and [54] present a framework for automated design space exploration and layer scheduling for HDAs, by using the cost model in [53]. The framework presented in [55] employs [10, 53] to co-explore DNN architectures and the associated heterogeneous ASIC accelerator design to satisfy target design specifications while maximizing accuracy. SCALE-Sim [56] enables comprehensive design space exploration by presenting a configurable DNN accelerator simulator based on systolic arrays. The work in [57] compares an HDA for analytical query processing, and a homogeneous systolic array alternative. The authors in [58] optimize performance by determining the tensor partition among heterogeneous accelerator arrays.

The work in [59] targets edge devices and proposes a configurable hybrid architecture to support the execution of layers with varying bit-widths. In [60], a Network-on-Chip

(NoC) based DNN accelerator design paradigm is presented, offering reduced off-chip memory accesses and improvements in terms of run-time computational flexibility. The authors in [61] describe efficient mappings on tile-based accelerators, utilizing shared buffers, loop ordering, and application pipelining. Another heterogeneous tile-based architecture with a hierarchical interconnect is presented in [62]. The work in [63] introduces the Flexible Dot Product Engine units, where the NoC connecting these units together resembles tiled accelerator architectures like the two aforementioned works. Towards energy-efficient DNN inference, the work in [64] proposes an inference engine for compressed NNs, while the work in [65] proposes a processing-in-memory architecture that processes binary convolutional NNs in mobile devices.

Many works in the recent literature focus on mixed precision and multi-DNN workloads. The work in [27] utilizes reinforcement learning (RL) to determine fine-grain quantization within a given NN, which is inapplicable at run-time. The work in [66] proposes a mixed-precision quantization method, where the quantization precision is based on each layer's Hessian spectrum. The work in [67] supports mixed precision for NN quantization and conducts an evolutionary architecture search for an optimal design. [68] proposes a quantization method that utilizes mixed precision for different NN layers. The work in [69] explores both the neural architecture space and the quantization space to decrease DNN latency without accuracy degradation. [70] searches an energy-efficient neural architecture by adopting the MobileNetV2 block as the base search unit. The work in [71] proposes an accelerator architecture that is matching compute- and memory-intensive tasks from different networks to execute them in parallel.

Contributions: We combine the concepts of quantization, system design, and scheduling and:

(1) we do not build fully custom HDAs targeting a single or a family of NNs, rather we introduce HDA heterogeneity with respect to computational precision by utilizing varying bit-width accuracy NPUs to build HDAs, and employing a post-training

quantization method that allows fast run-time NN quantization and execution with negligible accuracy loss for a wide variety of NNs;

(2) we present a bottom-up design methodology that considers the sensitivity of multi-DNN workloads, while also utilizing an efficient NN-to-NPU scheduling algorithm that maps NNs to NPUs targeting the reduction of response time; and

(3) we showcase the impact of the proposed HDAs towards the improvement of the response time and energy efficiency of multi-DNN workloads under different arrival conditions.

2.1.3 Integrating Formal Properties in Approximate Computing Techniques

The two previous subsections overviewed the heavy utilization of approximate multipliers [72] and approximate hardware designs [27] for efficient DNN inference. However, most of these approaches either require retraining, time-consuming heuristic methodologies, or further fine-tuning mechanisms. This dissertation will present two different frameworks that utilize Signal Temporal Logic (STL) and Parametric Signal Temporal Logic (PSTL) to conduct an automated systematic search for approximation configurations that satisfy fine-grain accuracy properties.

As already mentioned in Section 1.2.1.1, when it comes to approximate multipliers and their mapping to DNN weights during inference, the difficulty lies in the fact that there is no *systematic approach to exploring different weight-to-approximation mappings* as previous works are based on hybrid methods [5, 25] and require constant manual tuning [1, 3]. To create more efficient mappings, while automating and enabling the systematic exploration of the introduced error, we can express the properties of the approximate accelerators using Signal Temporal Logic (STL) [73], a specification formalism to express system properties. Once an STL formula is expressed, a robustness analysis [74] is used to evaluate in which cases the properties hold true. This way, we can check the system robustness under different STL expressions which allows systematic exploration, but is not scalable as each expression captures a small part of the exploration. Parametric Signal Temporal Logic (PSTL) [75] extends STL by replacing threshold constants with parameters that get to be inferred.

Contributions: We present a unified framework that utilizes PSTL to express accuracy constraints and find weight-to-approximation mappings for the weights of a DNN on approximate multipliers so that the constraints are satisfied and the energy consumption is minimized. Specifically:

(1) we propose a novel framework which, based on appropriate constructed formal expressions and input formulation, utilizes robustness metrics to achieve energy-efficient approximation mappings;

(2) we systematically explore the properties of approximate accelerators when multipliers with different introduced errors are employed, in correlation with their utilization under specific accuracy thresholds;

(3) we show that although the state-of-the-art satisfy tight but coarse constraints, it fails to satisfy more fine-grain ones. To the best of our knowledge, this is the first work that allows fine-grain exploration of approximate accelerator error properties under multiple accuracy constraints, while avoiding manual tuning and retraining or weight tuning; and

(4) we evaluate the resulting mapping combinations against state-of-the-art mapping methodologies and compare our energy gain findings when considering both fine- and coarse-grain accuracy requirements.

Regarding efficient DNN inference, Section 2.1.2 mentioned quantization as a way to represent DNNs in lower precisions to achieve energy gains at the cost of accuracy degradation [27]. Another way to compress a DNN model is pruning, where the non-zero parameters in the DNN are decreased through a variety of ways, deeming its layers sparse. The work in [76] zeroes weights whose magnitude exceeds a specified threshold, but another approach is to remove entire channels (filters/neurons) [77]. The work in [78] introduced depth-wise and shape-wise sparsity in DNN pruning, where instead of pruning individual weights, groups of weights are pruned instead. The work in [79] employs RL to prune convolution channels. To recover from the pruning-induced accuracy loss, these works fine-tune the compressed generated models

The combination of both pruning and quantization has also been employed to generate compressed DNNs. It is evident that the search space becomes even more abysmal when compared to only employing quantization as a compression technique. The work in [80] performs training jointly with pruning and quantization through Bayesian optimization. However, this work alongside others uses human heuristics [81, 82], therefore another issue that arises is how to automatically generate compressed DNNs. The work in [83] jointly quantizes and prunes DNNs by updating the weight parameters with unified channel-wise pruning, while the work in [30] jointly compresses DNNs through the alternating direction method of multipliers. Both of these works propose automated frameworks, however, they still employ retraining/fine-tuning to recover from the compression-induced accuracy loss [81]. Retraining is a time-consuming process that might not even be feasible [4].

Contributions: We utilize Signal Temporal Logic (STL) to conduct an automated systematic search for compression configurations that satisfy fine-grain accuracy properties. The proposed framework generates energy-efficient compressed DNNs using both pruning and quantization. The contributions of this work are many-fold:

(1) we formalize accuracy properties of DNNs through STL and formulate fine-grain accuracy requirements,

(2) the proposed framework utilizes robustness metrics to guide the optimization procedure that generates compressed DNNs,

(3) the optimization phase explores the joint pruning-quantization search space based on formal properties,

(4) we enable the run-time control of the quality of DNN inference in terms of both

accuracy and energy consumption,

(5) we generate compressed DNNs without any additional fine-tuning or retraining, and

(6) we evaluate our work on different hardware architectures to show its adaptability.

2.2 RESOURCE MANAGEMENT IN HETEROGENEOUS EMBEDDED SYSTEMS

2.2.1 Clustered Chip Multiprocessors

Resource management of modern CMPs is a well-studied topic due to its importance and to the increased number of actuation knobs that modern platforms offer. Previous research works can be split into two categories. In the first one, single objective optimization techniques have been developed. Particularly, such approaches developed run-time resource managers that focus only on one metric such as application performance/system throughput [84, 85], energy consumption minimization [86, 87], or resource utilization [88]. In order to satisfy the objective, the aforementioned approaches search the configuration space and create policies for the run-time managers. To overcome the exploration time bottleneck, there have been efforts to accelerate the process by developing heuristics [89, 90]. Alternatively, there have also been efforts to develop multi-objective approaches that try to optimize both execution time and energy/power consumption [91–93]. Since these objectives are often conflicting, a variation of multi-objective optimization is to rank the objectives and assign priorities and goals to specific metrics [94]. In that way, the run-time manager monitors the different goals and utilizes the available actuators in order to enforce resource allocation policy to satisfy the hierarchy [9]. However, all the aforementioned resource allocators and controllers work considering fixed power, temperature, and performance constraints which are mostly extracted experimentally. Specifically, there is no justification for why meeting a power threshold will always keep the temperature within safe values since applications utilize on-chip resources in a different way. Furthermore, keeping a low power threshold will always satisfy temperature constraints; however, it may not allow the resource allocators to fully exploit the computing resources of the platform.

Contributions: Instead of designing resource managers under fixed and unexplored constraints, we present an approach to infer the ranges of parameters for specifications under different types of workloads. Specifically:

(1) we show how the designer can express any specification and automatically infer the ranges of the parameters for clustered CMPs,

(2) we go beyond traditional design space exploration by utilizing robustness metrics to explore and determine system properties,

(3) we propose a way to systematically infer the constraint parameters of clustered CMPs in order to satisfy multiple distinct objectives (system, application, and throughput),

(4) we demonstrate how the inferred parameters can be used in order to enhance run-time resource allocation decisions, and

(5) we evaluate the whole framework on Odroid-XU3 board [95] over varying workload conditions.

2.2.2 Heterogeneous Internet of Things Computing Systems

Research groups that focus on the efficient implementation of CNNs on embedded IoT devices utilize heterogeneous hardware in order to achieve power efficiency. Authors in [96] explore the trade-offs between power consumption and execution time of a given CNN architecture. They target a CPU-based embedded platform and they manage to reduce energy consumption by up to 8.9%. However, they exhaustively explore the design space, which is impractical in our case, as the number of possible combinations is significantly larger. Authors in [97] propose a coordinated run-time mechanism that trains a CNN and dynamically estimates the power characteristics of a heterogeneous IoT platform, They perform a local search that estimates the efficiency of the candidate system states with an average error of 7.7%. However, they focus on the training phase rather than the inference.

On the other hand, authors in [98] consider a given CNN architecture and re-shape it with respect to the width and the length of the network in order to produce an energy-efficient alternative network. The difference compared to our work lies in the context of the targeted problem: we utilize CNNs that have different shapes in order to evaluate the proposed framework, while we explore different frequency combinations of the targeted hardware platform. In [98], these parameters are not explored, as they are out of the scope of the examined problem. Authors in [99] perform hyper-parameter optimization when training a CNN architecture by taking into account the power limitations of the target IoT device. However, our approach differs since (1) we target the CNN inference and not the training phase and (2) our approach considers the CNN architecture as *given*, so it is a known factor in our problem.

Other works explore device collaboration in IoT networks, in order to increase power efficiency. Authors in [100] propose a mobile-cloud collaborative approach and they target IoT devices with a limited power budget. They utilize the same Jetson TX2 development board and they minimize the energy consumption of a given CNN by offloading computation from the IoT device to the cloud server. They are able to achieve a 2$\times$ on average and up to 4$\times$ reduction in mobile energy consumption. Authors in [101] propose a computation partitioning mechanism based on a given CNN architecture and they partition the CNN computation between mobile IoT devices and the cloud. They decide which CNN layers to compute locally and they offload the others on a remote server. They are able to decrease the device energy consumption by 59.5% on average and up to 94.7%. Although promising, these approaches require multiple IoT devices that communicate with each other through an orchestration mechanism.

Contributions: We propose a framework that efficiently deploys an existing CNN on low-power GPU-based architectures that are suitable for computer vision applications. The contributions can be summarized by the following:

(1) The development of a framework for efficient deployment of an existing CNN architecture on the Nvidia Jetson TX2 board under three different modes of operation,

(2) a flexible and scalable CNN formulation and representation suitable to cover a great variety of CNN architectures, and

(3) an efficient design space exploration for optimization of the CNN execution on the given device for different CPU, GPU, and memory frequencies. The design space exploration takes advantage of the underlying hardware resources and evaluates trade-offs between power consumption and inference time.

CHAPTER 3

APPROXIMATE COMPUTING TECHNIQUES FOR DNN INFERENCE

This chapter will present in detail two different approximation techniques and their respective evaluation. First, the design and accompanying mapping methodology for an approximate multiplier will be presented. Second, a framework on hardware DNN accelerator design and DNN scheduling will be presented.

3.1 POSITIVE/NEGATIVE APPROXIMATE MULTIPLIERS FOR DNN ACCELERATORS

In the work described in this section, we target DNN inference and apply approximation through the utilization of a proposed approximate multiplier to maximize energy gains without significant losses in accuracy. Particularly, we present a reconfigurable multiplier, that follows a different approach than state-of-the-art, by comprising one exact mode of operation and two synergistic approximate modes, one positive and one negative, that aim to balance the introduced error. Additionally, we present an approximation strategy that assigns NN operations to specific approximation modes based on the respective layer, filter, and weight value of the operation. The proposed approach doesn't require retraining and aims to minimize the convolution error by introducing, in a directed manner, positive and negative approximation in the performed multiplications.

3.1.1 Methodology

An overview of our approach is depicted in Figure 3.1. First, we will present our positive/negative approximate multiplier and show a rigorous error analysis that is exploited in the error optimization of our mapping methodology. Then, given a trained DNN, we quantize weights and activations to 8-bit (in the range [0, 255] [28]) and we apply our mapping methodology, responsible for assigning the weights to the modes of

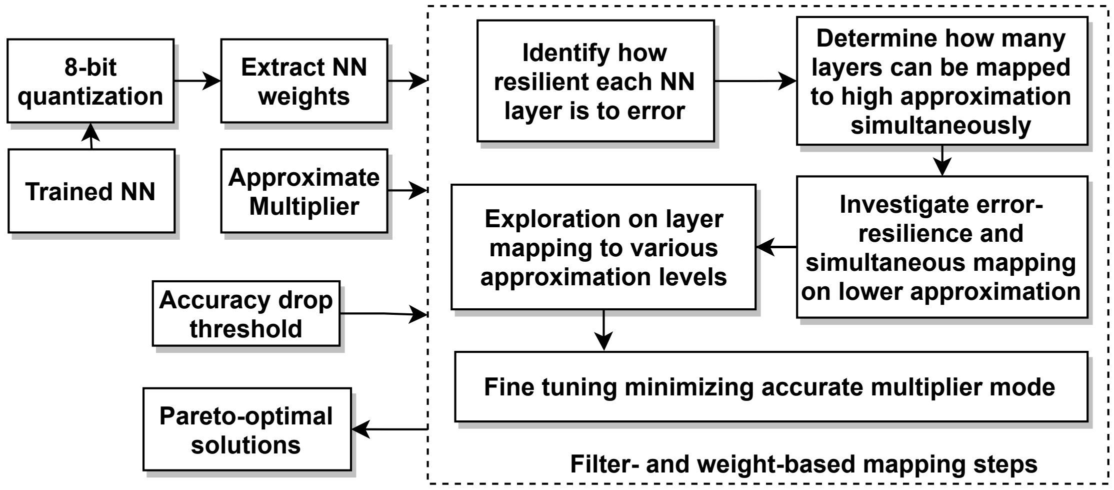
8-bit quantization
Trained NN
Extract NN weights
Approximate Multiplier
Accuracy drop threshold
Pareto-optimal solutions
Identify how resilient each NN layer is to error
Determine how many layers can be mapped to high approximation simultaneously
Investigate error-resilience and simultaneous mapping on lower approximation
Exploration on layer mapping to various approximation levels
Fine tuning minimizing accurate multiplier mode
Filter- and weight-based mapping steps

Figure 3.1: An overview of the proposed methodology [3]

our multiplier.

3.1.1.1 Positive/Negative Approximate Multiplier in NNs

The most complex computation in the CNN inference phase is the convolution operation. The latter is expressed as:

$$G = B + \sum_{i=1}^{k} W_i \cdot A_i, \tag{3.1}$$

where W_i are the weights, A_i are the input activations, and B is the bias of the neuron. We assume a microarchitecture similar to Google TPU that comprises a big systolic MAC array [11, 102]. In addition, we consider a weight-stationary mapping and we replace the exact multipliers with approximate ones. Denoting ϵ_i the error of the approximate multiplication:

$$\epsilon_i = W_i \cdot A_i - W_i \cdot A_i|_{approx} \tag{3.2}$$

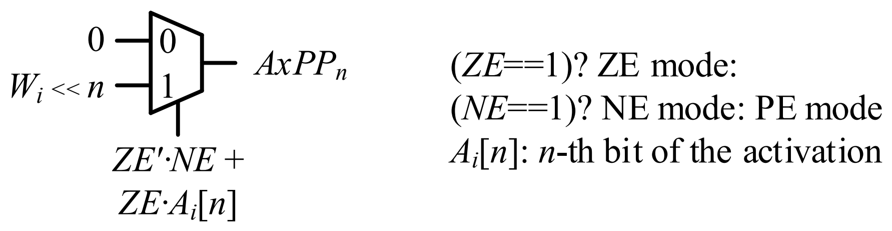

Figure 3.2: Run-time approximate generation of the n-th partial product [3].

the error ϵ_G of the approximate convolution is given by:

$$\epsilon_G = G - G|_{approx} = \sum_{i=1}^{k} \epsilon_i. \tag{3.3}$$

In [103], the authors proposed an approximate multiplier with predictable (known a priori) error. The multiplier of [103] eliminates the generation of some partial products (they are set to zero) and thus fewer partial products need to be accumulated. The technique in [103] always leads to positive error as the approximate product is always smaller than the exact one. Consequently, when approximating (eliminating) the z least partial products, the error e_i is given by:

$$\begin{aligned} \epsilon_i &= W_i \cdot A_i - W_i \cdot (A_i - A_i \bmod 2^z) \\ &= W_i \cdot r_i, \text{ with } r_i = A_i \bmod 2^z. \end{aligned} \tag{3.4}$$

Hence, the average multiplication error and the error variance $\forall A_i$ of [103] are given by:

$$\begin{aligned} \mathrm{E}[\epsilon_i] &= \frac{2^z - 1}{2} W_i \\ \mathrm{Var}(\epsilon_i) &= \frac{2^{2z} - 1}{12} W_i. \end{aligned} \tag{3.5}$$

The authors in [104] proposed to use switches and control, at run-time, the number of partial products that will be approximated (i.e., set the z value at run-time). Hence, [104] supports also exact multiplications (i.e., when $z = 0$). In our work, we extend [104] to support three different modes: *Zero Error (ZE), Positive Error (PE), and Negative Error (NE)*. The ZE mode refers to the exact operation, in which no error is introduced in the multiplication. In the PE mode, the z least partial products are perforated and thus positive error is obtained. In the NE mode, we force the generation and accumulation of the z least partial products and thus negative error is obtained. Fig. 3.2 presents how the three operating modes (ZE, PE, and NE) can be configured at run-time. Considering the weight-stationary mapping, in both NE and PE modes, the z partial products remain fixed at run-time (for several cycles) leading to reduced switching activity and thus high power gains.

Since, in the NE mode, we force the generation of the partial products, the multiplication error ϵ_i is given by:

$$\begin{aligned}\epsilon_i &= W_i \cdot A_i - W_i \cdot \big((A_i + (2^z - 1 - A_i \bmod 2^z)\big) \\ &= -W_i \cdot (2^z - r_i - 1), \text{ with } r_i = A_i \bmod 2^z.\end{aligned} \tag{3.6}$$

Thus, in the NE mode, the average multiplication error and the error variance $\forall A_i$ are given by:

$$\begin{aligned}\mathrm{E}[\epsilon_i] &= -\frac{2^z - 1}{2} W_i \\ \mathrm{Var}(\epsilon_i) &= \frac{2^{2z} - 1}{12} W_i.\end{aligned} \tag{3.7}$$

As a result, from (3.5) and (3.7), the average error and the error variance, $\forall A_i$, of our approximate multiplier are given by:

$$\begin{aligned} \mathrm{E}[\epsilon_i] &= s\frac{2^z - 1}{2}W_i \\ \mathrm{Var}(\epsilon_i) &= \frac{2^{2z} - 1}{12}W_i, \end{aligned} \tag{3.8}$$

where $s = 1$ in the PE mode, $s = -1$ in the NE mode, and $z = 0$ in the ZE mode.

Without any loss of generality, each multiplier in the DNN accelerator can be configured to a different mode, i.e., each multiplier features different s and z values, named s_i and z_i respectively. Therefore, using (3.3) and (3.8), the average convolution error $\mathrm{E}[\epsilon_G]$ is given by:

$$\mathrm{E}[\epsilon_G] = \sum_{i=1}^{k} \mathrm{E}[\epsilon_i] = \sum_{i=1}^{k} s_i\frac{2^{z_i} - 1}{2}W_i \tag{3.9}$$

and the convolution error variance $\mathrm{Var}(\epsilon_G)$ is given by:

$$\begin{aligned} \mathrm{Var}(\epsilon_G) &= \sum_{i=1}^{k} \mathrm{Var}(\epsilon_i) + \sum_{1\leq i\leq j\leq k} \mathrm{Cov}(\epsilon_i, \epsilon_j) \\ &= \sum_{i=1}^{k} \frac{2^{2z_i} - 1}{12}W_i + \sum_{1\leq i\leq j\leq k} W_i W_j 0 \mathrm{Cov}(r_i, r_j) \\ &= \sum_{i=1}^{k} \frac{2^{2z_i} - 1}{12}W_i. \end{aligned} \tag{3.10}$$

in which r_i and r_j are uncorrelated and thus their covariance $\mathrm{Cov}(r_i, r_j)$ is zero.

Exploiting that (3.9) and (3.10) depend only on the weights and leveraging the fact that the weight values are obtained after training and quantization, we can minimize the convolution error (i.e., minimize (3.9) and (3.10)) by carefully setting the approximation parameters of each multiplier (i.e., s_i and z_i).

Finally, Table 3.1 shows the achieved energy reduction of the PE and NE modes for different z values. As can be seen, the energy gains increase as the value of z increases.

Table 3.1: Energy gains of our 8-bit Positive/Negative Approximate Multiplier

Mode	$z = 1$	$z = 2$	$z = 3$
PE	8.3%	20.23%	36.6%
NE	5.5%	16.17%	31.8%

However, the magnitude of the multiplication error, both in PE and in NE, becomes larger as well, as calculated by (3.8). Therefore, in the next Section, we present a method to map the weights to specific modes in order to keep the overall inference accuracy loss low.

3.1.1.2 Filter- and weight-based mapping

In this section, we present a filter-oriented method for mapping the weights of an NN to the three aforementioned modes of the approximate multiplier as well as deciding the value of z for each one of them. For our analysis, the available values for z are 1, 2, and 3. We also tested values greater than 3, but the introduced error was very large and violated our tight accuracy thresholds. Our five-step mapping procedure aims to satisfy a given accuracy drop threshold while maximizing the number of weights that are assigned to high z values. Fig. 3.3 depicts an illustrative example of the steps.

Step 1 - Layer resilience: The goal of this step is to identify how error resilient each convolution layer of the targeted NN is (Fig 3.3 ①). Initially, we consider that all weights are assigned to the exact mode (i.e., ZE mode). Then, for each layer of the NN separately, starting from the first one, we count the occurrences of each weight value *per filter*. We define $w_{i,f,l}$ as the number of times that weight i occurs in layer l of filter f. We take advantage of the positive/negative architecture of the proposed multiplier and we assign the $w_{i,f,l}/2$ weights to the PE mode and the rest half of the weights to the NE mode in order to cancel out the introduced error (see (3.9)). If $w_{i,f,l}\%2 \neq 0$, we map $\lfloor w_{i,f,l}/2 \rfloor$ times the weight i to the PE and NE modes and the remaining occurrence of i to the ZE mode, keeping it also in a residue list, unique for each filter, to be used in the last step for further tuning. We call this concept *filter-oriented error balancing.* For this

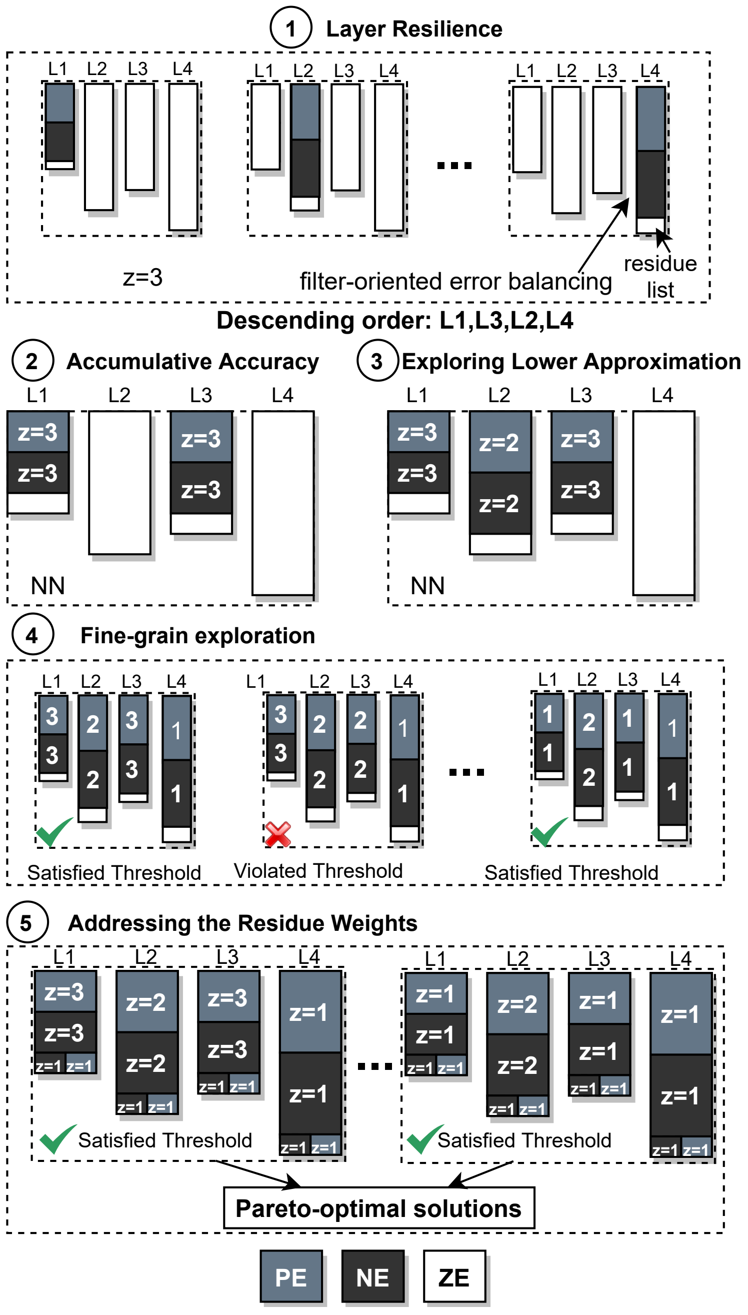

Figure 3.3: Illustrative example of the proposed five-step methodology [3].

particular step, we set $z = 3$ (for all weights) for the PE and NE modes, as it introduces the highest error compared to $z = 2$ and $z = 1$, achieving higher energy gains. Since the procedure described above is performed for each layer separately, we record the accuracy output of the network each time and we determine which layers are more sensitive to approximate multiplications, and which layers show small or no drop in the final network accuracy. Once we have obtained the layer resilience information, the output of this step is a list of convolution layers of the network sorted based on error resiliency (i.e., highest accuracy to lowest inference accuracy). At this point, although the weights assigned to the PE and NE modes (positive and negative error) are balanced, the convolution error is still defined by its variance, as (3.10) shows. The latter depends exponentially on z. Hence, this step sorts the layers with respect to z tolerance.

Step 2 - Accumulative inference accuracy: In this step, our goal is to find how many layers can be mapped to high approximation (high z) simultaneously, using the filter-oriented error balancing method presented in Step 1 (Fig 3.3 ②). Thus, starting from the most error-resilient layer of the network towards the least resilient one, we map the convolution layers to the PE and NE modes following the procedure described previously, but this time in an accumulative way, still using $z = 3$. We stop this step once we have reached the accuracy drop threshold.

Step 3 - Exploring lower approximation: In this step, we repeat the procedure that was described in Steps 1 and 2, however in this case we set $z = 2$ (Fig 3.3 ③). However, we do not perform the layer resilience and accumulative accuracy process to all convolution layers of the network, but only to the remaining ones out of the procedure described in Step 2. When this step is finished, in most cases we are left with a portion of the network's convolution layers mapped to PE/NE using $z = 3$, some mapped using $z = 2$, and the remainder of the layers mapped to ZE.

Step 4 - Fine-grain exploration: Up to this point, the actions described in Steps 1-3 let us reduce the search space. Particularly, we explored the error resilience of the NN layers for $z = 3$ and $z = 2$ but there are still layers entirely mapped to the ZE mode. These layers, which cannot tolerate approximation for $z = 3$ and $z = 2$, will be mapped to the PE/NE modes with $z = 1$. However, this new mapping to $z = 1$ can severely impact NN accuracy, violating the accuracy threshold. Hence, in this step we perform a fine-grain exploration for different mapping combinations among the different z values in order to balance this newly introduced error (Fig 3.3 ④). Additionally, this step lets us perform a more thorough search for more valid mappings and let us create a Pareto-front. The exploration is performed in three parts. First, we start moving one by one all the layers mapped to $z = 3$ to $z = 2$, starting from the layer that was the last one to be mapped to $z = 3$ and we keep the solutions that satisfy the accuracy drop threshold requirement. Second, we follow the same concept for the layers mapped to approximate modes with $z = 2$, this time moving them to $z = 1$, while still keeping each mapping combination that satisfies the accuracy drop threshold. This part is a step towards maximizing energy savings since all the layers mapped to $z = 3$ remain this way, but the layers mapped to $z = 2$ are being moved to $z = 1$ in order to reduce the accuracy drop. Finally, all layers initially mapped to PE/NE with $z = 3$ are moved to $z = 1$ approximate modes, while keeping each mapping combination along the way that does not violate the accuracy threshold requirement. This is another way to drastically reduce the introduced error, as the approximation under $z = 3$ is more aggressive, mostly relying on the layers mapped to $z = 2$ for energy gains. Overall, the output of this step is a list of valid mappings, with varying energy savings, utilized in Step 5 for final tuning.

Step 5 - Addressing the residue weights: So far, in all previous steps, the weights included in the residue lists of each filter, described in Step 1, are being mapped to the ZE mode. Thus, in this step, we use all the mapping configurations found so far, that satisfy the accuracy threshold, and we map all these residue weights to either the PE or NE mode (Fig 3.3 ⑤). Hence, for each filter, we partition the residue list into *two balanced summation sets* using the Largest Differencing Method [105] (LDM) algorithm. Then, we map all weight values in the first set to the PE mode, and the weight values in the second set to the NE mode. Again, in this step we keep all the solutions that satisfy the accuracy requirement. For all the solutions, the residue weights will be mapped to approximate modes starting with $z = 1$, then $z = 2$, and finally $z = 3$, in an attempt to push the approximation for better energy results.

Overall, targeting high energy gains, our mapping methodology aims in assigning each weight to either PE or NE with a high z value (see Table 3.1). Steps 1-4 perform an exploration in which entire layers are approximated (see mapping procedure in Step 1) using a greedy procedure that tries to find the highest z value per layer. After Step 4, the focus is given on the residue weights, which up to that point are mapped to ZE. Considering (3.9) and that up to now the positive and negative error weights are completely balanced, the average convolution error in steps 1-4 is zero. Therefore, only the convolution error variance (3.10) affects the accuracy. Finally, in step 5 we focus on assigning residue weights to non-ZE modes (i.e., boost further the energy gains). Note that, LDM aims to create subsets whose sums are as equal as possible, but it does not guarantee a balanced final partitioning. Thus, after step 5, (3.9) is close to zero (as we discuss later) but might not be actual zero. For this reason also, applying LDM from the beginning (Step 1) would lead to sub-optimal solutions. Using LDM for all filter weights would result in a biased error (non zero (3.9)) and thus, during the z optimization both (3.9) and (3.10) would contribute to the accuracy loss, resulting in smaller z values per layer and/or more complex z allocation procedure.

Considering (3.9), the efficiency of the error balancing (i.e., how close (3.9) will be to zero) depends on the weight values. Weight values close to each other increase the probability of error cancellation when employing our positive/negative approximation. Fig. 3.4 shows the weight value distributions for two different NNs: GoogleNet [106] and ResNet20 [107] on the CIFAR-10 dataset. As shown, for both NNs, the weight distributions are close to normal and weight values feature low dispersion. Finally, setting the mode of operation is seamlessly performed at run-time as the mapping decision is stored with the weights values (i.e., 3 bits per weight to encode z, ZE, and NE). As described in [47], targeting recent batch processing DNN accelerators, the storage requirements for similar methods are low since the required memory space is averaged

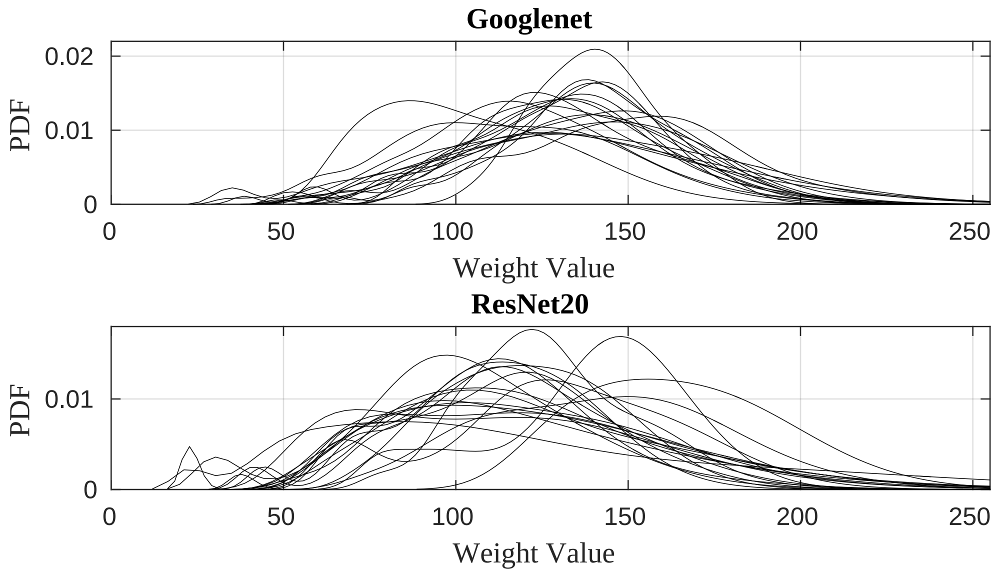

Figure 3.4: Distributions of weight values under 8-bit quantization [3]

over the entire batch.

3.1.2 Results & Evaluation

In this section, we provide the experimental evaluation of our proposed method in terms of energy savings and accuracy loss. As MAC operations consume a very significant portion of total energy cost, we evaluate the energy reduction w.r.t. the MAC operations. Note that MAC units are the basic building block of any DNN accelerator. Additionally, we present comparative results against a variety of state-of-the-art techniques. For the accuracy evaluation we consider seven DNNs of varying size and characteristics: ResNet20 [107], ResNet32 [107], ResNet44 [107], ResNet56 [107], MobileNetv2 [108], GoogleNet [106], and ShuffleNet [109]. The DNNs were trained on four different datasets: CIFAR-10 [110], CIFAR-100 [110], GTSRB [111] and LISA [112]. Overall, 28 models are considered in our analysis. In all experiments, 8-bit post-training quantization is used [28].

To evaluate our method, we conducted evaluation experiments with other state-of-the-art methods that employ approximate computing techniques, such as fixed approximation across all layers of an NN, or similar fine-grain weight-based approximation mapping. Specifically, we chose the following methods for evaluation comparison:

Exact: This method uses exact 8-bit multipliers and is therefore the baseline for our experiments.

ALWANN [5]: This method utilizes approximate multipliers from the library in [44] and employs weight-tuning to minimize the error that the approximate multiplications incur. Note that all multipliers used in this method are fixed and do not comprise different modes of operation. Additionally, this method utilizes non-uniform approximate architectures across the network (i.e., a different approximate multiplier per layer) eliminating flexibility and applicability to other networks and datasets when implemented in hardware. For this reason and for fair comparisons, we consider a homogeneous architecture for [5]. In our evaluation, for each use case, we considered all of the Pareto-optimal approximate multipliers described in [44], as different NN might require different approximate multipliers from [44] to satisfy the accuracy loss threshold.

LVRM [1]: This is a more fine-grain weight mapping approach that employs a low-variance reconfigurable multiplier and additionally applies a constant error correction term by modifying the biases of the filters.

ConVar [18]: This work uses fixed approximation enhanced with a run-time error correction method. [18] induces high approximation at the multiplier level to achieve high energy gains and relies on error correction to achieve high accuracy at convolution and consequently inference levels.

Filter Balanced Sets (FBS): In this method, we use the proposed positive/negative multiplier and we employ the concept of LDM on all the weights of all the layers to create two balanced summation sets per filter, instead of applying this concept on just the residue weights as we do in our methodology. By comparing with this method, we want to showcase that just creating balanced sets per layer (from step 1) leads to a biased error and suboptimal results. For a fair comparison, we tried all z combinations and selected the one that satisfies the accuracy thresholds and yields the highest energy gains.

All the aforementioned methods do not require retraining as our methodology. In addition, they enable us to evaluate our work against the state of the art, i.e., fixed approximation with statistical error correction [5] and [18], but also against more fine-grain run-time reconfigurable approximation [1].

As we target high accuracy, we consider the following accuracy drop thresholds: 0.5%, 0.75%, and 1%. All the aforementioned NNs are trained on each dataset described above. Specifically, all NNs are trained using the Tensorflow machine learning library [113] and are then frozen and quantized to 8-bit. The accuracy evaluations are performed by describing in `C` all the approximate multipliers and using the approximate extension of Tensorflow proposed in [114]. Accuracy loss is calculated w.r.t. the accuracy achieved by the 8-bit quantized model with exact multiplications. Regarding the energy gains, we describe all the examined MAC units in Verilog RTL, and industry-strength tools are used for the hardware analysis. All the MAC units are synthesized using Synopsys Design Compiler and are mapped to a 14nm technology library calibrated with Intel data [14]. The `compile_ultra` command is used for synthesis targeting the maximum frequency that the exact design achieves. We run post-synthesis timing simulations using Mentor Questasim and 1 million randomly generated inputs to capture the switching activity of the MAC units. The switching activity is fed to Synopsys PrimeTime to calculate the power consumption. In each MAC unit, we replace the multiplier with the respective approximate one. To be in compliance with [5] and [1], in order to implement our approximate multiplier we used the exact multiplier (1JFF) from [44] as the baseline. Similarly, 1JFF is used in the exact MAC unit.

For each of the 4 considered datasets, Fig.3.5 - 3.8 show the respective results for all NNs and all three different accuracy thresholds 0.5%, 0.75% and 1%. Specifically, Fig. 3.5 - 3.8 report the energy reduction achieved by our method and the state of the art. Energy reduction is calculated w.r.t. the energy consumption of the exact design.

Our methodology achieves overall higher energy gains when compared to the

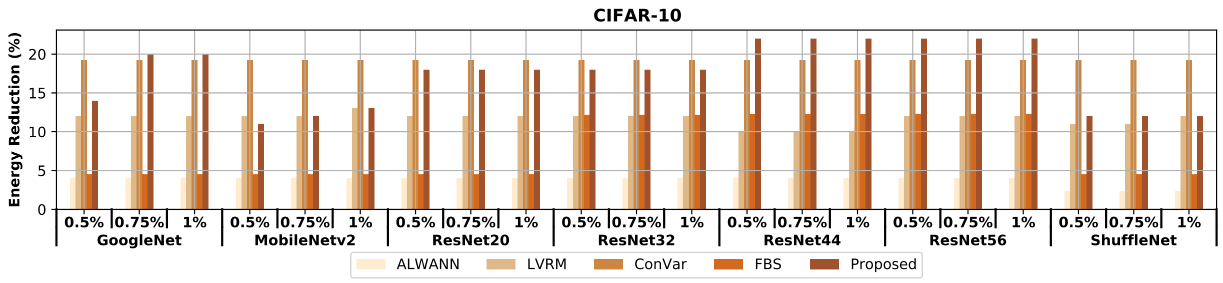

Figure 3.5: CIFAR-10 normalized MAC operation energy savings

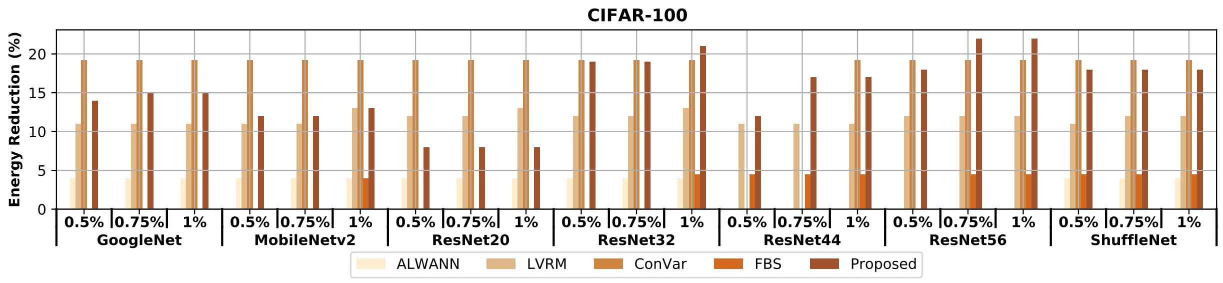

Figure 3.6: CIFAR-100 normalized MAC operation energy savings

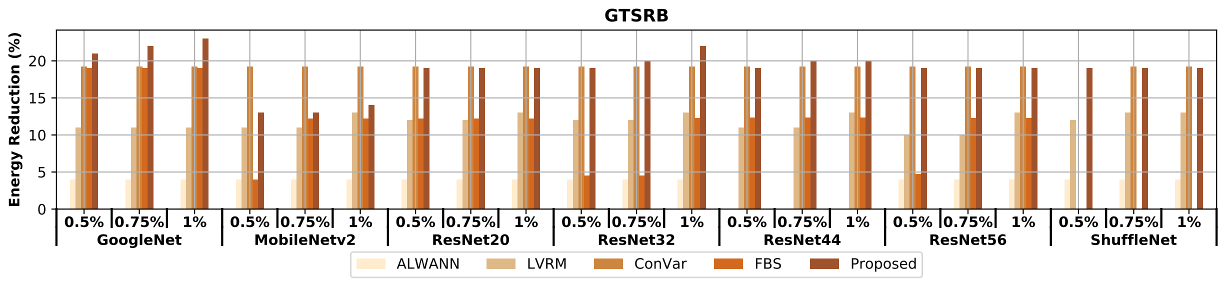

Figure 3.7: GTSRB normalized MAC operation energy savings

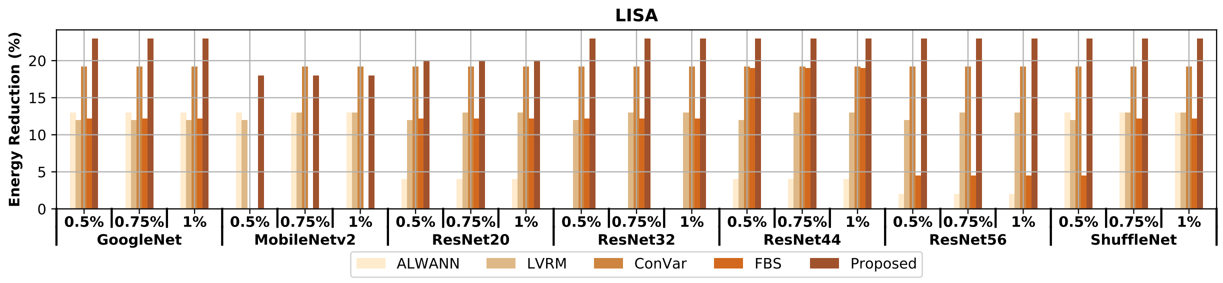

Figure 3.8: LISA normalized MAC operation energy savings

corresponding reconfigurable weight-oriented method presented in LVRM [1], surpassing it in some cases by as much as 12%. On average, our method achieved 18.33%, while

ConVar [18] yielded 18.29%, LVRM [1] 11.9%, FBS 7.46% and ALWANN [5] 4.27% in terms of energy savings. ConVar [18] was the method that reached and maintained energy savings similar to our proposed method's, in some cases surpassing our results by up to 11% (ResNet20 for CIFAR-100). However, ConVar [18] failed *repeatedly* to satisfy the given accuracy thresholds as opposed to our technique that always satisfied the accuracy constraints. Particularly, ConVar [18] reports an accuracy loss that exceeds the 0.5% mark, especially for the CIFAR-100 dataset. By pushing the approximation more we were able to counteract smaller energy gains in some cases with greater gains in others, reaching average energy savings similar to ConVar [18]. FBS also failed to produce acceptable solutions on multiple occasions, justifying our choice to only employ LDM on smaller sets of weight values in the final step of our mapping methodology.

The execution time of the proposed methodology was similar to the one reported in LVRM [1], where the largest execution time was observed for ResNet-56 which did not surpass the two hour mark. Each of the steps that constitute the proposed methodology can be divided into multiple threads and executed in parallel on multicore systems, thus leading to savings in terms of execution time.

3.2 TARGETING DNN INFERENCE VIA EFFICIENT UTILIZATION OF HETEROGENEOUS PRECISION DNN ACCELERATORS

In this work, we couple the concepts of HDAs and multi-DNN workloads to propose and investigate the design of HDAs based on the numerical precision of the NPUs, i.e., NPUs that comprise different bit-width MAC arrays (e.g., 7-, 6-bit), and how these HDAs impact the response time of the system. We propose a complete run-time flow that enables efficient execution of diverse DNN workloads on such HDAs. Reducing the computational precision of the NPU allows us to leverage the resulting area gain to scale its size, boosting the NPU's processing power (i.e., integrating more MAC units), which leads to high energy gains and significant improvements in NN response time.

3.2.1 Methodology

Fig. 3.9 depicts an overview of our methodology for the design and efficient utilization of HDAs under different NN-based workloads. At design time, our approach starts from the circuit level where we use commercial EDA tools to perform the full-chip design of systolic MAC arrays, similar to the one used in the Google TPU microarchitecture. We design varying low bit-width (e.g., 7- and 6-bit) MAC arrays, namely Low Precision (LP) NPUs, and we explore the trade-offs in terms of energy consumption and area when compared to conventional 8-bit NPUs (also used as the baseline in this work). We also investigate the execution time benefits that LP NPUs bring, due to the scaling of the number of MAC units, during NN inference with an HDA cycle-accurate simulator. At run-time, we consider an NN workload queue and bridge the gap between low-bit width NPUs and accuracy loss by employing a fast and efficient post-training quantization method and a run-time NN-to-NPU scheduling policy. The latter finds the best NPU to utilize, in order to decrease NN response time while satisfying a specified accuracy threshold. All the steps described hereafter are implemented on top of industrial tool flows to enable the most accurate analysis.

The NN workload queue we consider is inspired by an edge server scenario. In such circumstances, a server is processing a variety of distinct DNN requests from multiple users and applications. Therefore, the response time of the NNs varies as their arrival rates fluctuate. As an example, consider a user's device offloading the execution of an AR workload that requires DNNs for object detection and classification [54], a common use-case that requires the execution of multiple DNNs. Similar use cases include the usage of Unmanned Aerial Vehicles (UAVs) to assist and accelerate video-based applications via the use of drones [115]. In these cases, drones are equipped with hardware accelerators to speed up the content delivery and analysis process at the edge, while keeping power consumption low. Many drone-based services heavily rely on the utilization of concurrent DNNs, such as object detection and classification for public safety services [116].

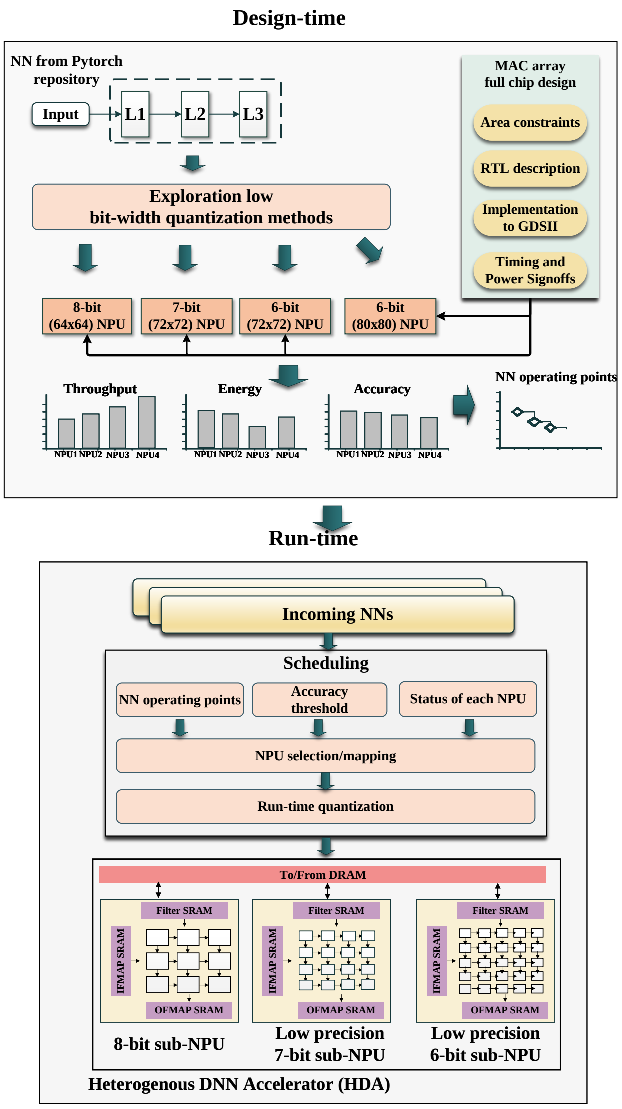

Figure 3.9: Overview of the proposed framework [4].

It is evident that the application/user requirements are characterized by a variety of computational and power consumption needs. By employing scenarios that target edge devices, we aim to stress this observed heterogeneity, and we eventually consider multiple NNs with different characteristics. The goal of our proposed methodology is to make use of the error resilience of NNs to satisfy specific accuracy thresholds, while significantly reducing energy consumption on the accelerator employed on the edge server and improving the average NN response time.

3.2.1.1 Low bit-width NPUs

As aforementioned, the MAC unit is the essential building block of our NPU microarchitecture, performing millions of operations per NN inference. Thus, the overall execution time of the NPU heavily depends on the size of the MAC array. The baseline architecture for our approach, also used in our evaluation, is a system that accommodates four NPUs, each comprising an 8-bit 64×64 systolic MAC array [11]. Our goal is to follow this architectural principle of four NPUs but introduce asymmetric heterogeneity in the form of lower bit-width NPUs.

By reducing bit-width, the area and power consumption of the MAC array reduce significantly, allowing us to increase the number of MAC units that are integrated into the NPU. Consequently, we designed and synthesized NPUs operating at 7- and 6-bit that allowed us to scale the MAC array size to 72×72 and 80×80 while having a negligible area overhead of only 5% when compared to the conventional 8-bit 64×64 NPU, as shown in Table 3.2. Additionally, in the case of the 6-bit 72×72 LP NPU design, the number of the integrated MAC units do not cause any area overhead but instead results in a 16% smaller area when compared to the 8-bit 64×64 NPU. Overall, our goal was to stay as close as possible to the 8-bit 64×64 baseline area to achieve fair comparisons in our exploration. Having low bit-width (7- and 6-bit) NPUs with smaller sizes (e.g., 64×64 or 32×32) would have resulted in wasted area and significant throughput reduction. On the other hand, having more MAC units (e.g., 80×80) would have violated the area constraint.

Moreover, Table 3.2 shows the results of the conducted power analysis of the LP NPU designs. According to this analysis, when compared to the 8-bit 64×64 NPU, the 7-bit 72×72 MAC array has 28% less power consumption, while the 6-bit 72×72 and 6-bit 80×80 NPUs achieve power gains of 39% and 24% respectively.

At this point, it is important to mention that we experimented with NPU designs with even lower bit-widths (e.g., 4- and 5-bit). Even though such NPU designs are

Table 3.2: Power and Area analysis of our LP NPUs compared to the “big” NPU

NPU size	**Power gain**	**Norm. area**	**# MAC units**
7-bit 72×72	28%	1.05	5,184
6-bit 72×72	39%	0.84	5,184
6-bit 80×80	24%	1.05	6,400

possible to design and they still preserve the benefits of lower precisions (both in area and power consumption), they could not be utilized in our design. The main reason, which is explained in detail in Section 3.2.1.2, lies in the accuracy drop of the NN while operating at such small precision. In order to utilize such a low-bit width NPU, an efficient quantization mechanism is needed. Specifically, from the NNs we considered only 5% of them could tolerate such low precision without accuracy loss greater than 1.5%. Therefore, considering 5- and 4-bit NPUs would lead to no benefits since the accuracy loss would be prohibitive.

To precisely obtain the power and area values of Table 3.2, all the examined NPUs are described in Verilog RTL using the optimized arithmetic components of the commercial Synopsys DesignWare library. The designs are then synthesized using the Synopsys Design Compiler and are mapped to a 14nm library fully calibrated with Intel measurement data [14]. All optimizations are enabled during synthesis. Then, the Cadence toolflow is used for the full chip implementation. We implement the floorplan and power delivery network and perform high optimization place and route including clock tree synthesis. Power and delay analysis is performed on the *post-layout* netlist including parasitics. To obtain accurate switching activity for the power analysis, we run timing simulations using realistic input traces extracted from the inference phase of the examined networks. All the NPUs are operated at the same clock frequency of 700MHz. Given the operating frequency and average power consumption, to compute the energy consumption we also need to calculate cycles required for each inference, which is achieved through the SCALE-Sim cycle-accurate CNN simulator from ARM [56]. The aforementioned toolflow is based on commercial tools, providing most accurate modeling of the hardware characteristics of the examined NPUs.

To investigate the benefits of the MAC array size scaling with respect to NN execution time, we heavily extended SCALE-Sim [56] in the following ways. First, we created a high-level interface that allows us to express an NN using the Pytorch machine learning library. Then, SCALE-Sim retrieves all the necessary information regarding the inputs, the structure, and the layer dependencies from the Pytorch representation, and performs a cycle-accurate simulation on a MAC systolic array with defined characteristics (e.g., size of the MAC array, buffer/memory size, etc.). Second, and more importantly, we enhanced SCALE-Sim to support heterogeneous accelerator designs. With these two extensions, we can evaluate different HDA configurations from a single PyTorch description in a fast and accurate manner.

3.2.1.2 NN quantization for low precision NPUs

In Section 3.2.1.1, we explained how the utilization of low numerical precision at the MAC unit level enables larger NPU sizes that improve NN latency and energy consumption. However, reducing the number of bits used for weights and activations can result in unacceptable accuracy loss if applied in isolation. Thus, in order to sustain high NN accuracy while executing on heterogeneous DNN accelerators, we employ low bit-width quantization. In this work, we consider 8-bit as the baseline precision of the NPU, as this is considered the standard numerical representation in embedded devices and edge accelerators [102, 117]. Additionally, 8-bit quantization is used for both weights and activations [11] due to device properties and execution requirements (e.g., fast and power-efficient inference).

Quantization inflicts a loss in accuracy for a target NN. Retraining the quantized NN is a powerful and reliable method to recover the induced accuracy loss. However, retraining a DNN at the edge is a very time-consuming process that can violate any user-based time constraints, deteriorating the overall response time of the system. In addition, retraining might not even be an option when the training dataset is not available (privacy reasons, no longer accessible, etc) [118]. Furthermore, since we are evaluating a queue where NNs *continuously arrive* and need to be scheduled at run-time to different NPUs, applying quantization with retraining would practically be unfeasible.

Our primary and fundamental objective, while using low-bit-width NPUs, is to keep accuracy as close as possible to the accuracy when executing at 8-bit. At this point, we would like to mention that the focus of this work *is not to propose a new quantization mechanism.* Contrary, our goal is to investigate multiple methods and employ a low-overhead quantization as a way to improve latency and energy consumption, without severe accuracy degradation. To that end, we investigated multiple low-bit width post-training quantization methods based on recently published approaches, such as symmetric [119] and asymmetric min/max quantization [28], PWLQ [120], LAPQ [121], and ACIQ [118]. However, many of these methods rely on offline statistics or other techniques that cannot be used at run-time. Additionally, other state-of-the-art techniques such as BRECQ [122] still need a small subset of the training dataset to calibrate the quantized model, but in many cases where pre-trained DNN models are utilized, the initial training dataset might not be accessible.

Since we require a fast and low-overhead quantization method to use it at run-time, we utilized the core analytical clipping method for weights and activations presented in [118]. [118] tries to minimize the mean-squared-error when estimating the optimal clipping value. Our method is based on this analytical clipping approach for quantizing both weights and activations, as it is simple to use during run-time, producing also good results for 7 and 6 bits. To avoid additional run-time overhead, we did not combine the analytical clipping with per-channel bit allocation and bias-correction techniques that require special hardware.

Additionally, as presented in the original research in [118], the authors do not quantize the first, last, and pooling layers. This convention, or variations of it (e.g., only the first layer skipped), is widely used [120,123,124]. Currently, however, NPUs do not support preemptive scheduling. Thus, with the current low-bit quantization method in [118], it is not possible to map an NN on a low bit-width NPU (e.g., 7-bit 72×72), as the first and last layers need to be mapped on an 8-bit NPU. This problem becomes more prevalent when multiple NNs have to be executed concurrently, creating resource contention and execution time overhead. To overcome this problem, our method quantizes both weights and activations for all layers including the pooling ones, thus allowing us to schedule an NN exclusively, without interruptions, on a low-bit width NPU if needed.

Compared with [118], our approach enables run-time quantization and NN execution on low-bit-width NPUs. We let go of the exclusion of some DNN layers from the quantization process which is a commonly used convention, at the cost of a slight decrease in accuracy.

3.2.1.3 Heterogeneity-aware NN-to-NPU scheduling

An HDA with k NPUs can be described as $HDA_k = \{NPU_1^{n_1}, NPU_2^{n_2}, \dots, NPU_k^{n_k}\}$, where n_i corresponds to the bit-width of each NPU. Accordingly, an NN can be profiled and described as $\{(C, A)_{NPU_1^{n_1}}, (C, A)_{NPU_2^{n_2}}, \dots, (C, A)_{NPU_k^{n_k}}\}$, where (1) C corresponds to the execution cycles that required by that NN to run on the $NPU_i^{n_i}$ and (2) A corresponds to the respective accuracy loss. A is calculated w.r.t. 8-bit inference when weights and activations are quantized to n_i bits. This information and our run-time quantization (Section 3.2.1.2) are used in our NN-to-NPU scheduling.

Assuming (1) a queue of M ($M > k$) NNs, and (2) an acceptable accuracy drop threshold Acc_{thr} we apply an efficient NN-to-NPU scheduling that targets to minimize the average response time of the NNs in the queue. The time from the arrival up to the exit of a NN from the system is the *response time.* In other words, the response time includes the execution time as well as the waiting time inside the queue. At this point, we want to mention that the proposed HDAs target systems with an increased workload, i.e., many NNs arrive in the queue waiting to be executed. We want to avoid cases in which the queue is empty and a single NN arrives, as in that case the most appropriate NPU will always be selected hiding the benefits of heterogeneity. In that case, the unused NPUs will be power gated and high energy savings will be still obtained when a low bit-width NPU is selected. Based on the analysis presented in Section 3.2.1.1 and Table 3.2, ideally, we want to map each NN to the NPU with the lowest acceptable bit-width that does not violate the accuracy drop threshold ($A_{NPU_j^{n_j}} \leq Acc_{thr}$). As aforementioned, low-bit width NPUs incorporate more MAC units significantly reducing the inference speed of the NN, while the reduced numerical representation bit width also decreases the energy consumption of the MAC units.

However, such scheduling would significantly increase the waiting time of NNs in cases where all NNs are assigned to the same NPU. To that end, and in order to provide further information and insight during scheduling, we conduct an offline evaluation of multiple already published state-of-art DNNs under different computational precisions. The goal of this step is to help the scheduler identify which DNNs are sensitive and which DNNs can tolerate lower computational precision, thus being candidates for execution on the LP, i.e., lower bit-width NPUs. The extracted information is stored as a look-up table $LUT_{NN}^{Acc_{thr}}$ at the scheduler to help it take decisions at run-time, depending on the NN under consideration and the given Acc_{thr}. The Acc_{thr} values are set at the *design-time*, and the described profiling procedure is being conducted under these values. Therefore, we can know the accuracy of the NNs when quantized to lower bit-widths beforehand, a piece of information that plays a crucial part in the scheduling method that is described below. When we refer to Acc_{thr} we consider the average accuracy achieved on the entire target dataset and is used as an indicator of the sensitivity of NNs [124]. Note that the profiling of the NN is performed offline and only once. DNNs not previously encountered need to undergo this profiling procedure, since it provides necessary feedback to the

Algorithm 1 Run-time NN-to-NPU mapping

```
Inputs: 1. Heterogeneous NN accelerator: $HDA_k$,
        2. Arrival queue of NNs,
        3. Accuracy Loss Threshold: $Acc_{thr}$,
        4. Look-up table of NN accuracy: $LUT_{NN}^{Acc_{thr}}$
 1: for each $NN_i \in$ Arrival queue
 2:   for each $NPU_j^{n_j} \in LUT_{NN_i}^{Acc_{thr}}$
 3:     for each $NN_x \in RL_{NPU_j^{n_j}}$
 4:       $response\_time_{NN_i}^{NPU_j^{n_j}} = NPU\_workload + cycles_{NN_x}$
 5:     $response\_time_{NN_i}^{NPU_j^{n_j}}$ += $cycles_{NN_i}$
 6:   $target\_NPU_{NN_i}$ = NPU with lowest $response\_time_{NN_i}$
 7:   quantize $NN_i$ for $target\_NPU$ (8-bit or lower)
 8:   $RL_{target\_NPU} \leftarrow$ add($NN_i$)
 9: for each $NPU_j^{n_j} \in HDA_k$
10:   if $NPU_j^{n_j}$ is available
11:     remove $NN$ from head of $RL_{NPU_j^{n_j}}$
12:     schedule $NN$
```

scheduling algorithm presented below. Since the only bit-widths we consider are no less than 6-bit, it is not a time-consuming process. Additionally, it does not inflict any additional hardware overhead since it can be done independently.

Algorithm 1 presents the run-time NN-to-NPU scheduling that targets optimizing the average response time of our HDA. The inputs to the algorithm are (1) the characteristics of the HDA, (2) the arrival queue of NNs to be executed (3) the accuracy threshold Acc_{thr}, and (4) the $LUT_{NN}^{Acc_{thr}}$ that was acquired during the offline profiling procedure. The scheduling algorithm is triggered whenever a new NN enters the arrival queue or when an NPU is unoccupied (e.g., a NN leaves the system). This is actually reflected by the two main loops of the algorithm (lines ①-⑧, lines ⑨-⑫). Additionally, each NPU has its own ready list (RL) which contains the NNs to be executed on that particular NPU. For each NN in the arrival queue, we utilize the look-up-table $LUT_{NN}^{Acc_{thr}}$ for the specified accuracy threshold Acc_{thr} to check the possible NPUs the NN can be executed on. Then, we examine the contents of the ready lists for each one of these NPUs in order to calculate the response time for the examined NN (lines ①-③). The response time is calculated by adding the cycles required for the NPU to finish any current workload, the cycles of the NNs inside the ready list (this is also considered to be the waiting time) and the cycles of the examined NN (lines ④-⑤). Then, we select the NPU that features the lowest calculated response time, we quantize the NN (8-bit or lower), and we add it to the corresponding ready list (lines ⑥-⑧). Then, we check whether any NPU is currently available (this loop is also triggered when an NN leaves the system). If this is the case, then we schedule the first NN from the corresponding ready list (lines ⑨-⑫). As aforementioned, the scheduling algorithm is triggered whenever a NN enters the arrival queue or an NPU becomes available. Thus, by utilizing different ready lists for each NPU based on the minimum response time, if multiple NNs arrive at the same time, but the first one cannot be executed, the utilized algorithm will not block. Contrary, it will try to place all NNs appropriately so as to reduce the overall response time.

Note that each time an NN is mapped to an NPU and gets quantized, it is not re-quantized for this bit-width again in the future. Therefore, the quantization overhead is only applied once per NN-to-NPU mapping.

3.2.2 Results & Evaluation

Based on the analysis in Section 3.2.1.1, we designed low-bit width NPUs that allow us to scale the number of MAC units. Particularly, we considered 7-bit 72×72, 6-bit 72×72, and 6-bit 80×80 MAC arrays. On average, the 72×72 MAC array has a latency gain of 9%, while the gain for 80×80 increases to 18% when compared to the 64×64 baseline. For 7-bit and 6-bit NPUs we can accommodate more MAC units with resulting power gains when compared to the baseline, with a small 5% area overhead. Designing a lower bit-width (e.g., 7- or 6-bit) 64×64 NPU would have resulted in lower throughput and wasted area.

We evaluate 10 different HDA designs as shown in Table 3.3 All of the HDA designs we evaluate utilize one “big” 8-bit 64×64 NPU as a fallback choice, in order to serve

Table 3.3: Evaluated HDAs: NPUs and normalized area.

	8-bit 64×64	**7-bit 72×72**	**6-bit 72×72**	**6-bit 80×80**	**Norm. area**
Baseline	4	-	-	-	1
HDA-1	1	3	-	-	1.04
HDA-2	1	2	1	-	1
HDA-3	1	2	-	1	1.04
HDA-4	1	1	2	-	0.94
HDA-5	1	1	1	1	1
HDA-6	1	1	-	2	1.04
HDA-7	1	-	3	-	0.9
HDA-8	1	-	2	1	0.94
HDA-9	1	-	1	2	1
HDA-10	1	-	-	3	1.04

sensitive NNs and thus always satisfy the accuracy threshold. For the remaining three NPUs in the designed HDAs, we evaluate all the possible combinations with respect to NPU sizes of 7-bit 72×72, 6-bit 72×72, and 6-bit 80×80. Table 3.3 also depicts the area of each HDA. The area is reported as a normalized value with respect to the area of the baseline, i.e., a homogeneous design consisting of four 8-bit 64×64 NPUs.

Regarding the evaluated NNs, we consider the following: AlexNet, GoogleNet, InceptionV3, MnasNet, MobileNetV2, VGG11, VGG13, VGG16, VGG19, ResNet18, ResNet34, ResNet50, ResNet101, ResNet152, ResNext101, ResNext50, ShuffleNetV2, SqueezeNetV1.0, SqueezeNetV1.1, WideResNet50, and WideResNet101. All of these NNs comprise different structures and have vastly different model sizes and numbers of parameters, having diverse requirements in terms of computation. We then created different queues of incoming NNs to evaluate the performance of each HDA. Each queue comprises randomly selected NNs from the aforementioned ones, and is characterized by the total number of incoming NNs M (the size of the queue), and the mean arrival time for each NN (T_{ar}).

We distinguished five cases of T_{ar} based on the distribution of the cycles of all evaluated NNs when they run on an 8-bit 64×64 NPU. Particularly, we considered:

- T_{ar}^{0}: The mean arrival time for each NN is set equal to the cycles of the fastest NN (minimum value);

Table 3.4: Overview of the evaluated scenarios

Accuracy threshold (Acc_{thr})	**Size of queue**	**Arrival rates per queue** (T_{ar})
1.5%	30, 50, 70	T_{ar}^0, T_{ar}^1, T_{ar}^2, T_{ar}^3, T_{ar}^4
1.0%	30, 50, 70	T_{ar}^0, T_{ar}^1, T_{ar}^2, T_{ar}^3, T_{ar}^4
0.5%	30, 50, 70	T_{ar}^0, T_{ar}^1, T_{ar}^2, T_{ar}^3, T_{ar}^4

- T_{ar}^1: The mean arrival time is set to the first quartile;
- T_{ar}^2: The mean arrival time is set to the second quartile
- T_{ar}^3: The mean arrival time is set to the third quartile;
- T_{ar}^4: The mean arrival time is set equal to the cycles of the slowest NN (maximum value).

By using these different T_{ar} values, we aim to apply a variety of pressure on the considered HDAs of Table 3.3. We took into consideration three different accuracy threshold Acc_{thr} values in our evaluation: 0.5%, 1%, and 1.5%. A summary of our evaluated scenarios for the examined Acc_{thr} values is shown in Table 3.4. Each scenario was evaluated 100 times, each time with a random NN composition. In total, we conducted 4500 experiments. We integrated the SCALE-Sim cycle-accurate CNN simulator [56] into STOMP [125] and evaluated the scheduling policy on different arrival rates.

Figures 3.10 - 3.12 depict the normalized response time and energy consumption of each HDA for the three different queue sizes (30, 50, and 70 respectively) compared to the baseline when $Acc_{thr} = 1.5\%$. Each boxplot shows the combined behavior under all arrival times for all experiments (i.e., 100 evaluations per arrival time). All HDAs have on average better response time and significant energy gains, since all the evaluated NNs satisfy the 1.5% accuracy drop threshold when quantized to 7 bits, but only half of them satisfy it for 6 bits. Consequently, the HDAs 7-10 are affected more as they integrate three 6-bit-width MAC arrays. For the HDA-7 we can observe cases where the response

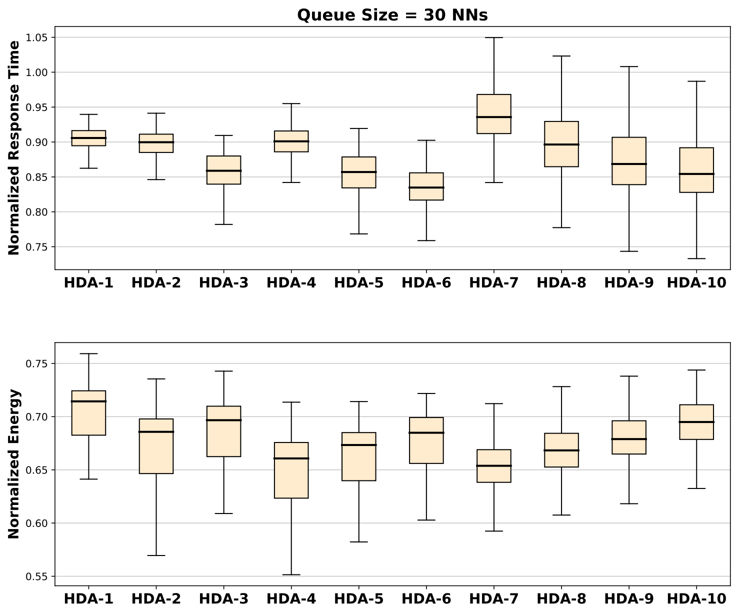

Figure 3.10: Results for a queue consisting of 30 NNs for $Acc_{thr} = 1.5\%$

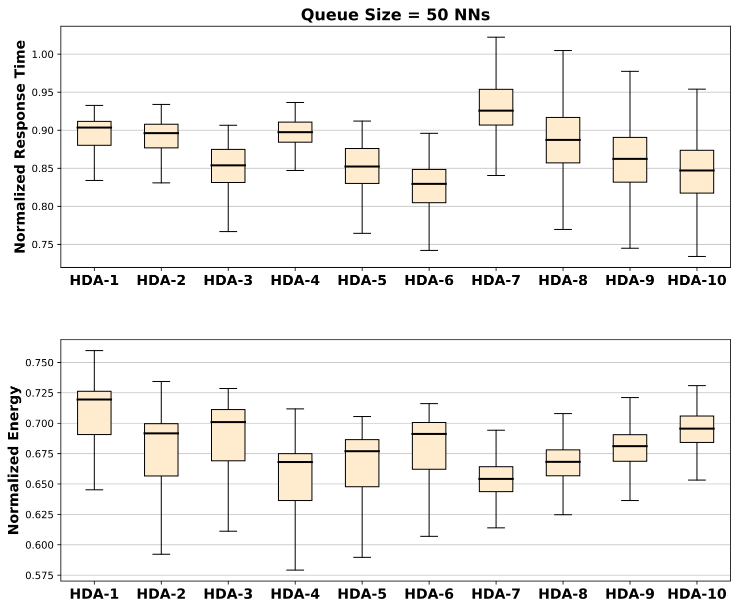

Figure 3.11: Results for a queue consisting of 50 NNs for $Acc_{thr} = 1.5\%$

time is above the baseline for small arrival times (T_{ar}^0 and T_{ar}^1), because NNs arrive fast and many of them can be executed only on the 8-bit NPU, increasing the waiting time. The same phenomenon happens for HDAs 8-10 but the waiting time overhead is mitigated by the fast execution on the 80×80 NPUs.

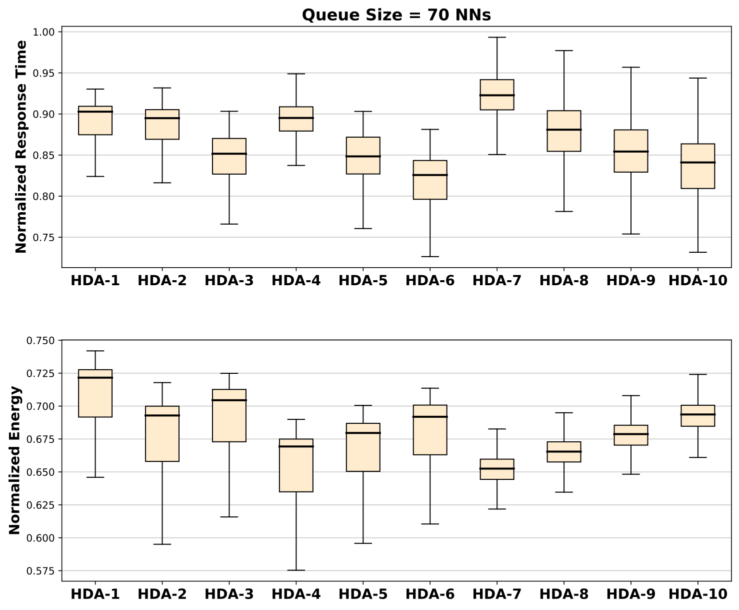

Figure 3.12: Results for a queue consisting of 70 NNs for $Acc_{thr} = 1.5\%$

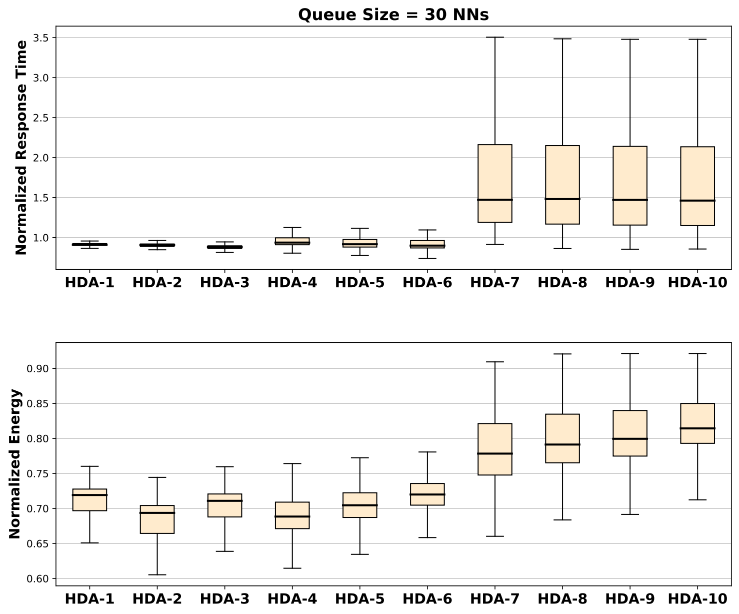

Figure 3.13: Results for a queue consisting of 30 NNs for $Acc_{thr} = 1\%$

Respectively, Figures 3.13 - 3.15 show the normalized response time and energy consumption of each HDA for the considered queue sizes when $Acc_{thr} = 1\%$. In this case, since the accuracy drop threshold is now relatively small, only a third of the NNs can satisfy it using 6-bit quantization. Thus, HDAs that mainly utilize 6-bit NPUs are greatly

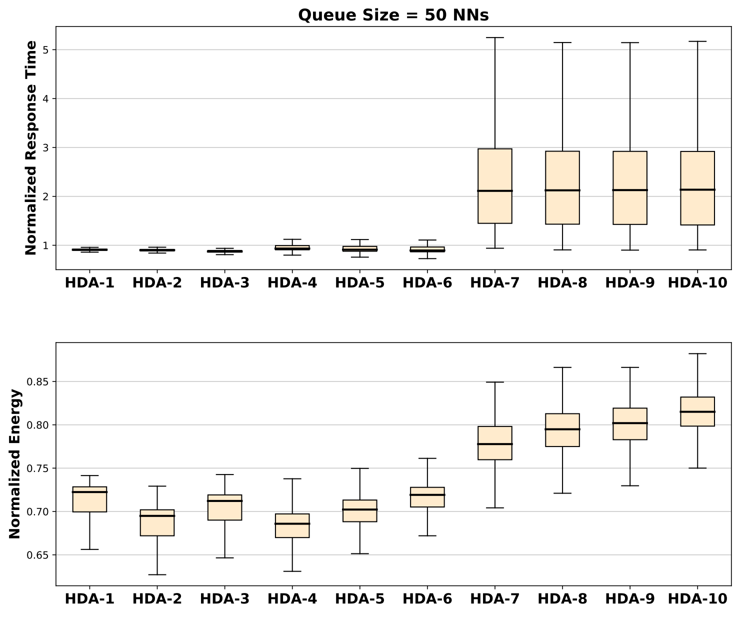

Figure 3.14: Results for a queue consisting of 50 NNs for $Acc_{thr} = 1\%$

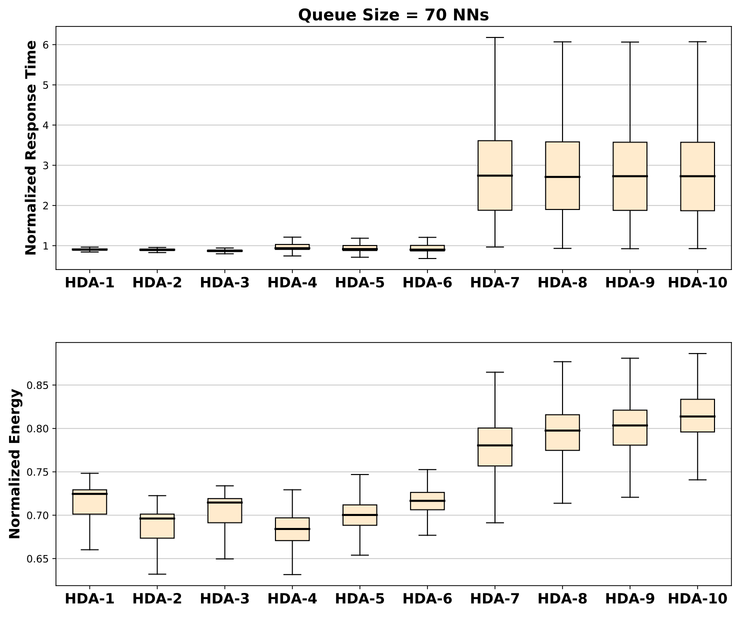

Figure 3.15: Results for a queue consisting of 70 NNs for $Acc_{thr} = 1\%$

impacted. For example, HDAs 7-10 have a worse response time than the baseline, as their 6-bit NPUs are rarely utilized, and the NNs fight to access the remaining NPUs.

When $Acc_{thr} = 0.5\%$, the drop is so small that no NN can utilize 6-bit NPUs. Thus only HDA-1 is a valid design. For this minor accuracy drop, our heterogeneous HDA-1

Table 3.5: Comparison of the evaluated HDAs against a monolithic NPU

	Selected HDA designs					Homogeneous 8-bit
Max acc. drop	1.5%		1%		0.5%	-
Accelerator	HDA-4	HDA-6	HDA-2	HDA-3	HDA-1	128 × 128
Energy gain	35.17%	32.3%	31.8%	29.9%	28.9%	-355%
Resp. time gain	11%	18.2%	10.6%	13.4%	10%	-51%
Norm. area	0.94	1.04	1	1.04	1.04	1
Comments	most energy efficient	highest resp. time gain	most energy efficient	highest resp. time gain	-	-

design achieved reduced response time on average by 10% having also a significant energy gain of 28.9%.

Table 3.5 summarizes the gains of the selected HDAs with respect to energy and response time compared to a homogeneous design that accommodates four 8-bit 64 × 64 NPUs (i.e., the baseline design). Additionally, we compared the selected HDAs against a large 8-bit 128 × 128 NPU which has a similar area as our baseline architecture (Table 3.3). Importantly, the results in Table 3.5 are the average values for all the utilized arrival rates and queue sizes. When compared to a single 64 × 64 NPU, the 128 × 128 NPU is approximately 48% faster. However, this gain is achieved for the maximum arrival rate (T^4_{ar}) as in that case the queue is mainly empty. As the arrival rate decreases, the 128 × 128 NPU suffers from resource contention, yielding significant deterioration in terms of NN response time (51% reduction). Regarding energy consumption, the 128 × 128 NPU consumes about four times more energy, revealing the benefits of the proposed asymmetric NPU design. Even though the 8-bit 128 × 128 NPU can accommodate DNNs without accuracy losses caused by quantization, this design is not possible to contest our proposed HDAs in terms of efficiently serving multi-DNN workloads, since it comprises a single NPU.

Overall, from Table 3.5 we can infer that guaranteeing one and only one optimal design for any queue is not a straight-forward decision, since the behavior of an HDA heavily depends on the distribution of the DNNs in the queue with each DNN having different sensitivity to low bit-width quantization. For instance, for a 1.5% accuracy drop threshold, the most energy-efficient design is HDA-4 but for a more strict 1% threshold it is HDA-2. Additionally, simply employing lower bit-width NPUs does not automatically result in significant gains in energy or response time, especially when dealing with tight accuracy constraints. LP NPU designs might result in low energy consumption, however not all DNNs can tolerate such a drop in precision as observed in our evaluation for a 0.5% accuracy drop threshold. This is actually one of the main differentiators of this work compared to previous approaches that focus on custom and specific NN-tailored designs.

CHAPTER 4
FORMAL PROPERTY INTEGRATION IN APPROXIMATE COMPUTING

In this chapter, two works will be presented that combine formal properties, falsification, and parameter mining in approximation applications. The first work introduces a framework that uses parameter mining to map approximate multipliers to DNN weights. The second work presents a framework that uses the falsification procedure for joint quantization and pruning of DNNs.

4.1 PRELIMINARIES

4.1.1 Signal Temporal Logic (STL)

This section presents background information about Signal Temporal Logic (STL) and how we utilize this concept for clustered CMPs. In the following, $\mathbb{N}$ is the set of natural numbers and $\mathbb{R}$ the set of reals. We view the input-output behavior of the clustered CMP as a discrete time signal (also referred to as *trace* in the following) $\sigma : N \to \mathcal{Y}$, where $N \subseteq \mathbb{N}$ is the domain of the signal and $(\mathcal{Y}, \mathbf{d})$ is an output (measurement) space equipped with a metric $\mathbf{d} : \mathcal{Y} \times \mathcal{Y} \to \mathcal{V}$ (see [126] for a more detailed discussion). For example, in this work, the space $\mathcal{Y}$ consists of the measurements over time of temperature, power, application performance, and system throughput. Also, if $\mathcal{V} = \overline{\mathbb{R}}_{\geq 0} \triangleq \mathbb{R}_{\geq 0} \cup \{+\infty\}$, then the truth values of STL would be over the set $\overline{\mathbb{R}} = \mathbb{R} \cup \{\pm\infty\}$. We assume the same constant sampling rate Δt for all the signals σ. That is, the value of the signal at sample i, i.e., $\sigma(i)$, is sampled at time $i\Delta t$. The results in this work do not depend on the assumption of a constant sampling rate, but this assumption greatly simplifies the notation.

Originally, STL [127] was defined to express bounded time requirements over continuous-time signals. However, the extension over discrete-time signals, which we review here, is straightforward [128].

Definition 4.1.1 (STL Syntax for discrete-time signals). *Let x be a vector variable, i.e., $x = [x_1, \dots, x_n]^T$, $p(x)$ be a function over the reals, and $\mathcal{I}$ be any non-empty interval of $\mathbb{R}_{\geq 0}$. The syntax for STL formulas is provided by the grammar:*

$$\phi ::= \mathbf{T} \mid p(x) \geq 0 \mid \neg\phi \mid \phi \vee \phi \mid \bigcirc_{\mathcal{I}}\, \phi \mid \phi\, \mathcal{U}_{\mathcal{I}} \phi$$

where $\mathbf{T}$ is true, $\bigcirc_{\mathcal{I}}$ is the next sample operator, and $\mathcal{U}_{\mathcal{I}}$ is the until operator.

In the following, for simplicity in the presentation, we will assume that n in Def. 4.1.1 is the dimensionality of the output space $\mathcal{Y}$. For this case study, we will be using a quantitative interpretation of the STL semantics (see [129] for an overview). In order to define quantitative semantics with a topological interpretation over arbitrary predicates $p(x) \geq 0$, we will need to use a metric $\mathbf{d}$ to define a signed distance function:

Definition 4.1.2 (Signed Distance). *Let $x \in X$ be a point, $S \subseteq X$ be a set and $\mathbf{d}$ be a metric. Then, we define the Signed Distance from x to S to be*

$$\mathbf{Dist_d}(x, S) := \begin{cases} -\inf\{\mathbf{d}(x, x') \mid x' \in S\} & \text{if } x \notin S \\ \inf\{\mathbf{d}(x, x') \mid x' \notin S\} & \text{if } x \in S \end{cases}$$

where ($\sqcup$) and ($\sqcap$) correspond to the extended definition of supremum and infimum, respectively. That is, when $\mathcal{V} = \overline{\mathbb{R}}_{\geq 0}$, then $\inf \emptyset := +\infty$. Intuitively, the distance function returns positive values when x is in the set S and negative values when x is outside the set S.

We review STL semantics that map a formula φ and a trace σ to a value drawn from the set $\overline{\mathbb{R}}$. We denote the robust valuation of the formula φ over the trace σ at sample i by $[\![\varphi]\!]_{\mathbf{d}}(\sigma, i)$. The semantics for a predicate $p(x) \geq 0$ evaluated at time i over trace σ is defined as the distance between $\sigma(i)$ and the set $[\![p(x) \geq 0]\!] \triangleq \{x \,|\, p(x) \geq 0\}$. Intuitively, this distance represents how robustly the point $\sigma(i)$ lies within (or is outside) the set $[\![p(x) \geq 0]\!]$. If this distance is zero, then the smallest perturbation of the point $\sigma(i)$ can affect the outcome of $\sigma(i) \in [\![p(x) \geq 0]\!]$.

Definition 4.1.3 (Discrete-Time Robust Semantics). *Consider a metric space* $(\mathcal{Y}, \mathbf{d})$. *Let* $\sigma : N \to \mathcal{Y}$ *be a trace, then the robust semantics of an STL formula* φ *with respect to* σ *at time sample* i *is defined as:*

$$\begin{aligned}
[\![\mathbf{T}]\!]_{\mathbf{d}}(\sigma, i) &:= \bigsqcup \overline{\mathbb{R}} := +\infty \\
[\![p(x) \geq 0]\!]_{\mathbf{d}}(\sigma, i) &:= \mathbf{Dist}_{\mathbf{d}}(\sigma(i), [\![p(x) \geq 0]\!]) \\
[\![\neg\varphi_1]\!]_{\mathbf{d}}(\sigma, i) &:= -[\![\varphi_1]\!]_{\mathbf{d}}(\sigma, i) \\
[\![\varphi_1 \vee \varphi_2]\!]_{\mathbf{d}}(\sigma, i) &:= [\![\varphi_1]\!]_{\mathbf{d}}(\sigma, i) \sqcup [\![\varphi_2]\!]_{\mathbf{d}}(\sigma, i)
\end{aligned}$$

$$[\![\bigcirc_{\mathcal{I}}\varphi_1]\!]_{\mathbf{d}}(\sigma, i) := \begin{cases} [\![\varphi_1]\!]_{\mathbf{d}}(\sigma, i+1) & \text{, } \textit{if } i+1 \in N \textit{ and} \\ & (i+1)\Delta t \in (i\Delta t \oplus \mathcal{I}) \\ -\infty & \text{, } \textit{otherwise} \end{cases}$$

$$[\![\varphi_1 \mathcal{U}_{\mathcal{I}} \varphi_2]\!]_{\mathbf{d}}(\sigma, i) :=$$

$$\bigsqcup_{i' \in \{j \in N \mid j\Delta t \in (i\Delta t + \mathcal{I})\}} \left([\![\varphi_2]\!]_{\mathbf{d}}(\sigma, i') \sqcap_{i \leq i'' < i'} [\![\varphi_1]\!]_{\mathbf{d}}(\sigma, i'')\right)$$

where $t \oplus \mathcal{I} = \{t'' \mid \exists t' \in \mathcal{I} \,.\, t'' = t + t'\}$.

Intuitively, the requirement $\bigcirc_{\mathcal{I}}\varphi$ states that φ should be true at the next sample, which should occur some time in the physical time interval $\mathcal{I}$. For example, consider $\Delta t = 0.1$ and the formula $\psi = \bigcirc_{[0,0.1]}\mathbf{T}$, then ψ is true ($\top$) at sample i since $(i+1)\Delta t - i\Delta t = \Delta t \in [0, 0.1]$. However, for $\Delta t = 0.2$, ψ would evaluate to false ($\bot$). The operator $\varphi_1 \mathcal{U}_{\mathcal{I}} \varphi_2$ states that φ_2 should be satisfied at some time in the interval $\mathcal{I}$

and until then φ_1 should hold. The other common Boolean and temporal operators can be defined as syntactic abbreviations (see [127, 128]). For example:

- $\Diamond_{\mathcal{I}}\phi \equiv \mathbf{T}\,\mathcal{U}_{\mathcal{I}}\phi$ stands for eventually at some time in the time interval $\mathcal{I}$, ϕ should be true, and
- $\Box_{\mathcal{I}}\phi \equiv \neg\Diamond_{\mathcal{I}}\neg\phi$ stands for always during the interval $\mathcal{I}$, ϕ should be true. When $\mathcal{I} = [0, \infty)$, we will be dropping $\mathcal{I}$ from the notation, e.g., $\Box_{[0,\infty)}\phi \equiv \Box\phi$.

In the following, we let $(\sigma, i) \models \varphi$ denote the standard Boolean STL satisfiability. Note that Boolean satisfiability reduces to an application of Def. 4.1.3 wherein the metric $\mathbf{d}$ is the discrete metric. It is easy to show that if the signal satisfies the property, then its robustness is non-negative and, similarly, if the signal does not satisfy the property, then its robustness is non-positive [128]. The robustness $[\![\varphi]\!]_{\mathbf{d}}(\sigma, i)$ can be computed in polynomial time in the size of the formula and the time domain N of σ. The authors in [129] provide a review of different computation algorithms.

4.1.2 Parametric STL (PSTL) & Parameter Mining

In many applications, it is important to explore the properties that a signal satisfies as opposed to issuing a verdict that a signal satisfies or does not satisfy a given specification φ. For example, instead of asking the question, does σ satisfy the specification that the power is always less than $6W$ in the first 5sec, i.e., $\varphi = \Box_{[0,5]}(Power \leq 6)$, we may leave the time constraint as parameter θ to be mined, e.g., $\varphi[\theta] = \Box_{[0,\theta]}(Power \leq 6)$. In other words, we would like to learn what is the longest time θ for which the requirement holds.

Parametric Signal Temporal Logic (PSTL) was introduced in [130] as a formal language to capture such questions. In PSTL, propositions of the form $p(x) \geq 0$ do not only depend on the state variables x, but also on some parameter vector θ. In lieu of a formal definition, we just mention that in PSTL formulas, i.e., $[\theta]$ with $\theta = [\theta_1, \ldots, \theta_m]$, each parameter θ_i either appears in an arithmetic expression, i.e., $p(y)[\theta_i] \equiv g(y) \leq \theta_i$, or in the timing constraint of a temporal operator, i.e., $\mathcal{I}[\theta_i]$. Given a vector of parameter values $\theta \in \Theta$, then the formula $\phi[\theta]$ is an STL formula. That is, once a parameter valuation is defined, then a PSTL formula is transformed into an STL formula.

The parameter mining problem is formally defined as follows.

Problem 1 (STL Parameter Mining)**.** *Given a PSTL formula $\phi[\theta]$ with a vector of m unknown parameters $\theta \in \Theta = [\underline{\theta}, \overline{\theta}]$ and a system Σ, find the set* $\Psi = \{\theta^* \in \Theta \mid \Sigma \not\models \phi[\theta^*]\}$.

Here, the notation $\Sigma \not\models \phi$ means that there exists a trajectory σ of the system Σ such that σ does not satisfy φ. Dually, the notation $\Sigma \models \phi$ means that all the trajectories σ of the system Σ satisfy φ. Hence, in other words, we are searching for the set of all parameter valuations for which the system is falsified. Note that ideally we would like to identify the set $\Psi^+ = \{\theta^* \in \Theta \mid \Sigma \models \phi[\theta^*]\}$. However, the problem of identifying Ψ^+ is an undecidable one for the types of systems we consider in this paper (see the discussion in [73]).

Sampling positive observations, i.e., $[\![\varphi]\!]_{\mathbf{d}}(\sigma) > 0$ is not a proof that $\Sigma \models \varphi$. On the other hand, sampling a negative observation, i.e., $[\![\varphi]\!]_{\mathbf{d}}(\sigma) < 0$, then we have proof that $\Sigma \models \varphi$. Hence, our goal is to identify the boundary of Ψ as close as possible and, then use the notion of temporal logic robustness to provide a measure of robust system performance.

We can provide an approximate solution to Problem 1 by translating it into an optimization problem through the theory of temporal logic robustness. Formally, we solve the following optimization problem:

$$\begin{aligned} \text{optimize} \quad & f(\theta) \\ \text{subject to} \quad & \theta \in \Theta \text{ and} \\ & [\![\phi[\theta]]\!]_{\mathbf{d}}(\Sigma) = \min_{\mu \in \mathcal{L}_{\hat{\tau}}(\Sigma)} [\![\phi[\theta]]\!]_{\mathbf{d}}(\mu) \leq 0 \end{aligned} \tag{4.1}$$

where $f : \mathbb{R}^n \to \mathbb{R}$ is a non-increasing or a non-decreasing function depending on the monotonicity properties of the function $[\![\varphi[\theta]]\!]_{\mathbf{d}}(\sigma)$ with respect to the parameter θ. In a multi-parametric search problem, the function f formulates a single optimization problem in case the goal is to give priority to certain parameters as opposed to exploring the whole Pareto front. For single-parameter search problems, f is typically the identity function $f(\theta) = \theta$.

Given a parameter vector θ, let the function $[\![\phi[\theta]]\!]_{\mathbf{d}}(\Sigma)$ represent the minimum robustness value of the system over all system behaviors. This value cannot be computed for the classes of systems we are considering. Therefore, we have to compute an under-approximation of Ψ. We reformulate an optimization problem that can be solved using stochastic optimization methods such as Simulated Annealing, Cross-Entropy Optimization, etc. [131].

In particular, we reformulate the optimization problem (Equation 4.1) into a new one where the constraints due to the specification are incorporated into the cost function:

$$\text{optimize}_{\theta \in \Theta} \left(f(\theta) + \begin{cases} \gamma \pm [\![\phi[\theta]]\!]_{\mathbf{d}}(\Sigma) & \\ \quad \text{if } [\![\phi[\theta]]\!]_{\mathbf{d}}(\Sigma) \geq 0 & \\ 0 \quad \text{otherwise} & \end{cases} \right) \tag{4.2}$$

where the sign $(\pm)$ and the parameter γ depend on whether the problem is a maximization or a minimization problem. The parameter γ must be properly chosen so that the solution of the problem in Equation 4.2 is in Θ if and only if $[\![\phi[\vec{\theta}]]\!](\Sigma) \leq 0$. Therefore, if the problem in Equation 4.1 is feasible, then the optimal points of Equation 4.1 and Equation 4.2 are the same.

4.2 ENERGY-EFFICIENT DNN INFERENCE ON APPROXIMATE ACCELERATORS THROUGH FORMAL PROPERTY EXPLORATION

This work presents a framework that systematically explores approximate multiplier mappings on the convolution layers of DNNs. We employ PSTL to automatically search for energy-efficient mappings that can satisfy fine-grain accuracy requirements, instead of only focusing on the final average accuracy of the inference.

4.2.1 Motivation

This section contains a motivational example that shows the necessity of automatic and fine-grain exploration of the error properties of approximate accelerators in order to generate efficient mappings. Overall, we focus our analysis on state-of-art approximate multipliers and mapping methodologies [1, 3, 5, 25]. Specifically, the works in [1, 3, 25] propose reconfigurable multiplier designs that comprise three modes of operation, each one introducing varying levels of error and energy gains. These works also present mapping methodologies and specifically the works in [5, 25] present layer-wise approximation mappings where each layer is entirely mapped to a different multiplier or multiplier mode respectively. To combat minimal energy gains achieved by layer-wise approaches, the works in [1, 3] propose fine-grain weight-oriented methodologies to decide which approximation modes of the respective reconfigurable multiplier will be used for each weight value in each layer of the DNN.

We argue that the existing mapping approaches are inadequate for the following reasons: (i) they may result in resource under-utilization due to biased decisions; (ii) they target only average accuracy, which can be misleading; and (iii) they ignore big accuracy drops on specific dataset batches.

Resource under-utilization due to biased decisions: With the term *utilization* we refer to the amount of times each distinct multiplier mode is being used in a mapping. The methods in [1,3] both perform weight-based mapping of approximation by employing the concepts of layer significance and weight magnitude. For instance, the authors in [1] presented LVRM (Low-Variance Reconfigurable Multiplier), an approximate reconfigurable multiplier that supports three operation modes: (i) $LVRM0$, which triggers exact multiplications; (ii) $LVRM1$, which introduces low error and small energy gains contrary to $LVRM0$; and (iii) $LVRM2$, which introduces greater error with larger energy gains than $LVRM1$. The method in [1] tries to initially identify which layers are more resilient to error, and map their weights entirely to the most aggressive approximate mode $LVRM2$. Then, the weights of the remaining layers are mapped first to $LVRM2$, then to $LVRM1$, and finally to $LVRM0$ based on some experimentally derived ranges around the value zero, requiring manually tuning as each layer has different weight distribution. Even though this method produces considerable energy savings, it is not scalable and results in *resource under-utilization*. By mapping complete layers to $LVRM2$, [1] introduces significant error and makes the DNN susceptible to further approximation. Thus, the utilization of the $LVRM1$ mode is reduced. As an example, their methodology on the ResNet20 and CIFAR-10 dataset, for a 0.5% accuracy drop threshold, produces a mapping where the 22% of the total multiplications is assigned to $LVRM0$, only 2% to $LVRM1$, and the vast majority of 76% to $LVRM2$. Thus, one of the approximate multiplier modes are barely utilized. Considering the sub-linear relation between induced error and energy reduction in approximate multipliers [20,132], we argue that approximating multiple layers with $LVRM1$ (i.e., moderate approximation) will deliver an energy reduction closer to linear and thus, potentially higher energy reduction than the one achieved by approximating only a few layers with $LVRM2$.

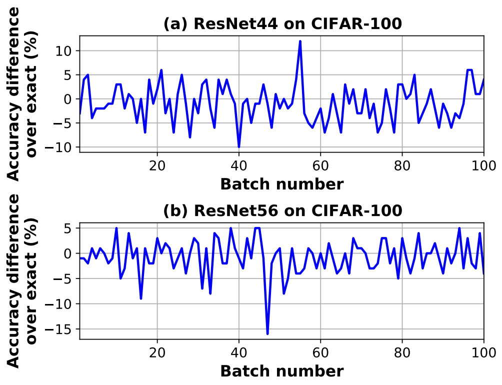

Figure 4.1: Motivational example [3] with methods (a) [5] and (b) [3].

Targeting only average accuracy can be misleading: Works that propose approximate multipliers and mapping methodologies [1, 3, 5, 21, 25], evaluate their findings on the *average accuracy* of a DNN over the target dataset. However, the introduction of error in computations does not affect all the batches of the dataset equally, creating, in many cases, great variations in the achieved accuracy. Such behavior is not acceptable when quality of service requirements need to be achieved over the entire dataset [26]. Figure 4.1(a) shows the inference accuracy differences of the ALWANN method [5] against the baseline (exact computations without error) for ResNet44 over the entire CIFAR-100 test dataset. Particularly, we split the 10,000 images into 100 equal batches and we show the accuracy differences for each one of them. Even though ALWANN had an overall accuracy drop of only 1% over the entire dataset, when we take a deeper look into the achieved accuracy per batch, we can see cases where the accuracy drops as low as 10% when compared to the exact operation (e.g., batch 40). Additionally, looking at the batches in which ALWANN achieved lower accuracy, over 20% of them have an accuracy drop of more than 5%, which is significant considering the strict constraint of 1%. Similar behavior has been observed from the mappings produced by [1]. Satisfying such fine-grain requirements could be posed in the form of *queries*. For example, "For the dataset batches that perform worse than the exact behavior, we want no more than 20% of the cases to drop more than 5% when we introduce approximation."

Ignoring big accuracy drops on specific batches: Introducing approximation can also have an additional effect. Even though the variation of the achieved accuracy can be within specific values, there might be specific batches with very low accuracy. Figure 4.1(b) shows the inference accuracy difference of the method in [3] against the baseline (exact computations without error) for ResNet56 over the entire CIFAR-100 test dataset (10,000 images split into 100 equal batches). Even though [3] achieves an overall accuracy drop of 1% and satisfies the query presented in the previous observation, we can see that for batch 47 the accuracy drop is 16%. Having such large drops during the inference phase could in some cases not be deemed acceptable, even if it is only for a small number of batches. For instance, such applications could be image recognition tasks in autonomous vehicles, where a steady stream of data is provided through different sensors (RADAR, LIDAR, etc) [26, 133]. In such cases, it is important to study the accuracy for each block of this data stream instead of evaluating an NN over the final average accuracy. Therefore, even more complex queries are needed to capture this behavior. For instance: "For the dataset batches that perform worse than the exact behavior, we want no more than 20% of the cases to drop more than 5% and no case whatsoever to drop more than 15% when we introduce approximation."

The analysis presented above shows the necessity to express complex queries in a formal way and explore solutions systematically without manual tuning. In that way, we will be able to produce flexible and scalable mappings, provide more fine-grain control of the introduced error on the overall dataset, and support different levels of quality of service.

4.2.2 Methodology

This section presents a methodology to systematically map DNN weights to approximation, under multiple and distinct objectives, using PSTL. By employing PSTL, we show that it is possible to describe more intricate properties of accelerators that comprise approximate multipliers. We additionally show that through PSTL we are allowed to formulate and solve optimization problems with respect to energy gains. The user provides an approximate multiplier, a DNN already trained and quantized to 8-bits, and a dataset. The system we target in this work is a DNN accelerator comprising MAC units that utilize the given approximate multiplier. The output of the system is a single trajectory that captures the accuracy behavior of the given DNN for each batch of the given dataset. Specifically, the trajectory captures the *accuracy drop* of the utilized approximate multiplier against the exact multiplier for each respective dataset batch. By considering this accuracy drop per batch for a given DNN as a trajectory, we can investigate more specific and fine-grain properties of the overall system and acquire more knowledge on the impact of approximate accelerators. Having defined the output trajectory of the accelerator, we can express a property query using PSTL. An example of such a query is "For a given accelerator employing a reconfigurable multiplier and a given accuracy drop threshold, what is the maximum achievable energy gain we can achieve without violating the accuracy requirement?". After expressing a PSTL query, the mapping exploration phase, also called *parameter mining phase*, is triggered. Initially, the weights of the DNN are randomly assigned to approximation modes of the given reconfigurable multiplier. The output accuracy trajectory is then analyzed for its robustness, which is then fed to a stochastic optimizer that decides on the next approximation mapping for each DNN layer. Essentially, the stochastic optimizer correlates the robustness value with per-layer approximation, and tries to find operating conditions that satisfy the defined PSTL query. The aim of the stochastic optimization step is to push the system's behavior to the constraint boundaries that are set through the PSTL queries. Once the exploration phase is completed, we build a Pareto-front of mined parameters where the PSTL query is satisfied.

4.2.2.1 Expressing system properties via Signal Temporal Logic

As aforementioned, state-of-the-art mapping methodologies require manual tuning. For example, specific layers need to be selected based on their error resilience, and then further exploration is needed on individual weight value ranges [1, 3]. These methods mostly rely on experimental observations without *systematic exploration*. How can queries like the ones described in Section 4.2.1 be posed? How can we exploit such queries to infer system properties? In order to support such fine-grain exploration and automate the mapping procedure, we utilize STL [73, 134] and specifically we build queries through PSTL [75], considering the accuracy of all the incoming batches over time as the output trajectory. From this point onward, we refer to such trajectories as *signals*.

As mentioned and presented in Section 4.1.1, STL is a specification formalism used to express the properties of a given system in a compact way. To define the quantitative semantics of STL over arbitrary predicates, robustness is utilized as a quantitative measure of a given STL formula φ. Robustness indicates how far the signal is from satisfying or violating the defined STL specification [73]. An example of an STL expression is: "Does the accuracy signal always remain above 98% accuracy". The syntax of STL comprises multiple Boolean operators but in this work, we only utilize the conjunction $\wedge$. Additional operators can be defined as syntactic abbreviations. In this work, we will be using the notion of the "always" operator [73] $\Box_{\mathcal{I}}\phi$, meaning that "always during the interval $\mathcal{I}$, ϕ should be true". We consider $\mathcal{I} = [0, \infty)$, and therefore $\mathcal{I}$ can be dropped from the notation. These STL operators have been defined and used in literature to express temporal properties [134, 135]. Moreover, we extend "always" to a more relaxed operator ${}^{X}\Box\phi$, where instead of demanding the entire specification ϕ to hold over the entire interval $\mathcal{I}$, we consider that ϕ is true if it holds just for $X\%$ of the signal values over the interval $\mathcal{I}$.

Using STL, it is possible to judge whether a signal satisfies or not a specified system property ϕ. However, it is possible to also explore more elaborate system properties that a signal satisfies. Specifically, consider the aforementioned example of the STL query "Does the accuracy signal always remain above 98% accuracy?". This query is essentially reduced to a "yes or no" problem: the signal either remains over 98% over the entire interval $\mathcal{I}$ or not. Instead of pre-defining the 98% accuracy value, we can leave it as a parameter θ to be mined. Therefore, the query can be rephrased as "Which is the lowest accuracy value θ that the signal always satisfies?". Such questions can be posed through the formal language PSTL [75]. PSTL, an extension of STL, is a formal language used to express questions when we need to further explore the properties of a system instead of just determining whether an STL specification is satisfied or not. Parameter mining is the procedure of determining parameter values for PSTL formulas for which the specification is falsified. Therefore, parameter mining answers the question of which parameter ranges cause falsification [73]. Through the robustness metric, the parameter mining problem can be posed as an optimization problem [73].

4.2.2.2 Parametric Signal Temporal Logic Queries

The employment of PSTL can assist in the exploration of DNN weight-to-approximation mappings w.r.t. to maximizing a given parameter, which in our case is the *energy savings* of the approximate accelerator. In this section, we show how we build incremental queries to capture energy gain values while keeping inference accuracy within multiple specific constraints: some of them more strict than others. Considering equal batches of our dataset as the stream of input data, we build the following initial query:

IQ1: *What is the maximum achieved energy gain θ such that, when applying approximation, any accuracy drop from the baseline is no more than $Accuracy_{thr}$%*

for X% of the time?

$$\varphi^{IQ1}[\theta] = \square(Energy_{gain} \leq \theta) \implies$$

$$^{X\%}\square(Accuracy_{diff} \leq Accuracy_{thr}\%)$$

In this initial query IQ1, values X and $Accuracy_{thr}$ are defined by the user. For instance, in the motivation example presented in Figure 4.1(a) and described in Section 4.2.1 $X = 80\%$ and $Accuracy_{thr} = 5\%$. The baseline is the achieved accuracy without applying any approximation (i.e., exact computations). The parameter θ is at this stage unknown to the user and is left to be mined. Note that, the $Accuracy_{diff}$ value refers to the *accuracy difference per batch* $accuracy_{exact} - accuracy_{approximate}$ of each target DNN. In other words, we want to make sure that for all the input batches, in which approximation results in an accuracy drop, this accuracy drop is no more than $Accuracy_{diff}$ for $X\%$ of these batches.

With the query IQ1, we are able to impose fine-grain constraints across all batches of the dataset regarding the variation of the accuracy drop. However, as we showed in Section 4.2.1, there are cases where even though the variation of the achieved accuracy is within specific values, there might be specific batches with very low accuracy. To that end we extend IQ1 as follows:

IQ2: *What is the maximum achieved energy gain θ such that, when applying approximation, any accuracy drop from the baseline is no more than $Accuracy_{thr}\%$*

for X% of the time, and no more than $Accuracy_{thr,total}$ at any time?

$$\begin{aligned}\varphi^{IQ2}[\theta] = &\Box(Energy_{gain} \leq \theta) \implies \\ &{}^{X\%}\Box(Accuracy_{diff} \leq Accuracy_{thr}\%) \land \\ &\Box(Accuracy_{diff} \leq Accuracy_{thr,total}\%)\end{aligned}$$

Again in this case, the values of X, $Accuracy_{thr}$ and $Accuracy_{thr,total}$ are defined by the user based on the needs of the application.

Finally, since many related works on approximate reconfigurable multipliers take into consideration the average accuracy of each target DNN [1, 3, 5], we add it to query IQ2 to capture both fine-grain and coarse-grain accuracy information *simultaneously*. Therefore:

IQ3: *What is the maximum achieved energy gain θ such that, when applying approximation, any accuracy drop from the baseline is no more than $Accuracy_{thr}$% for X% of the time, no more than $Accuracy_{thr,total}$ at any time, and the average accuracy drop is below $Accuracy_{thr,avg}$?*

$$\begin{aligned}\varphi^{IQ3}[\theta] = &\Box(Energy_{gain} \leq \theta) \implies \\ &{}^{X\%}\Box(Accuracy_{diff} \leq Accuracy_{thr}\%) \land \\ &\Box(Accuracy_{diff} \leq Accuracy_{thr,total}\%) \land \\ &\Box(Avg\_Accuracy\_Drop \leq Accuracy_{thr,avg}\%)\end{aligned}$$

Once again, the values of X, $Accuracy_{thr}$, $Accuracy_{thr,total}$ and this time also $Accuracy_{thr,avg}$, are defined by the user. Concluding, in this section we used the initial query IQ1 to build the more elaborate query IQ3 in an attempt to profile more intricate accuracy properties for a given DNN. In our evaluation, we present the experimental results of different versions of the aforementioned queries, with different X, $Accuracy_{thr}$, and $Accuracy_{thr,avg}$ values.

4.2.2.3 Weight-to-approximation mapping

In our methodology, we follow a layer-oriented approach to map specified multiplications to approximation. To that end, we utilize a stochastic optimizer under multiple accuracy constraints (e.g., query IQ 3), that iteratively (i) takes as input the observed accuracy per batch and the estimated output energy of the system, and (ii) produces as output a signal that describes the mapping of the DNN weights of each layer to the approximate modes of the multiplier.

To formulate the problem and without loss of generality, we assume an accelerator whose MAC units are composed of reconfigurable approximate multipliers. Each reconfigurable approximate multiplier supports three operation modes: (i) $M0$, which corresponds to the exact operation; (ii) $M1$, which introduces a small error with small energy gains when compared to $M0$; and (iii) $M2$, which aggressively introduces greater error and achieves larger energy gains than $M1$. Even though our method can be used for any number of approximation modes, we select three because previous research works have shown that the area overhead is not big and the control logic remains simple [1,3]. Supposing that a DNN consists of L layers $(l_1, l_2, \cdots, l_L)$, then the outputs of the stochastic optimizer are two signals $V^{M2} = [v_1^{M2}, v_2^{M2}, \cdots, v_L^{M2}]$ and $V^{M1} = [v_1^{M1}, v_2^{M1}, \cdots, v_L^{M1}]$, where each element in both of them is a number between 0 and 1. Each element v_i^{M2} represents the percentage of all the multiplications of layer l_i that are chosen to be mapped on the approximate mode $M2$. Similarly, v_i^{M1} is the percentage of all the multiplications of layer l_i mapped on the approximate mode $M1$. Therefore, for any layer l_i, the percentage of the total layer's multiplications, that will be mapped to mode $M0$, will be $1 - (v_i^{M1} + v_i^{M2})$. Overall, the main goal of the stochastic optimizer is to observe the behavior of the DNN, in terms of accuracy and energy consumption, correlate it with the values of V^{M2} and V^{M1}, and finally find appropriate values iteratively for the two vectors such that the constraints are satisfied.

To guide the optimizer regarding which weight values would be most likely assigned to modes $M1$ and $M2$, we assign the different approximate modes to ranges around the median value of the weights for each DNN layer. We base this decision on the fact that in most cases the values of the weights of each layer are gathered around a centered value, featuring low dispersion as shown in [3]. This way, the more aggressive $M1$ and $M2$ modes would be utilized more frequently, maximizing the potential for higher energy gains. As aforementioned in Section 3.1.1.2 and shown in Figure 3.4, weight values follow a close to normal distribution with low dispersion. From our analysis, it is observed that the vast majority of layers follow this principle having either a very distinguished peak or a more flattened behavior. However, there were no layers with multiple peaks regarding the distribution. To accommodate fine-grain mapping we follow the approach described in [1] where a 2-bit signal is used to select the multiplier mode and a control unit that comprises 4 8-bit comparators, two AND gates, and an OR gate to activate multiplier modes based on weight values (i.e. selected ranges). Overall, the hardware needed to support such fine-grain weight mapping to different multiplier modes using ranges results in a minimal area overhead of less than 3% [1]. Also, the number of control units is equal to the number of the MAC array rows and thus, it increases only linearly as the MAC array size increases (quadratically).

Figure 4.2 shows the flow of the optimization steps in the proposed framework. When the exploration is triggered on a query $\varphi^{IQ}[\theta]$, initially the V^{M1} and V^{M2} signals contain random values. The output signal of the accelerator for the executed DNN, in terms of accuracy per batch, is then analyzed for its robustness, the stochastic optimizer correlates the robustness value with per-layer approximation, and it alters the impact of the approximation through modifications on V^{M1} and V^{M2}. In each optimization

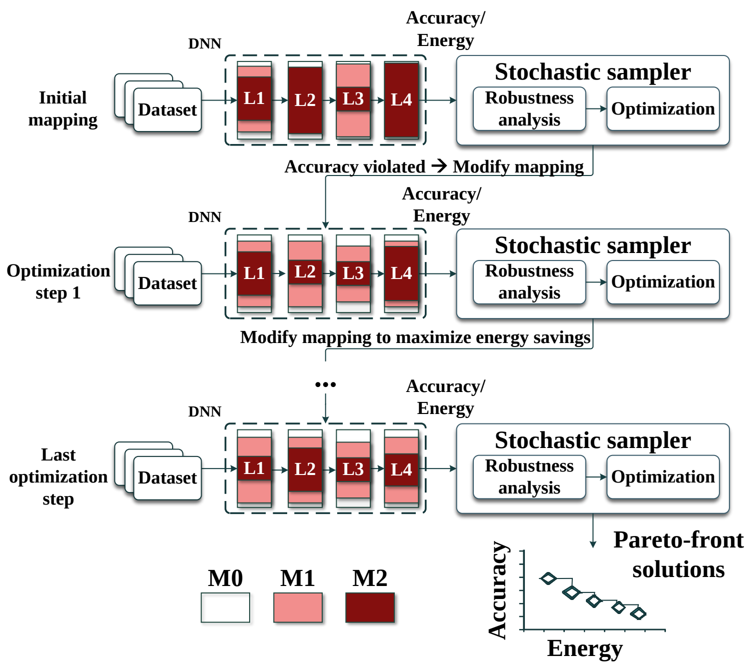

Figure 4.2: Optimization steps for mapping DNN weights to approximate modes [6].

iteration, the stochastic optimizer aims to gradually minimize the analyzed robustness of the system based on its output. Therefore, the goal of the optimizer is to eventually tweak the V^{M1} and V^{M2} signals in a way that would satisfy all the given constraints described in the query (e.g., IQ1-IQ3). Regarding the utilized stochastic optimizer, we employ the Expected Robustness Guided Monte Carlo (ERGMC) algorithm, which is based on simulated annealing and is presented in [136]. We set the number of control points to be equal to the number of convolution layers of each target DNN, and evenly distribute them. Overall, the stochastic optimizer aims to push the system's behavior as close as possible to the specified constraint boundaries. The parameter mining phase is completed after a predefined number of tests. We then generate a Pareto-front in the parameter space, resulting from all the conducted tests. What we consider to be the final output of this phase is the mapping that corresponds to the maximum value that was found for the parameter θ.

Overall, the robustness value is an indication of how "far" or "near" the signal is from satisfying the initially set accuracy requirements. The robustness is evaluated on the accuracy signal based on the PSTL requirements, and is then utilized by the stochastic optimizer to select the next approximation mappings, which lead to θ parameter ranges.

4.2.3 Results & Evaluation

We aim to showcase the strengths of the proposed method in terms of 1) efficient mappings with higher energy savings than previous methods; 2) efficient utilization of all approximate modes; 3) automatic and scalable fine-grain exploration. Regarding the energy consumption, the MAC units are described in Verilog RTL, synthesized using Synopsys Design Compiler and mapped to a 7nm technology library. Mentor Questasim is used for post-synthesis timing simulations and the switching activity of the MAC units is captured through 1 million randomly generated inputs, which is then fed to Synopsys PrimeTime to calculate power consumption. Additionally, we bridge S-TaLiRo [73], a toolbox for temporal logic falsification, with the Tensorflow machine learning library, in which we overrode the convolution layers and replaced the exact multiplications with the respective approximate ones [5]. Finally, we use the stochastic optimizer in S-TaLiRo to solve the PSTL query and acquire the approximation mappings.

We consider seven different PSTL queries and evaluate our findings against the mapping methods presented in LVRM [1] and ALWANN [5] across seven DNNs quantized to 8 bits: GoogleNet [106], MobileNetv2 [108], ResNet20 [107], ResNet32 [107], ResNet44 [107], ResNet56 [107], and ShuffleNet [109]. All the aforementioned DNNs were trained on two datasets: CIFAR-10 and CIFAR-100, each of which considers 32×32 input image size. We additionally evaluate our method on the Imagenet dataset (224×224 image size) on the following four DNNs: InceptionV3 [137] (rescales images to 299×299), MobileNet [138], NasNet [139], and VGG16 [140].

Our initial motivation was to express intricate properties of systems through PSTL and find energy-efficient approximation mappings given any approximate reconfigurable multiplier. In our evaluation, we build the Queries Q1-Q7 as shown in Table 4.1. We

Table 4.1: All considered PSTL queries where $Accuracy_{thr,avg} = \{0.5\%, 1\%, 2\%\}$.

What is the maximum achieved energy gain θ during inference such that:	
$\varphi^{Q1}[\theta] = \square(Energy_{gain} \leq \theta) \implies {}^{40\%}\square(Accuracy_{diff} \leq 3\%)\wedge$ $\square(Accuracy_{diff} \leq 15\%)\wedge$ $\square(\text{Avg\_Accuracy\_Drop} \leq Accuracy_{thr,avg})$	any per batch accuracy drop is less than 3% for 40% of the batches, and the per batch accuracy drop is less than 15% at any time, and the average accuracy drop is less than $Accuracy_{thr,avg}$.
$\varphi^{Q2}[\theta] = \square(Energy_{gain} \leq \theta) \implies {}^{60\%}\square(Accuracy_{diff} \leq 3\%)\wedge$ $\square(Accuracy_{diff} \leq 15\%)\wedge$ $\square(Avg\_Accuracy\_Drop \leq Accuracy_{thr,avg})$	any per batch accuracy drop is less than 3% for 60% of the batches, and the per batch accuracy drop is less than 15% at any time, and the average accuracy drop is less than $Accuracy_{thr,avg}$.
$\varphi^{Q3}[\theta] = \square(Energy_{gain} \leq \theta) \implies {}^{80\%}\square(Accuracy_{diff} \leq 3\%)\wedge$ $\square(Accuracy_{diff} \leq 15\%)\wedge$ $\square(Avg\_Accuracy\_Drop \leq Accuracy_{thr,avg})$	any per batch accuracy drop is less than 3% for 80% of the batches, and the per batch accuracy drop is less than 15% at any time, and the average accuracy drop is less than $Accuracy_{thr,avg}$.
$\varphi^{Q4}[\theta] = \square(Energy_{gain} \leq \theta) \implies {}^{40\%}\square(Accuracy_{diff} \leq 5\%)\wedge$ $\square(Accuracy_{diff} \leq 15\%)\wedge$ $\square(Avg\_Accuracy\_Drop \leq Accuracy_{thr,avg})$	any per batch accuracy drop is less than 5% for 40% of the batches, and the per batch accuracy drop is less than 15% at any time, and the average accuracy drop is less than $Accuracy_{thr,avg}$.
$\varphi^{Q5}[\theta] = \square(Energy_{gain} \leq \theta) \implies {}^{60\%}\square(Accuracy_{diff} \leq 5\%)\wedge$ $\square(Accuracy_{diff} \leq 15\%)\wedge$ $\square(Avg\_Accuracy\_Drop \leq Accuracy_{thr,avg})$	any per batch accuracy drop is less than 5% for 60% of the batches, and the per batch accuracy drop is less than 15% at any time, and the average accuracy drop is less than $Accuracy_{thr,avg}$.
$\varphi^{Q6}[\theta] = \square(Energy_{gain} \leq \theta) \implies {}^{80\%}\square(Accuracy_{diff} \leq 5\%)\wedge$ $\square(Accuracy_{diff} \leq 15\%)\wedge$ $\square(Avg\_Accuracy\_Drop \leq Accuracy_{thr,avg})$	any per batch accuracy drop is less than 5% for 80% of the batches, and the per batch accuracy drop is less than 15% at any time, and the average accuracy drop is less than $Accuracy_{thr,avg}$.
$\varphi^{Q7}[\theta] = \square(Energy_{gain} \leq \theta) \implies$ $\square(Avg\_Accuracy\_Drop \leq Accuracy_{thr,avg})$	the average accuracy drop is less than $Accuracy_{thr,avg}$.

constructed queries that aim to capture different levels of requirements, with some queries being more relaxed than others. Note that, there are some user-defined variables: X, $Accuracy_{thr}$, $Accuracy_{thr,total}$ and $Accuracy_{thr,avg}$. For all of the considered queries in this evaluation, $Accuracy_{thr,total}$ is set to be 15%. This $Accuracy_{thr,total}$ value was selected based on the assumption that such a big accuracy drop should be the maximum drop per batch that can be tolerated. The queries shown in Table 4.1 are split into three main parts.

Strict fine-grain constraints (Q1-Q3): For the first three queries Q1-Q3, we set the maximum acceptable accuracy drop per batch from the baseline to be $Accuracy_{thr} = 3\%$ and we set this requirement to hold for $X = \{40\%, 60\%, 80\%\}$ of these batches. Requiring this specification to hold for 40% of the batches (Q1) is a less aggressive approach compared to requiring it to hold for 80% of the batches (Q3). Overall these three queries are the strictest considered in this evaluation.

Relaxed fine-grain constraints (Q4-Q6): The next three queries Q4-Q6 in Table 4.1 are a more relaxed version of the previously analyzed Q1-Q3 queries. We variate $X\%$ in the same way, i.e., $X = \{40\%, 60\%, 80\%\}$ but this time the per batch acceptable accuracy drop threshold is larger and set to $Accuracy_{thr} = 5\%$. With queries Q4-Q6, we wanted to allow more room for aggressive approximation, by slightly increasing the acceptable accuracy drop per batch. Similar to the previous triad of queries, there is a gradient strictness as we move from query Q4 to Q6 caused by the $X\%$ value.

No fine-grain constraints (Q7): Q7 is the last query considered in this evaluation and is the most relaxed among all. We did not impose any constraints per batch and we only set the coarse-grain requirement of $Accuracy_{thr,avg}$ to hold. We included this query in our evaluation since it is the same requirement enforced by previous works, considering only average accuracy drop [1, 3, 5, 21, 25]. For all seven queries, we consider three different cases of $Accuracy_{thr,avg}$: $0.5\%, 1\%$ and 2%. The different combinations of $Accuracy_{thr}$, $Accuracy_{thr,total}$, and $Accuracy_{thr,avg}$ values provide a diverse coverage in terms of requirement strictness on DNN accuracy.

Table 4.2: Energy gains over [1] for the CIFAR-10 dataset.

	ResNet20	**ResNet32**	**ResNet44**	**ResNet56**	**GoogLeNet**	**MobileNetV2**	**ShuffleNet**
Q1	1.07	1.07	1.14	1.08	1.09	1.04	1.13
Q2	1.02	0.99	1.08	1.01	1.05	1.01	1.08
Q3	0.87	1.00	0.89	0.96	1.03	0.96	1.05
Q4	1.13	1.08	1.24	1.13	1.12	1.05	1.21
Q5	1.04	1.03	1.20	1.07	1.07	1.02	1.10
Q6	0.95	1.00	1.07	1.06	1.04	0.97	1.08
Q7	1.13	1.09	1.28	1.13	1.13	1.05	1.21

Tables 4.2 - 4.4 show the resulting evaluation against LVRM [1]. For the CIFAR-10 dataset (Table 4.2), the gains of our method over LVRM [1] are overall lower than the ones achieved for the other datasets. This is attributed to the fact that CIFAR-10 is overall an "easy" dataset that comprises a number of 10 classes in total. Therefore, LVRM can achieve energy-efficient mappings that leave little room for improvement; however, the proposed method can exploit this tight margin to produce even better solutions in terms of energy gains. The even more difficult 100-class dataset CIFAR-100 (Tables 4.3) shows the biggest energy gains overall. The hardest dataset evaluated in this work is the ImageNet which comprises 1000 classes and shows the highest gains over LVRM overall. This is attributed to the fact that LVRM is mainly targetting mapping

Table 4.3: Energy gains over [1] for the CIFAR-100 dataset.

	ResNet20	**ResNet32**	**ResNet44**	**ResNet56**	**GoogLeNet**	**MobileNetV2**	**ShuffleNet**
Q1	1.08	1.07	1.18	1.13	1.10	1.10	1.11
Q2	0.99	1.08	1.07	1.05	1.05	1.03	1.07
Q3	0.93	0.96	0.87	0.95	0.80	1.01	0.99
Q4	1.14	1.13	1.22	1.16	1.12	1.10	1.18
Q5	1.03	1.08	1.13	1.13	1.12	1.05	1.10
Q6	0.97	1.01	1.07	1.02	1.02	0.99	1.02
Q7	1.18	1.16	1.24	1.17	1.20	1.13	1.18

Table 4.4: Energy gains over [1] for the ImageNet dataset.

	0.5%	**1%**	**2%**	**0.5%**	**1%**	**2%**	**0.5%**	**1%**	**2%**	**0.5%**	**1%**	**2%**
	Inceptionv3			**NasNet**			**Mobilenet**			**VGG16**		
Q1	1.01	1.29	1.29	1.31	1.48	1.13	0.94	1.04	1.04	1.25	1.41	1.16
Q2	0.92	0.89	1.05	0.67	0.80	0.69	0.63	1.00	1.00	0.95	1.08	0.84
Q3	0.61	0.51	0.51	0.57	0.54	0.69	0.60	0.52	0.58	0.54	0.54	0.42
Q4	1.65	1.29	1.29	2.55	2.40	1.83	1.26	1.10	1.10	1.60	1.60	1.24
Q5	1.37	0.99	1.05	1.63	1.69	1.29	1.07	1.03	1.03	1.45	1.56	1.21
Q6	1.38	0.96	0.96	1.05	0.92	1.08	0.99	0.95	0.95	1.23	1.54	1.19
Q7	1.65	1.29	1.29	2.55	2.40	1.83	1.26	1.10	1.10	1.60	1.60	1.24

entire convolutional layers to the most aggressive multiplier mode, which can lead to very pessimistic solutions. For instance, if mapping the two most error-resilient layers to the most aggressive multiplier mode violated the accuracy threshold, only the most error-resilient layer will be mapped to this mode entirely. For the rest of the layers, only ranges of weights are being examined for approximation mappings, which can provide very low final energy gains. Note that, in Tables 4.2 - 4.4 the green color indicates higher energy gains across all considered DNNs for each respective query and underlying average accuracy thresholds.

Based on the findings presented above, our *methodology allows us to find mappings under multiple and fine-grain constraints, being also more energy-efficient than the state-of-art. Additionally, due to the formalization and the usage of PSTL, it is easy to create new queries and automate the search process (increased scalability), avoiding manual tuning.*

We compare our proposed methodology against ALWANN [5], a layer-oriented method where a different static approximate multiplier from [20] is mapped to a DNN

Table 4.5: Energy gains over [5] for the CIFAR-10 dataset.

	ResNet20	**ResNet32**	**ResNet44**	**ResNet56**	**GoogLeNet**	**MobileNetV2**	**ShuffleNet**
Q1	1.42	1.53	1.45	1.11	1.14	1.31	1.37
Q2	1.11	1.31	1.07	1.04	0.95	1.07	1.13
Q3	0.82	1.03	0.87	0.98	0.88	0.89	0.97
Q4	1.53	1.73	1.58	1.41	1.26	1.38	1.43
Q5	1.23	1.41	1.17	1.09	1.13	1.22	1.16
Q6	0.96	1.40	1.06	1.03	1.11	1.05	1.03
Q7	1.59	1.77	1.59	1.45	1.31	1.42	1.46

Table 4.6: Energy gains over [5] for the CIFAR-100 dataset.

	ResNet20	**ResNet32**	**ResNet44**	**ResNet56**	**GoogLeNet**	**MobileNetV2**	**ShuffleNet**
Q1	1.44	1.15	1.72	1.35	1.47	1.46	1.16
Q2	1.23	0.67	1.23	1.07	1.24	1.33	1.00
Q3	1.05	0.61	0.96	0.90	0.71	0.86	0.90
Q4	1.88	1.18	1.76	1.44	1.55	1.50	1.27
Q5	1.23	0.75	1.50	1.28	1.34	1.39	1.04
Q6	1.13	0.74	1.05	1.07	0.97	0.98	0.94
Q7	2.05	1.33	1.91	1.49	1.59	1.53	1.33

Table 4.7: Energy gains over [5] for the ImageNet dataset.

	0.5%	**1%**	**2%**	**0.5%**	**1%**	**2%**	**0.5%**	**1%**	**2%**	**0.5%**	**1%**	**2%**
	Inceptionv3			**NasNet**			**Mobilenet**			**VGG16**		
Q1	1.60	2.00	2.00	1.06	1.22	1.39	1.42	1.56	1.56	1.34	1.16	1.14
Q2	1.00	1.00	1.25	0.88	0.87	1.04	0.94	1.17	1.17	1.15	1.09	1.06
Q3	0.67	0.67	1.15	0.72	0.87	0.96	0.81	0.78	0.84	0.62	0.63	0.91
Q4	1.89	2.47	2.47	1.06	1.39	1.39	1.68	1.62	1.62	1.44	1.31	1.40
Q5	1.10	1.25	1.25	1.02	1.05	1.05	1.05	1.24	1.24	1.24	1.16	1.14
Q6	1.00	1.10	1.25	0.72	1.04	1.04	0.87	0.84	1.10	1.15	0.93	0.91
Q7	2.00	2.47	2.61	1.06	1.44	1.44	1.68	1.62	1.75	1.55	1.31	1.67

layer. The authors consider a heterogeneous platform of set tiles and employ a multi-objective genetic algorithm to map multipliers to each tile. Even though ALWANN does not propose a reconfigurable multiplier, the tile-based architecture allows the existence of multiple approximate multipliers at the same time. In our evaluation, we consider the number of multipliers per tile to be 3. For the evaluation, we:

- Generated layer-to-approximation mappings following the ALWANN procedure to select the multipliers. ALWANN requires a constraint regarding the average accuracy achieved over the dataset, and thus we set $Accuracy_{thr,avg} = \{0.5\%, 1\%, 2\%\}$ for each DNN.
- We used the same approximate multipliers selected by ALWANN under our proposed mapping framework, performed the proposed exploration and generated different mappings per DNN, dataset and $Accuracy_{thr,avg}$ value.

Tables 4.5 - 4.7 show the resulting evaluation against ALWANN [5]. The mappings produced by the proposed method were significantly higher in terms of energy gains, since ALWANN is a layer-based mapping approach, while the proposed method is weight-based. Specifically for the ImageNet dataset (Table 4.7), the multipliers selected by ALWANN were some of the least aggressive ones available [20] to satisfy the average accuracy constraints. Note that again, in Tables 4.5 - 4.7 the green color indicates higher energy gains across all considered DNNs for each respective query and underlying average accuracy thresholds.

4.3 AUTOMATED ENERGY-EFFICIENT DNN COMPRESSION UNDER FINE-GRAIN ACCURACY CONSTRAINTS

This work introduces a framework for compressing deep neural networks (DNNs) by utilizing both pruning and quantization techniques at the same time. Similar to the approach presented in Section 4.2, this framework also takes fine-grained accuracy properties into account and leverages STL and falsification to create compressed DNNs.

4.3.1 Methodology

The motivation that drove the development of this framework was inspired by the observations that were presented in Section 4.2.1. Similar to approximate multiplier-related works, research targeting DNN compression [27, 30] also focuses on Top-1 and/or Top-5 average accuracy. Contrary to related works, we consider the inference accuracy to be a trajectory with elements $Acc_{compressed,i} - Acc_{baseline,i}$ where $i \in [1, batches]$. By following such an approach, we can further examine how a compressed DNN behaves during inference for each incoming batch, while also aiming for a specific average accuracy target. For simplicity purposes, we consider image classification tasks and DNNs, even though the proposed method can be applied to any type of DNN.

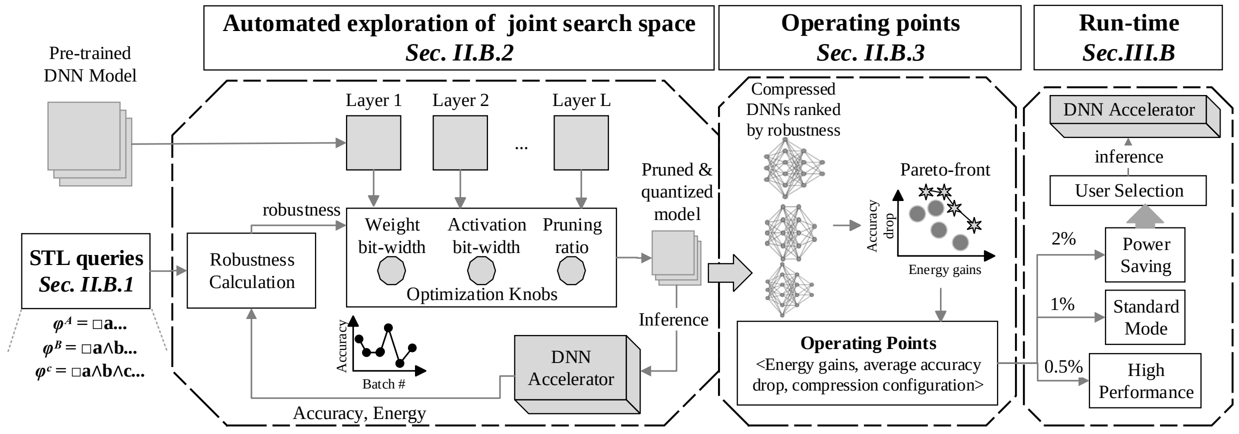

Figure 4.3: The execution flow of the proposed framework on automated compression

The target of the proposed framework is to use formal properties to jointly and systematically explore the pruning and quantization search space, automatically generating DNNs that satisfy specific performance constraints without additional retraining or fine-tuning. Figure 4.3 provides an overview of the proposed framework, presented in detail in this section. First, we define the STL queries that formally express the fine-grain accuracy properties each given DNN must satisfy. Each layer of a given pre-trained DNN model gets assigned a weight bit-width, an activation bit-width, and a pruning ratio, and the resulting compressed DNN is evaluated for its inference accuracy and energy consumption. The inference accuracy is analyzed for its robustness, which is calculated based on the initially defined STL specification. A stochastic optimization process is guided by the robustness value to select the next weight bit-width, activation bit-width, and pruning ratio values for each layer. Afterward, all the generated compressed DNNs are ranked based on their robustness, and we extract operating points based on the DNN compression configurations that lie on the Pareto-front. We further filter these configurations to create three different modes of operation that enable the end user to control the quality of DNN inference at run-time.

4.3.1.1 Capturing Accuracy Properties with Signal Temporal Logic

As aforementioned, in this work, we intend to explore more accuracy properties during DNN inference contrary to state-of-art where only average accuracy matters [27, 30]. We achieve this by treating the inference accuracy as an *output trajectory*, where each of its points is the achieved accuracy per batch. To directly derive information about the inference accuracy of a compressed model, we examine the accuracy drop per batch when compared to the initial DNN model. Specifically, each output trajectory point corresponds to $Acc_{DNN_c,i} - Acc_{DNN_b,i}$ where $Acc_{DNN_c,i}$ is the compressed model's accuracy for batch i and $Acc_{DNN_b,i}$ is the baseline model's accuracy for batch i. By incorporating more fine-grain accuracy information in the exploration procedure of the proposed framework, we *avoid additional time-consuming retraining and/or fine-tuning. At the same time, we guarantee a compression combination of pruning and quantization that can satisfy tight accuracy constraints.*

By considering DNN inference as a trajectory, we can use STL (presented in Section 4.1.1), where, as aforementioned, the metric of *robustness* is utilized to quantify how near or far a trajectory is from violating a system requirement φ. STL requirements are posed using combinations of Boolean and temporal operators. In this work, we will be using the notions of conjunction $\wedge$, and the "always" operator $\square\phi$ which means that the STL specification ϕ should always hold in the entirety of the trajectory. We also use the "relaxed always" operation ${}^{X}\square\phi$, where ϕ is allowed to hold just for $X\%$ of the trajectory points [6].

We construct a variety of STL queries in an attempt to generate as many diverse compression combinations as possible. We start with the initial query φ^A and build more elaborate queries on top of it.

A: *The average accuracy drop of the compressed DNN_c network should not fall below a predefined drop threshold from the average accuracy of the baseline DNN_b network.*

$$\varphi^A = \square(Acc_{DNN_c,avg} \geq Acc_{DNN_b,avg} - Acc_{thr,avg})$$

Query φ^A expresses a constraint commonly found in related works that only targets a specific average accuracy threshold [4, 27, 30]. To fully utilize the ability to systematically explore DNN inference accuracy properties, we augment query φ^A with additional requirements.

B: *The average accuracy drop of the compressed DNN_c network should not fall below a predefined drop threshold from the average accuracy of the baseline DNN_b network.* ***Additionally****, the maximum number of images that can be misclassified in each batch is I_{max}.*

$$\varphi^B = \square(Acc_{DNN_c,avg} \geq Acc_{DNN_b,avg} - Acc_{thr,avg}) \wedge$$
$$\square(Acc_{DNN_c,batch} - Acc_{DNN_b,batch} \leq \frac{I_{max}}{batch_{size}})$$

In query φ^B we add the requirement of *per-batch additional misclassification.* Note that, we count the misclassification occurrences when compared to the baseline DNN_b network. Therefore, if $I_{max} = 5$ and DNN_b for a random batch misclassifies 2 images and the compressed DNN_c network misclassifies 3 images for the same batch, the requirement *is not violated.* With this requirement, we aim to eliminate big per-batch accuracy drops that would severely deteriorate the quality of run-time inference. Finally, we further augment φ^B.

C: *The average accuracy drop of the compressed DNN_c network should not fall below a predefined drop threshold from the average accuracy of the baseline DNN_b network.* ***Additionally****, the maximum number of images that can be misclassified in each batch is I_{max}.* ***Additionally****, there cannot be image misclassifications for more than $X\%$ of the entire inference duration.*

$$\begin{aligned}\varphi^C = \;&\square(Acc_{DNN_c,avg} \geq Acc_{DNN_b,avg} - Acc_{thr,avg}) \wedge \\ &\square(Acc_{DNN_c,batch} - Acc_{DNN_b,batch} \leq \frac{I_{max}}{batch_{size}} \wedge \\ &{}^{X\%}\square(Acc_{DNN_c,batch} - Acc_{DNN_b,batch} \geq 0)\end{aligned}$$

In this query, we refine the final accuracy requirements by combining queries φ^A and φ^B to demand a certain quality of run-time inference. For instance, if $X = 70\%$ for 100 batches, then we can have image misclassifications occur only in 30 of the total batches. Again, note that we count these misclassifications *on top of the misclassifications of the baseline DNN_b network.* The values of $Acc_{thr,avg}$, I_{max}, and $X\%$ are selected based on the demanded quality of DNN inference.

4.3.1.2 DNN compression through Falsification

In this section, we describe how we combine pruning and quantization on given pre-trained DNNs, without violating a given STL requirement. At this point we want to note that, the goal of this work is not to propose a new pruning or quantization technique, but to combine them in an automated framework that produces energy-efficient compressed DNNs through the exploration of the joint search space. Any existing pruning and quantization algorithm can be integrated into our framework. To automatically generate compression combinations that satisfy STL requirements, we use falsification. Falsification is the process that aims to find violating behaviors of a given STL specification through stochastic optimization that utilizes the robustness metric [31, 136, 141].

Assuming (i) a given pre-trained DNN consisting of L convolution layers, and (ii) given accuracy specifications, we characterize each layer $l_i, \forall i \in [1, L]$ with three different properties: (a) weight bit-width W_{l_i}, (b) activation bit-width A_{l_i}, and (c) pruning ratio PR_{l_i}. We want to jointly combine all of this information for each DNN layer concisely. Therefore, for any DNN that comprises L layers, we formulate its properties through three different vectors: $V_{DNN,w} = [W_{l_1}, ..., W_{l_L}]$, $V_{DNN,a} = [A_{l_1}, ..., A_{l_L}]$ and $V_{DNN,pr} = [PR_{l_1}, ..., PR_{l_L}]$. These three vectors are the *compression configuration* of the DNN. Vectors $V_{DNN,w}$ and $V_{DNN,a}$ can have values ranging within predefined lower and upper bit-width limits. Accordingly, vector $V_{DNN,pr}$ can have values in the range $[0, 0.9]$. The falsification procedure comprises the following steps:

Step 1: Initially, all the element values of vectors $V_{DNN,w}$, $V_{DNN,a}$ and $V_{DNN,pr}$ are random. The lower the selected values for $V_{DNN,w}$ and $V_{DNN,a}$, the larger the energy gains of the DNN inference. Respectively, large values for the pruning ratios in $V_{DNN,pr}$ are expected to yield elevated gains in energy consumption. The initially random output compression configuration is applied on the given DNN, and after the inference phase on the DNN accelerator, its accuracy trajectory and output energy consumption are monitored.

Step 2: After the compressed DNN inference, the respective output accuracy trajectory is analyzed for its robustness based on the given STL requirement. A positive value of robustness indicates satisfaction of the STL requirement, while a negative value indicates a violation.

Step 3: The robustness value from Step 2 is used as a guide in a stochastic optimizer that selects the next values of vectors $V_{DNN,w}$, $V_{DNN,a}$ and $V_{DNN,pr}$. We utilize the stochastic Nelder-Mead algorithm [142] but note that any minimization algorithm can be employed in this part of the framework. At first, the vector values selected by the optimizer are expected to yield negative robustness results, but with each iteration the robustness value is correlated with the compression configuration, leading eventually to acceptable results.

Steps 1-3 constitute the *falsification procedure* which is repeated for a pre-determined amount of iterations. Through falsification, we aim to find robustness values close to zero in order to push the compressed DNN accuracy toward the requirement boundaries. This way, the generated compression configurations will satisfy the initial set requirements, while achieving gains in energy consumption.

4.3.1.3 Creating operating points

The falsification loop described in Section 4.3.1.2 results in the generation of multiple compression configurations. However, we are only interested in keeping the configurations that are correlated with non-negative robustness values. To that end, we construct a Pareto-front from the results of the falsification process with respect to the average accuracy drop and the energy gains monitored in Section 4.3.1.2. For each combination that lies on the Pareto-front, we construct operating points that contain the following information: (a) the associated energy gains, (b) the average accuracy drop $Acc_{DNN_c,avg}$, and (c) the compression configuration. We further filter these operating points to achieve run-time control of DNN inference. We vary $Acc_{thr,avg} \in \{0.5\%, 1\%, 2\%\}$ and offer three different modes of operation to the end user shown in Figure 4.3, namely:

(a) *High Performance mode (HP)*: When HP is selected, the configuration that corresponds to the highest energy savings value is selected, with $Acc_{thr,avg} = 0.5\%$.

(b) *Standard mode (SM)*: Initial mode of operation. User can switch back to SM at any given time. The configuration with the highest energy savings is selected with the accuracy constraint: $Acc_{thr,avg} = 1\%$.

(c) *Power Saving (PS)*: This mode expands the accuracy drop tolerance threshold, aiming to offer the ability to maximize energy gains. The accuracy constraint is $Acc_{thr,avg} = 2\%$.

By setting $Acc_{thr,avg} \in \{0.5\%, 1\%, 2\%\}$ we, therefore, enable the co-existence of *three different user modes of operation*. Allowing for interactive user input requests that affect the quality of run-time performance (in this case DNN inference accuracy) is common in systems that are powered by batteries such as smartphones [9]. Each one of these three final configurations is stored in memory and can be loaded at run-time based on the user input command.

4.3.2 Results & Evaluation

We evaluate the proposed framework on the ImageNet dataset and compare our results with the state-of-the-art works HAQ [27] and ANNC [30]. The DNNs chosen for the evaluation are the ResNet-18, ResNet-50, VGG-11, VGG-16, AlexNet, MobileNetV2, and SqueezeNet, which are all trained through PyTorch. As a baseline for the accuracy and energy consumption shown in our experiments, we consider the 8-bit quantized equivalent of each DNN. We evaluate the proposed framework on an Eyeriss-based accelerator [7] and the state-of-the-art spatial ASIC DNN accelerator BitFusion [143]. Regarding pruning, we employ the filter pruning in [77], while for quantization we use the post-training method in [144]. Note that, any pruning and quantization algorithm can be used as part of the falsification-compression procedure since each compression configuration will always be evaluated on the output accuracy. The optimization procedure described in Section 4.3.1.3 is repeated for 100 iterations.

We evaluate our work on query φ^C shown in Section 4.3.1.1, considering a batch size of 16 and evaluating different queries φ^{E_i} with the following constraints:

(a) φ^{E_1}: for each batch at most 25% of the images can be misclassified and therefore, $I_{max} = 4$. For the entirety of the inference duration, there will be no additional misclassifications on top of the baseline ones for 70% of the time and thus, $X = 70\%$.

(b) φ^{E_2}: $I_{max} = 5$ and $X = 60\%$ (more relaxed than φ^{E_1})

(c) φ^{E_3}:$I_{max} = 3$ and $X = 80\%$ (more strict than φ^{E_1})

As aforementioned in Section 4.3.1.3, we vary $Acc_{thr,avg} \in \{0.5\%, 1\%, 2\%\}$ Even though we consider a batch size of 16, the proposed approach is scalable to any desired number of batch sizes as long as the values for $Acc_{thr,avg}$, I_{max}, and $X\%$ change accordingly. For instance, for a batch size of 1, which is commonly used in edge devices, I_{max} should be removed since there can only be 0 or 1 miss-classification per batch.

Table 4.8: Query evaluations on the ImageNet dataset for $Acc_{thr,avg} = 1\%$

		$\varphi^{E_1}_{1\%}$	$\varphi^{E_2}_{1\%}$	$\varphi^{E_3}_{1\%}$
ResNet-18	Acc Drop	0.56%	0.8%	0.66%
	Energy Gains	21.65%	27.41%	18.85%
ResNet-50	Acc Drop	0.74%	0.78%	0.53%
	Energy Gains	15.5%	18%	11.77%
VGG-11	Acc Drop	0.63%	0.93%	0.57%
	Energy Gains	10.72%	13.87%	8.08%
VGG-16	Acc Drop	0.56%	0.84%	0.78%
	Energy Gains	33.03%	36.86%	21.38%
AlexNet	Acc Drop	0.95%	0.97%	0.69%
	Energy Gains	15.51%	16.32%	9.44%
MobileNetV2	Acc Drop	0.72%	1%	0.62%
	Energy Gains	11.2%	14.45%	6.74%
SqueezeNet	Acc Drop	0.73%	0.89%	0.7%
	Energy Gains	9.85%	10.61%	6.54%

Table 4.8 shows the conducted evaluation on the three queries φ^{E_1}, φ^{E_2} and φ^{E_3} for the average accuracy threshold $Acc_{thr,avg} = 1\%$ on Eyeriss [7]. As expected, in all cases the relaxed query φ^{E_2} achieves the highest gains in energy, and the strict φ^{E_3} query the

lowest. On average, φ^{E_1} achieved 16.78%, φ^{E_2} 19.65%, and φ^{E_3} 11.83% in energy savings across all DNNs. Specifically, in the case of the VGG-16 DNN the gains achieved by the strict φ^{E_3} query are more than 11% lower than the ones achieved by φ^{E_1} and φ^{E_2}, however, they are still significant (21.38%). The proposed framework was able to produce compression configurations under all accuracy constraints imposed by the φ^{E_i} queries, both strict and relaxed.

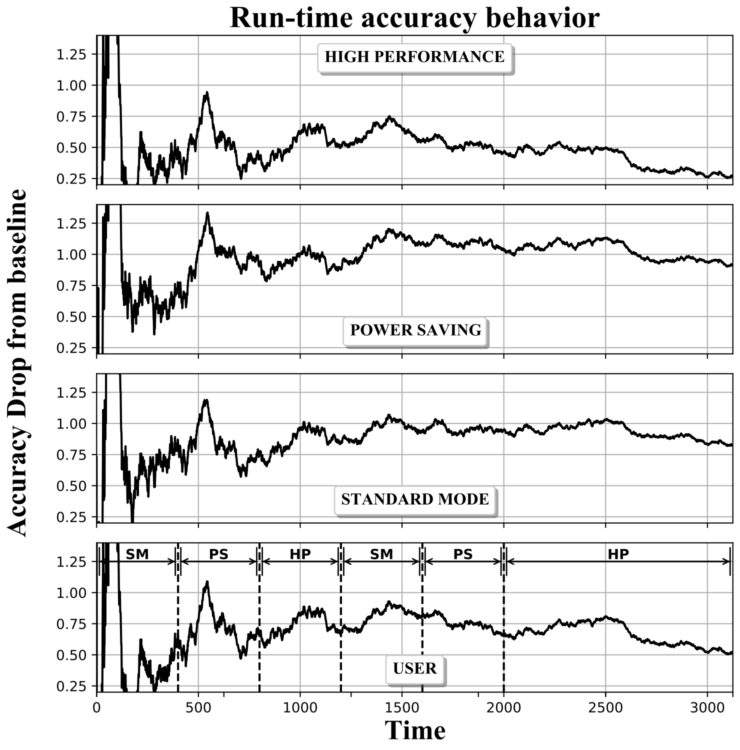

Figure 4.4: ResNet-18 on ImageNet: run-time inference on Eyeriss [7].

Figure 4.4 shows an example of the usage of different operating modes for the ResNet-18 where the DNN inference is executed on Eyeriss [7]. Each time instance represents a batch of 16 incoming images. At each time instance, we calculate the *accumulated accuracy* up to that point and show the drop from the baseline for each operating mode. The mode of operation is being changed every 400 batches except at the very end where the HP mode is selected for the last 724 batches. By combining different modes, we can achieve a run-time behavior that is completely customized to the end user. In this example, the HP mode achieves an average of 0.48% accuracy drop from the baseline, PS 1.08%, SM 0.89%, and the custom user scenario 0.67%. For the entire inference phase, the energy gains from just the HP mode would be 13.7%, the SM mode 21.65%, and the PS mode 38.38%. With this user combination, the energy savings are 23.3%. Consequently, it can be observed that custom combinations of operating modes can achieve balanced solutions that outperform the original operating modes in terms of energy savings and accuracy drop.

We compared our method against HAQ [27] and ANNC [30]. HAQ [27] only targets mixed precision quantization without considering pruning in the compression process, while ANNC [30] considers both the weight bit-width and the pruning ratio as the optimization knobs. Contrary to both of these works, the proposed framework does not require any retraining or fine-tuning. Without any retraining involved, neither of HAQ [27] nor ANNC [30] were able to produce solutions with an accuracy drop of less than 15%. Therefore, we only included part of the retraining required in our evaluation, since the initially required retraining is time-consuming. For instance, it took more than 2 days for ANNC [30] to produce a solution for AlexNet on a server with two Tesla V100S PCIe 32GB GPUs.

Table 4.9: Comparison of the proposed framework on the ImageNet dataset.

				PROPOSED FRAMEWORK					
		HAQ Eyeriss	ANNC Eyeriss	φ^{E_1} SM Eyeriss	φ^{E_1} HP Eyeriss	φ^{E_1} PS Eyeriss	φ^{E_1} SM BitFusion	φ^{E_1} HP BitFusion	φ^{E_1} PS BitFusion
ResNet-18	Acc Drop	6.45%	5.16%	0.56%	0.24%	1.97%	0.95%	0.44%	1.05%
	Energy Gains	36.25%	38.5%	21.65%	13.7%	38.38%	17.85%	12.69%	28.11%
ResNet-50	Acc Drop	8.67%	6.42%	0.74%	0.37%	1.32%	0.62%	0.41%	1.18%
	Energy Gains	30.76%	27.65%	15.5%	14.2%	19.73%	14.79%	9.09%	16.56%
VGG-11	Acc Drop	5.75%	4.5%	0.63%	0.39%	1.91%	0.95%	0.47%	1.3%
	Energy Gains	35.86%	34.69%	10.72%	8.51%	36.23%	9%	8.1%	27.06%
VGG-16	Acc Drop	4.85%	4.48%	0.56%	0.48%	1.17%	0.78%	0.34%	1.52%
	Energy Gains	35.16%	27.22%	33.03%	21.38%	34.36%	14.3%	12.36%	28.04%
AlexNet	Acc Drop	8.76%	4.2%	0.95%	0.45%	1.15%	0.85%	0.49%	1.09%
	Energy Gains	33.11%	12.48%	15.51%	13.65%	22.23%	13.82%	12.84%	22.2%
MobileNetV2	Acc Drop	5.21%	4.83%	0.72%	0.36%	1.38%	0.63%	0.36%	1.6%
	Energy Gains	32.45%	7.65%	11.2%	9.68%	25.15%	10.45%	7.78%	13.57%
SqueezeNet	Acc Drop	7.65%	6.81%	0.73%	0.46%	1.89%	0.63%	0.11%	1.69%
	Energy Gains	11%	8.73%	9.85%	8.66%	11.47%	7.54%	5.54%	8.71%

Table 4.9 shows the energy gains and accuracy drop achieved by our framework for query φ^{E_1} and all three modes of operation SM, HP, and PS against HAQ [27] and ANNC [30] on Eyeriss [7]. The accuracy drop from the 8-bit baseline is always more than 4% for the compressed DNNs produced by HAQ [27] and ANNC [30], which is expected since we do not perform the necessary retraining to its entirety due to its time cost. HAQ [27] achieves an average of 30% and ANNC [30] 22.5% in energy gains. The energy gains achieved by the proposed framework were 16.78%, 12.83%, and 26.8% for the SM, HP, and PS modes respectively, with the accuracy requirements always satisfying the strict constraints described in Section 4.3.1.1. For the MobileNetV2 and SqueezeNet DNNs, our framework generated acceptable accuracy solutions with significant gains in energy, despite their inherent already compressed structure. Again, we want to emphasize that these strict accuracy constraints are being satisfied *without the need for retraining and fine-tuning.*

We also evaluate our framework on the BitFusion accelerator [143] which can support multiplications of 2, 4, and 8 bits. We tweak the optimization procedure and round the stochastic optimizer selections to the nearest bit-width: 2, 4 or 8. No other alteration in the framework is needed to adapt to different hardware accelerators. The results of this evaluation are shown in the last three columns of Table 4.9. Overall, the proposed framework achieves 12.54%, 9.77%, and 20.61% in energy gains for the SM, HP, and PS modes for query φ^{E_1} respectively. It can be observed that the gains on BitFusion [143] are smaller than the ones on Eyeriss [7] since the bit-width combinations are not as flexible. The most commonly selected bit-widths were 4 and 8, since 2 severely deteriorates accuracy.

CHAPTER 5
RESOURCE MANAGEMENT ON HETEROGENEOUS EMBEDDED SYSTEMS

Chapter 3 introduced two frameworks that achieved energy savings on hardware accelerators through the utilization of approximate multipliers and the design of the accelerators themselves. Chapter 4 presented frameworks that rely on approximation techniques to achieve energy savings through the systematic exploration of the solution space. This chapter will present two resource management techniques on heterogeneous embedded systems that divert from approximate computing concepts. The first framework that will be presented, utilizes formal property exploration to select the operating frequency in a clustered multi-processor system. The second framework presents a resource management methodology for the optimization of the resources on a heterogeneous edge device that alters the frequencies of the CPU, GPU, and memory.

5.1 EFFICIENT RESOURCE MANAGEMENT OF CLUSTERED MULTI-PROCESSOR SYSTEMS THROUGH FORMAL PROPERTY EXPLORATION

The framework presented in this section infers the ranges of parameters for specifications under different types of workloads. Particularly, we present a unified framework that utilizes PSTL to express run-time constraints and mine multiple parameters in order to create optimized system- and application-aware operating points. These operating points are used to enhance the run-time resource manager in order to trigger DVFS efficiently and satisfy system, user, and throughput requirements at the same time. To the best of our knowledge, this is the first work that couples the concept of PSTL and run-time resource management on CMPs.

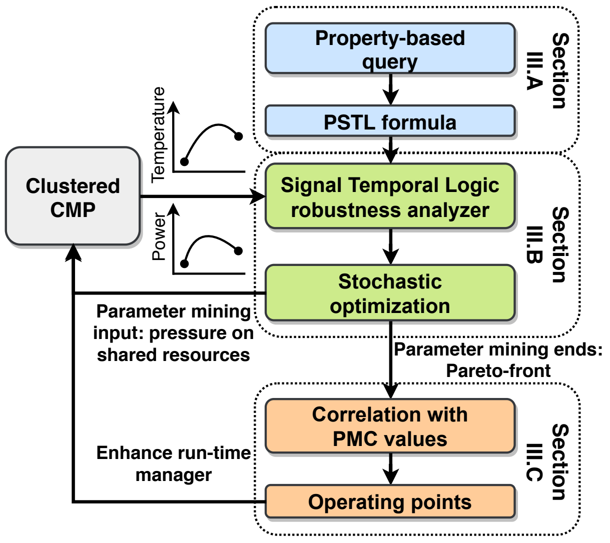

Figure 5.1: Proposed framework methodology [2]

5.1.1 Methodology

Figure 5.1 presents an overview of the proposed framework, which consists of three steps. **Step 1:** Express a property-based query using Parametric Signal Temporal Logic (PSTL) and define the output trajectories (e.g., power, temperature, throughput) to monitor and analyze. Examples of such queries are "What is the maximum power threshold and overall system throughput we can achieve while maintaining thermal safety", or "What is the maximum power threshold and overall system throughput so that the application performance is satisfied while maintaining thermal safety". **Step 2**: Once the query has been expressed in PSTL, we trigger the parameter mining phase. Initially, a random workload is assigned to each core, utilizing a developed library of micro-benchmarks, while the output trajectories are monitored by an analyzer. Then, based on the robustness of the trajectories, a stochastic sampler selects a new workload for each core, trying to find operating conditions that falsify the query. In the end, we get the Pareto-front in the parameter space for which the requirement is guaranteed to not be satisfied. Note that, frequency is not a part of the search space. Step 2 is repeated for all the frequencies in the range 1.4 - 2.0GHz. **Step 3**: The mined values of the Pareto-front are used to build run-time operating points and replace static power thresholds. In that way, the manager decides at run-time the power budget and based on the current status of the platform adjusts the operating frequency accordingly.

We utilize PSTL (overviewed in Section 4.1.2) to systematically infer the constraint parameters of clustered CMPs in order to increase system power efficiency (coarse-grain optimization) and user-defined throughput objectives (fine-grain optimization). To that end, we consider three levels of objectives such as *system*, *application*, and *throughput* and two queries to mine parameters for. *System* is the primary objective which ensures that the power consumption will keep the temperature under a safe value $Temp_{safe}$. This objective is of high priority in order to preserve thermal safety. The *application* is the secondary objective which targets the user experience. In modern embedded systems, users require specific applications to achieve a certain degree of what is considered acceptable performance. Finally, *throughput* is a tertiary objective and targets a minimum throughput for the overall platform in order to avoid starvation for background processes and services running on an embedded device.

The first query (Q1) targets the optimization of the power efficiency of the system while avoiding system under-utilization. The second query (Q2) takes into consideration specific application requirements imposed by the user:

Q1: *For a thermal safety temperature value $Temp_{safe}$, what is the maximum achievable power P_{thr} and maximum overall system throughput Th_{thr} before facing temperature issues?*

$$\varphi^{Q1}[\theta] = \varphi_1^{Q1}[\theta_1] \wedge \varphi_3^{Q1}[\theta_3] \tag{5.1}$$

where

$$\varphi_1^{Q1}[\theta_1] = \Box(Power \leq \theta_1)$$

$$\varphi_3^{Q1}[\theta_3] = \Box(\text{Throughput} \leq \theta_3) \implies$$

$$\Box(\text{Temp} \leq Temp_{\text{safe}})$$

Q2: *For a given user application and a thermal safety temperature value $Temp_{safe}$, what is the maximum achievable power P_{thr}, maximum user application performance $Perf_{thr}$, and maximum overall system throughput Th_{thr} before facing temperature issues?*

$$\varphi^{Q1}[\theta] = \varphi_1^{Q1}[\theta_1] \wedge \varphi_2^{Q1}[\theta_2] \wedge \varphi_3^{Q1}[\theta_3] \tag{5.2}$$

where

$$\varphi_1^{Q1}[\theta_1] = \Box(Power \leq \theta_1)$$

$$\varphi_2^{Q1}[\theta_2] = \Box(\text{Perf}_{app} \leq \theta_2)$$

$$\varphi_3^{Q1}[\theta_3] = \Box(\text{Throughput} \leq \theta_3) \implies$$

$$\Box(\text{Temp} \leq Temp_{\text{safe}})$$

In both queries, θ is the vector of parameters to be inferred, while temperature ($^{\circ}C$), power (W), application performance (Instructions per Second (IPS)) in Q2, and system throughput (total IPS) are the output trajectories. As shown in [73], each of the sub-queries $\varphi_i^{Qi}[\theta]$ could be explored independently by formulating an optimization problem over the corresponding single parameter θ_i. In this work, we utilized the software toolbox S-TaLiRo [131], which automatically formulates and solves the optimization problem for a given parametric specification $\varphi[\theta]$.

In CMPs, cores are not fully independent processors. Simultaneous executing applications fight for shared resources such as memory controller and last-level cache, significantly affecting temperature, power, application performance, and overall throughput. Consequently, the return values of the parameter mining phase depend on the actual workload. For that reason, we created a library of micro-benchmarks that consists of memory-, cache-, and compute-intensive micro-benchmarks that put tunable pressure on the shared resources. Specifically, the memory- and cache-intensive benchmarks have constant memory bandwidth with configurable size, while the compute-intensive ones pressure the integer and floating point units of the cores. Additionally, each micro-benchmark takes as input a value that dictates its working dataset. If the dataset value is less than the L1 cache, then a compute-intensive benchmark is triggered. If the dataset value is less than the double size of L2 cache, then a cache-intensive is used that heavily utilizes the data in L2 cache. In all other cases, a memory-intensive micro-benchmark is triggered putting pressure on the memory controller and thrashing the L2 cache. This value of the dataset is the control knob of the stochastic optimizer.

Initially, a random micro-benchmark is loaded on each core (random dataset value). In the case of query Q2, one core always executes the user-defined application, and the rest random micro-benchmarks. After the end of the execution, we retrieve the trajectories of temperature, power consumption, application performance, and system throughput. In order to correlate the output trajectories with the activity of the cluster and the behavior of the concurrent executing applications, we use the per core and total Memory Reads Per Instruction (MRPI) as metrics, which can be monitored at run-time (MRPI = last level cache misses/instructions retired) [87]. The MRPI is a frequency-less metric that has been proven to provide better performance insights for concurrent executing applications [87]. According to the PSTL expression used, the output

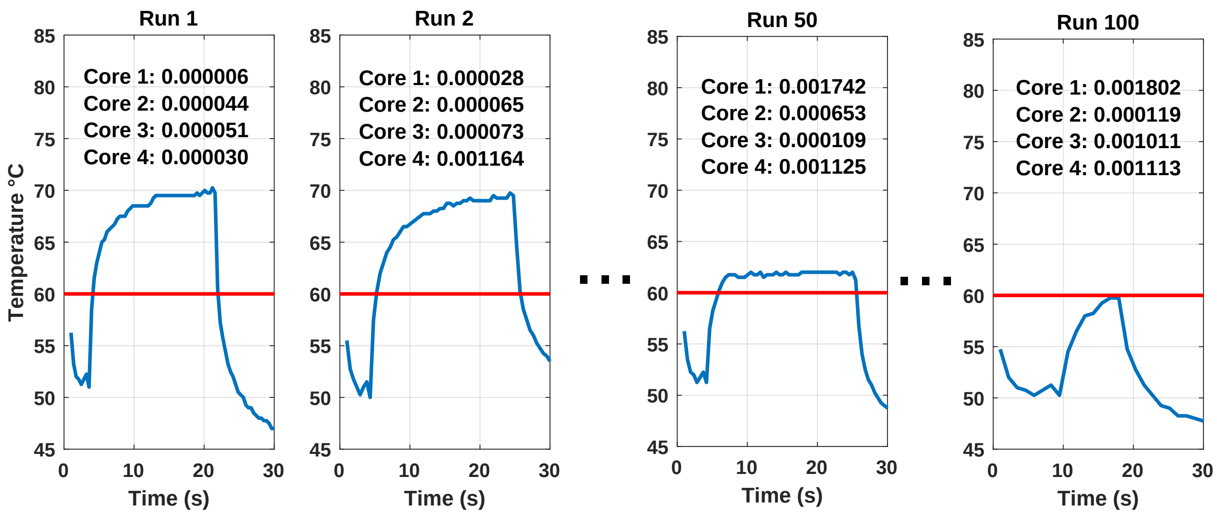

Figure 5.2: Example on PSTL parameter mining and stochastic optimization [2]

trajectories are analyzed and the robustness value of the system is calculated.

The stochastic optimizer uses this robustness value and correlates it with the per-core dataset values that produced that output. Then, it decides the next workload by modifying the dataset value per core, thus modifying the pressure on the shared resources. By tuning the dataset value or micro-benchmarks per core, the stochastic optimizer aims to push system behavior as close as possible to the specified constraint boundaries. The parameter exploration terminates once a maximum number of tests is completed and it is performed for all the available frequencies. At the end, the framework returns the mined values of θ along with the maximum MRPI value under which the values were mined. This value shows how compute- or memory-intensive the workloads can be, in order to satisfy the initially given constraints. Low values of MRPI indicate a more compute-intensive behavior, while larger values indicate a more memory-intensive one.

Figure 5.2 shows an example of the mining process for query Q1 considering $Temp_{safe} = 60^{\circ}C$. For this example, we set the maximum number of tests for the stochastic optimizer to 100. For each run, Figure 5.2 shows the output temperature signal, as well as the MRPI value of each core (4 cores total). Initially (Run 1), random micro-benchmarks are assigned on each core. At this point, the PSTL formula is not satisfied. The robustness value of the system is negative (since we have exceeded $Temp_{safe}$). Thus, the stochastic optimizer selects to modify the dataset value for the cores. In the second run, the temperature drops slightly, since the sampler picked a more memory-intensive workload for Core 4. As the procedure continues and the stochastic optimizer keeps modifying the dataset, at run 50 the temperature has dropped $15°C$, and the MRPI values of Core 1 and Core 4 correspond to more memory-intensive workloads. At this point, the robustness value of the system is still negative, however, it is approaching zero as the distance of the trajectory from the threshold becomes smaller. Finally, at run 100 which is the last one, the stochastic optimizer has selected such values, that the output temperature of the system does not exceed $60°C$. The robustness value of the system is now positive with an absolute value close to zero. The resulting mined parameter values in this example were θ_1, θ_2, which correspond to the Pareto-front of maximum power and system throughput that satisfied the temperature constraint.

After the exploration phase is done for each frequency value, we create a look-up table that contains tuples of the frequency of the mining phase, and the respective workloads associated with their MRPI values. We built our run-time manager on top of the goal manager presented in [9] as this method follows the same principle as our design, having three hierarchical goals (1) thermal safety, (2) user experience, (3) and performance. The thermal safety goal has the highest priority and it should always be satisfied in terms of a fixed power threshold. An urgency metric is introduced in order to capture the extent of the violation of a constraint [9]:

$$U_{Pow} = \frac{P_{curr}}{P_{ref}}, U_{Perf} = \frac{perf_{max} - perf_{curr}}{perf_{max} - perf_{ref}} \tag{5.3}$$

where P_{curr} is the current power consumption and $perf_{curr}$ is the current application performance, while values of urgency equal or greater than one indicate a constraint violation. However, for all the constraints the reference values are fixed and the same for all applications (e.g., $P_{ref} = 4W$ (TDP)). As a way to adjust the frequency at run-time taking into consideration the effect of each application and the properties of the platform, the run-time manager monitors periodically the per core and total MRPI values, finds in the look-up table the corresponding mined power threshold (P_{ref} in Equation 5.3), and adjusts the operating frequency accordingly so as not to exceed it. Thus, the proposed method allows the run-time manager to associate the power threshold with the current workload, ensuring better thermal safety and increasing power efficiency.

5.1.2 Results & Evaluation

The developed automation framework was evaluated on top of the A15-cluster of Odroid-XU3 board utilizing the S-TaLiRo [131, 145] tool for PSTL mining. All workloads run on the Odroid platform, and the output observations (e.g., temperature, power) are sent to a host computer that executes S-TaLiRo and performs the parameter mining (i.e., robustness analyzer and stochastic optimization). The Odroid-XU3 board is organized into big and LITTLE clusters with four Cortex A15 and four Cortex A7 cores respectively. In this work, we considered only the A15 cluster as the A7 is a low-power cluster with minimum impact on temperature. Finally, we utilized the underlying Performance Monitoring Counters (PMCs) to monitor and record the retired instructions and L2 cache misses (MRPI), while the in-built Power Monitoring Unit (PMU) was used to record power consumption and temperature.

Regarding query Q1 (Equation 5.1), we performed the parameter mining considering $Temp_{safe} = 60°C$. We selected to run the parameter mining process by setting 100 iterations for the stochastic optimizer. This process requires approximately one hour on the Odroid-XU3. In order to extract a more optimized Pareto-front of solutions, we repeated this process 10 times. At this point, it is important to mention this step is required only once and is performed offline. Once the power threshold values have been extracted, we enhance the run-time allocator. We have experimentally observed that we are able to serve any incoming workload, regardless of whether it has been previously encountered. Finally, the mining process can be applied similarly to any other platform, as long as we have access to performance counters, temperature, and power monitoring units.

Regarding the evaluation of query Q1 (Equation 5.1), we created five workload mixes. Each mix consists of twelve applications from the Polybench [146] and Stream [147] benchmark suites and it is executed in the following way. Initially, four applications are launched on the four cores of the board. Once an application finishes its execution, it is replaced by another one until all twelve applications terminate. In that way, we are able to create operating conditions with varying pressure on the shared resources. Figures 5.3(a)-(d) depict the behavior of our method in terms of power consumption (W), temperature ($^{\circ}C$), power efficiency (normalized $MIPS/Watt$), and normalized energy consumption against:

M1: The goal-driven method in [9]. This work utilizes a TDP of $4W$, justified by the need for thermal safety. This method is also the baseline of our experiments in Figures 5.3(c)-(d);

M2: The method in [33], which uses Multi-Layer Perceptrons (MLPs) to set the operating frequency under specific power budgets. This method uses MLPs as classifiers, where their inputs are the run-time values of specific PMCs. In our experiments, we target the minimum power threshold of $5W$ that the authors also considered;

M3: The MRPI-based method in [87], which employs the MRPI as a classification metric to reduce energy consumption.

Figures 5.3(a)-(b) depict the shape of power and temperature distribution (respectively) per mix for its whole duration, as well as the corresponding median values observed. As mentioned, M1 considers $TDP = 4W$ and according to the results that it achieves in all mixes, it has a small variation. However, the temperature observations for

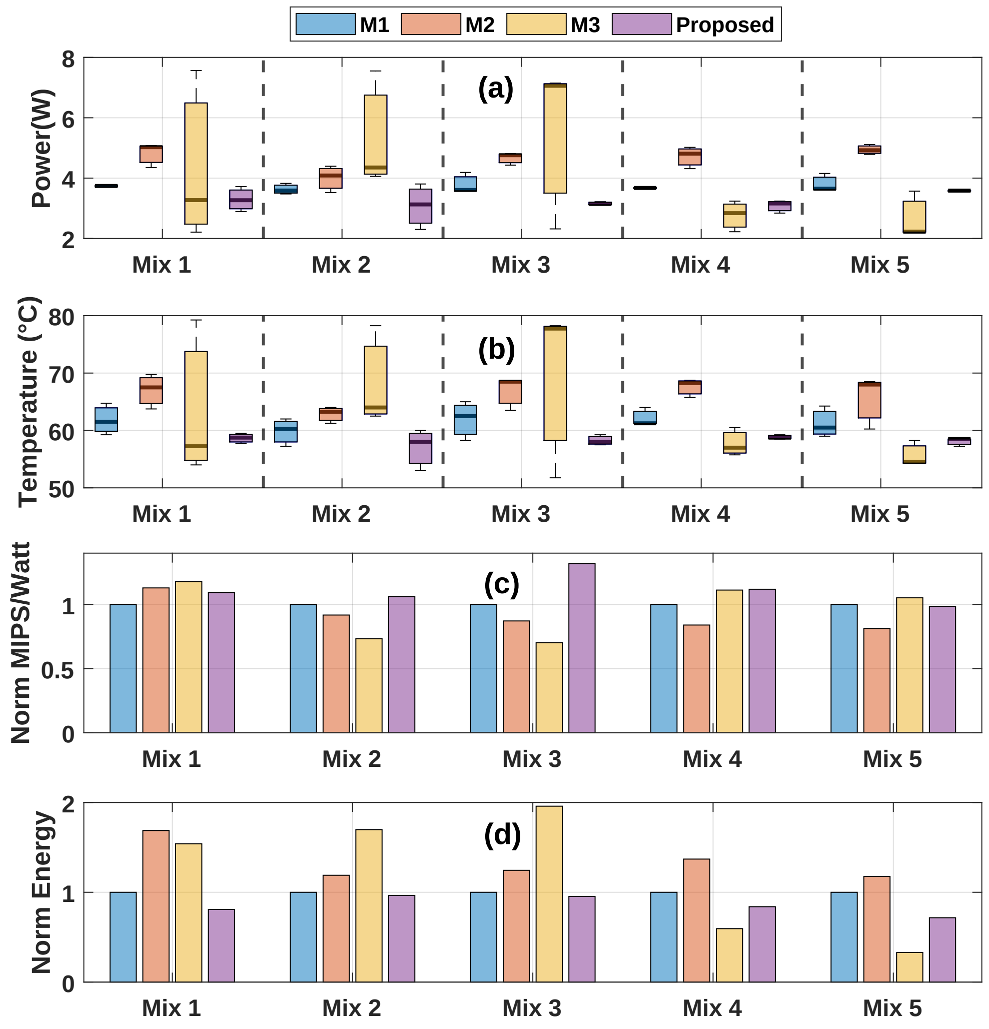

Figure 5.3: Evaluation of the proposed approach [2]

M1 (Figure 5.3(b)) along all the selected mixes, show a minimum value of $58°C$ and a maximum value of $66°C$. This difference of $8°C$ validates our motivation that a single TDP value can result in large temperature variations. Similarly to M1, M2 also shows a small variation in the resulting power, since it employs the usage of a neural network for different target frequencies, and is also characterized by power awareness. At this point, it is important to mention that the authors mention in their paper that the MLPs are trained with a target power consumption of $P_{safe} = 4.8W$ as a "safety net" [33]. This leads to more pessimistic predictions and therefore lower performance observations. Interestingly, M2 shows a minimum observed temperature of $60°C$ and a maximum temperature of $70^{\circ}C$, which again validates our motivation for mining and setting the power budget based on the behavior of the workload. On the contrary, M3 does not aim to remain below a certain TDP value. This method adjusts the operating frequency by monitoring the MRPI value of the currently executed mix in order to reduce energy consumption without affecting its performance. However, if the current workload contains compute-intensive applications (low MRPI), whose performance is greatly affected by frequency scaling, the MRPI method does not reduce the operating frequency drastically. In our scenario, mixes 1, 2, and 3 contain both compute- and memory-intensive groups of workloads. Hence M3 in these cases shows great variation in power and temperature since it will choose a lower frequency for compute-intensive groups and a higher frequency for memory-intensive groups. Therefore the observed temperature variation for just one mix, (e.g. Mix 1), can be up to $20^{\circ}C$. M3 results in low power and temperature observations for mixes 4, 5, since the MRPI value of these workloads, is assigned to very low frequencies. However, they still yield variation in power, without significant gains in MIPS/Watt. Our method managed to keep the temperature variation low by utilizing the mined parameters for different power budget thresholds. Additionally, our method achieved an average gain of 11% regarding power efficiency over the second-best M1 (Figure 5.3(c)). Similarly, the energy consumption of our method is on average 15.4% and 35.3% lower than M1 and M3, respectively. Finally, our method decreases throughput by 2.72% compared to M1. Overall, satisfying a global power threshold may result in undesired temperature variations. Our experiments revealed that our systematic method, for associating mined power threshold parameters with the current workload, ensures thermal safety and increased power efficiency.

Regarding query Q2 (Equation 5.2), we selected two applications with different behavior that cover a wide variety of activities in modern embedded systems. Specifically, we utilized the `stream` benchmark [147] and the `3mm` (3 Matrix Multiplications) application from the polybench benchmark suite [146]. For all the selected applications,

Table 5.1: Comparison against [9] for used defined applications

	3mm on [9]	**3mm Proposed**	**stream on [9]**	**stream Proposed**
Avg Temperature	$61°C$	$57°C$	$66°C$	$60°C$
Avg Power	3.06 W	2.76 W	3.57 W	3.61 W
Norm. IPS gain	1	0.97	1	1.02

we performed the parameter mining considering $Temp_{safe} = 60°C$.

We mined the parameters of P_{thr}, $Perf_{thr}$, and Th_{thr} for the frequency range of $1.4GHz$ - $2.0GHz$. We selected these frequencies, as values less than $1.4GHz$ do not have a significant impact on temperature. Finally, we selected to run the parameter mining process by setting 100 iterations for the stochastic optimizer and we repeated this process 10 times. This process is completed within an hour and needs to be done once per application. The evaluation scenarios are as follows. Initially, the user starts the system running the desired application on one core along with some random co-runners. The co-runners can terminate at any time and new different applications replace them. For all scenarios, the user-selected application runs always on one core and with an allowable maximum performance drop of 10% ($Perf_{min} = 0.9 \times Perf_{max}$). At the same time, the overall throughput of each core, in terms of IPS, should be at least 2.9×10^6. We set this value according to to [86] which utilized the same board. We compared the behavior of our system against the goal-driven method in [9] which also supports application performance thresholds. Table 5.1 depicts that, for the `3mm` compute-intensive application, our method dropped the performance by 3% compared to [9], since the achieved average power is lower. This is a direct result of our method's primary goal of thermal safety assurance. Indeed, our method manages to maintain an average temperature of $57°C$. For the `stream` memory-intensive application, we achieved lower temperature with approximately the same power consumption as [9]. In this case, our performance was higher by 2%. We consider the difference in performance negligible for both application scenarios since our primary goal is to maintain an average temperature less than or equal to $Temp_{safe} = 60°C$.

5.2 FREQUENCY-BASED POWER EFFICIENCY IMPROVEMENT OF CNNS ON HETEROGENEOUS IOT COMPUTING SYSTEMS

This section will present a framework that efficiently deploys an existing CNN on low-power GPU-based architectures that are suitable for computer vision applications. The proposed implementation is based on an efficient design space exploration to optimize the implementation of the CNNs and exploits the available hardware resources.

5.2.1 Methodology

In this framework, we considered a heterogeneous system that consists of both CPU and GPU, making it suitable for CNN execution. The operation of the device is described by the following **D**evice **O**perating **P**oints:

$$\text{DoP} = \{f_C, f_G, f_M, P_{total}, t_I\} \tag{5.4}$$

where f_C, f_G, f_M are the operating frequency of the CPU, GPU, and memory respectively. The total power consumption of the device $P_{total} = P_C + P_G + P_M$ is determined by the above operating frequencies and directly affects the performance of the CNN. We quantify the performance by measuring how fast is the CNN inference. Thus, we record the inference time t_I of the CNN as a function of f_C, f_G and f_M.

$$t_I = f(f_C, f_G, f_M) \tag{5.5}$$

For a given CNN architecture, the power footprint and the performance can vary depending on the selection of the f_C, f_G, f_M. For a device that schedules the CNN inference in a frequency-unaware manner, the performance is far from optimal. Although the device has a few pre-defined modes of operation (e.g., high performance, low power), the dynamicity of the IoT environments (varying resources over time) and the growing complexity of CNN architectures make it difficult to cover the increasing user needs. An existing CNN architecture is formulated as a set of different dimensions, targeting the ResNet neural network architecture [107]. Inspired by the work in [98] we consider that the dimensions of the CNN form the *shape* of the network. The residual block of the ResNet architecture is the building block for our formulation [107]. To that end, each CNN is formulated as $C(s, d, w)$, where s describes the stages, d the depth, and w the width of the CNN.

The questions that arise are *how is an efficient DoP defined* and *how do we choose this DoP*. In this work, we consider three different modes of device operation with three different objectives:

- high performance: in this mode, the system goal is to minimize inference time. Our focus is to find which DoP can give us inference time close to the minimum value with lower power consumption. Utilizing the highest operating frequencies does not always provide a significant performance gain.

- low power: in this mode, the system goal is to significantly reduce power consumption. Our focus is to select a DoP so that the total power consumption is minimized under no other constraints

- default mode: this mode offers an acceptable inference time with reasonable power consumption. Our focus is to select a DoP so that the total power observed consumption is maintained at an average acceptable level.

These modes are widely supported by modern mobile devices offering a more responsive system to the end user.

Figure 5.4 presents the overview of the proposed framework. The structure can be decomposed into two different stages. First, for each $C(s, d, w)$ architecture, we observe all the t, P_{total} values for all possible f_C, f_G, f_M combinations. Our framework employs a design space exploration mechanism to decide which candidate triples to be loaded on the

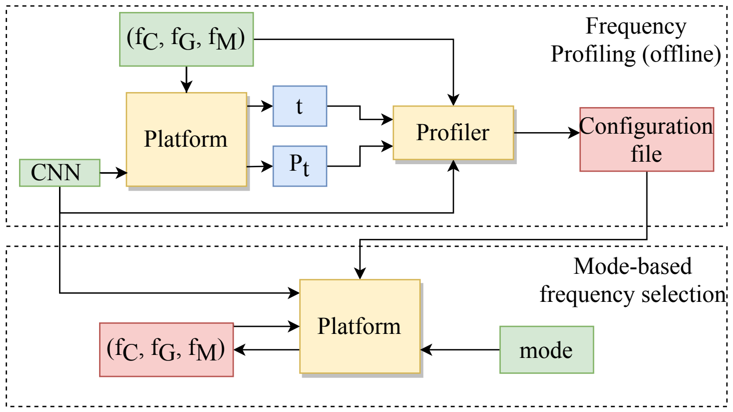

Figure 5.4: Execution flow of the proposed framework [8]

IoT device. Then, we create a configuration file for each architecture to hold this information. During this stage, there is also an elimination procedure involved where we discard some of these mixes. This stage happens only once per architecture during design-time. The user is then able to choose a mode of operation which can be either a high performance or a low power mode. If none is selected, the framework operates at a predefined default mode. Finally, the platform is set to operate in the selected combination which satisfies the mode-defined optimization objective.

We target heterogeneous computing platforms that consist of a single System-on-Chip (SoC) that incorporates a multi-core CPU and a many-core GPU that communicate through the main memory, a typical architecture in embedded systems [148], [40]. These platforms can be configured in multiple frequency levels for the CPU, GPU, and main memory and provide different performance levels to the user. We assume that each frequency can be adjusted to $N_{f_C}, N_{f_G}, N_{f_M}$ levels, respectively. We use this capability to explore all possible combinations of frequency values among the CPU, GPU, and memory in order to construct a configuration file which maps frequencies with inference time and power values. Then, the user can select a mode that either minimizes the total power consumption or maximizes performance with respect to power consumption.

Eventually, we want to create a configuration file with tuples of the form

$(f_C, f_G, f_M, t_I, P_{total})$, where t_I is the inference time in seconds and P_{total} is the total power consumption in W. For each CNN architecture, there is a corresponding configuration file consisting of these tuples. Initially, all combinations of the available frequencies are explored for each of the CNN architectures. In our case, we select the NVIDIA Jetson TX2 development platform for the development of the proposed framework. In total, the selected platform allows 12 frequency values for f_C, 14 for f_G, and 7 for f_M, adding up to a total of 1176 different combinations.

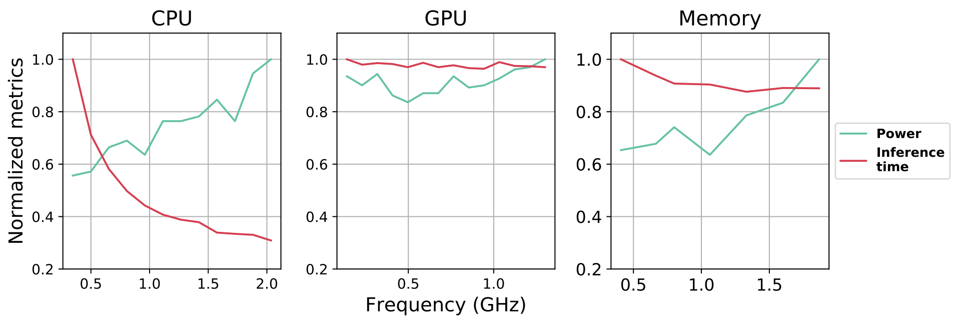

Figure 5.5: Behavior of power and inference time as frequencies change [8]

To reduce the design space, we monitor the behavior of each parameter separately. Figure 5.5 shows the behavior of the power and inference time as CPU, GPU, and memory frequencies change for a given CNN architecture. The baseline is the maximum power and inference time observed for the showcased interval. For each of the behavioral graphs, we are altering the frequency of the module under review within its available range, while we keep constant the frequencies of the other two. From these intermediate results, we can draw conclusions regarding the rejection of certain frequency values. Based on the behavior depicted in Figure 5.5, we can deduce that lower CPU frequencies result in comparatively high inference time. Therefore we exclude all the tuples that include $f_C \leq 0.8$ GHz. Regarding the GPU behavior, when the GPU frequency is increasing the resulted in inference time is reduced very slowly. Hence, for higher frequency values, there is no significant gain in performance, whereas there is a power increase. Thus, we discard the tuples that include the values $f_{G_{max}}$. We also reject $f_{G_{min}}$

tuples since they infer poor performance. Similar conclusions can be drawn from the observed memory behavior. For lower f_M values the inference time is low, hence we do not include $f_{M_{min}}$ in the final combinations. As the frequency of the memory increases, the inference time is decreasing, however, the power is increasing at a faster rate. Consequently, we eliminate tuples that include higher memory frequency values. Additionally, the frequency of the memory should be greater than the GPU because in this case, memory becomes a significant bottleneck in the computation. f Finally, we select 12 different frequency combinations (f_C, f_G, f_M): (2, 1.2, 1.3), (1.4, 1.2, 1.3), (0.81, 1.2, 1.3), (2, 1.03, 1.06), (1.4, 1.03, 1.06), (0.81, 1.03, 1.06), (2, 0.77, 0.8), (1.4, 0.77, 0.8), (0.81, 0.77, 0.8), (2, 0.59, 0.67), (1.4, 0.59, 0.67) and (0.81, 0.59, 0.67) (in GHz).

After this procedure, each CNN architecture is executed when the system operates under these frequency combinations. The goal is to receive information regarding the inference time t_I and power total power consumption P_{total} per CNN for each one of the 12 frequency triples and ultimately construct one configuration file for each CNN architecture, composed of 12 $(f_C, f_G, f_M, t_I, P_{total})$ tuples.

The presented framework can be used under three different modes as aforementioned: 1) high performance, 2) low power, and 3) default mode. For each of these modes, we choose one final device operating point from the configuration file. When the user chooses to operate the system under the low power mode, the frequency triple selected from the configuration file corresponds to the tuple where P_{total} is minimum. For the high performance operating mode, the resulting frequency combination corresponds to the tuple with the minimum P_{total} value objective to $t_I \geq \hat{t_I}$, where $\hat{t_I}$ is a user-defined inference time threshold. If there is not a specified mode from the user, then the framework switches the system's operation to a default state determined by the framework. In this case, the (f_C, f_G, f_M) are chosen at the tuple where the average value of P_{total} is observed. The 12-entry configuration file is queried at run-time for the user-specified mode and the framework changes the device's operating point accordingly.

5.2.2 Results & Evaluation

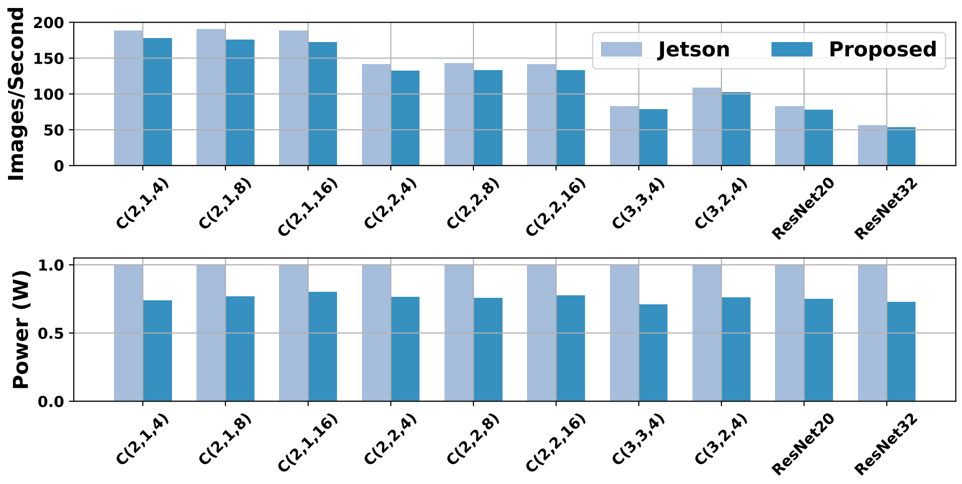

Figure 5.6: Comparison with Jetson's High Performance mode [8]

We conducted several experiments on the NVIDIA Jetson TX2 board, which is commonly used in IoT deployments [100], to assess the proposed framework. This embedded development board includes a CPU cluster comprising a dual-core Denver 2 processor and a quad-core ARM Cortex-A57, as well as a 256-core NVIDIA Pascal GPU. The Jetson TX2 module provides two power monitors for measuring power consumption[1] and permits the user to set individual ranges for the CPU, GPU, and memory frequency. The CPU, GPU, and memory frequencies are all available within specified ranges, which are 0.81–2GHz, 0.42–1.3GHz, and 0.67–1.87GHz, respectively. The board's sensors monitor the current frequency of each module, and the board also allows specific frequency values to be set. Our proposed framework is implemented in Python 3.5, using Keras[2], an open-source neural network library. We utilize various CNN architectures to evaluate the behavior of our framework against networks with different characteristics. We conducted training and inference experiments on the German Traffic Sign Benchmark (GTSRB) [111] dataset, which contains over 40 classes of 50,000 color images with varying dimensions ranging from 15x15 to 250x250 pixels. We selected this dataset to

[1]https://developer.nvidia.com/embedded/dlc/jetson-tx2-thermal-design-guide
[2]https://keras.io/

showcase the proposed framework's effectiveness in practical traffic sign detection applications using readily available hardware.

We conducted multiple experiments to evaluate our framework's performance on the Jetson TX2 platform's available modes: high performance, low power, and default. We used two metrics to measure the framework's effectiveness: normalized power in watts and the number of images per second. Ten different CNN architectures were deployed on the platform for the three different modes. Initially, we conducted the experiments using Jetson's built-in operating modes, and then with our proposed framework's modes. The following CNN architectures were utilized for the experiments: C(2,1,4), C(2,1,8), C(2,1,16), C(2,2,4), C(2,2,8), C(2,2,16), C(3,3,4), C(3,2,4), ResNet20, and ResNet32.

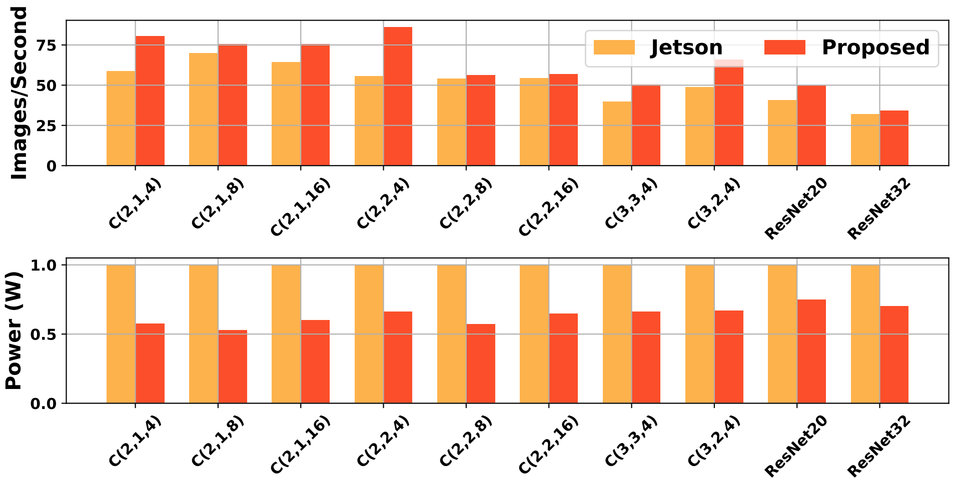

Figure 5.7: Comparison with Jetson's Low Power mode [8]

The results for the high performance mode are presented in Figure 5.6. Our framework achieved a slightly lower number of images per second than the Jetson, but the corresponding power consumption was significantly lower. Specifically, while the Jetson's high performance mode provided up to 9% higher images per second, the proposed framework achieved up to 31.5% lower power consumption. The lower performance of our approach can be attributed to the Jetson's use of the $(f_{C_{max}}, f_{G_{max}}, f_{M_{max}})$ frequency combination, which results in higher power consumption. However, we discarded the maximum frequency pairs during the profiling procedure to avoid high power consumption observations.

The results for the low power mode are shown in Figure 5.7. Our proposed framework achieved both higher performance and lower power consumption compared to Jetson in this mode. The observed performance rate was up to 43% higher, while the power consumption was up to 61.5% lower.

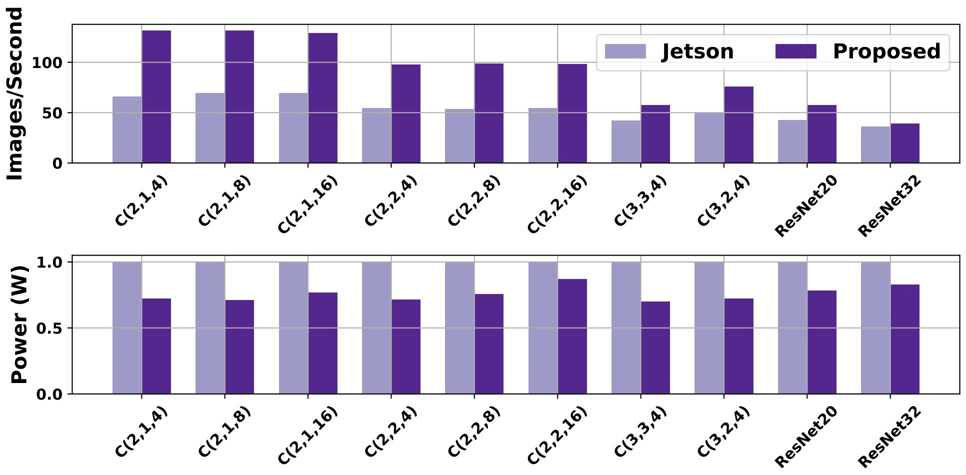

Figure 5.8: Comparison with Jetson's Default mode [8]

Finally, Figure 5.8 illustrates that our framework's default mode outperforms the corresponding mode on the Jetson platform in terms of both power and performance. The documented performance rate exceeded Jetson by up to 66.3%, and the observed power consumption was up to 35.35% lower.

CHAPTER 6
CONCLUSION

The main focus of this dissertation was on achieving energy efficiency in the deployment of machine learning applications on heterogeneous embedded devices. The work can be classified into three main categories, as follows:

- Chapter 3 addressed energy-efficient machine learning inference through approximate computing techniques. The focus was on designing approximate multipliers and hardware accelerators.
- Chapter 4 proposed frameworks that utilized approximate computing techniques to systematically explore the search space for energy-efficient Deep Neural Network (DNN) inference while ensuring fine-grain accuracy properties.
- Chapter 5 presented frameworks that selected operating frequencies on embedded devices to achieve lower power consumption.

Specifically, regarding Chapter 3, in Section 3.1 we introduced an approximate multiplier that offers the flexibility of generating positive, negative, or no error. Our mathematical analysis illustrated that by carefully configuring the modes of the approximate multiplier per weight, we can minimize convolution error and achieve high inference accuracy. To achieve this, we proposed a filter-oriented mapping methodology that maximizes the applied approximation while satisfying a specified accuracy drop threshold, thereby targeting high energy efficiency. Our comprehensive experimental evaluation indicated that our filter-oriented approximation using the positive/negative multiplier significantly outperforms the state-of-the-art without necessitating retraining. In the second part of Chapter 3, Section 3.2 presented an asymmetric Heterogeneous Deep Accelerator (HDA) design that includes 8-bit Neural Processing Units (NPUs) and lower bit-width NPUs, based on the *big.LITTLE* architecture paradigm. The goal was to efficiently deploy workloads comprising multiple DNNs. We designed multiple HDAs and utilized our run-time-enabled post-training quantization method and NN-to-NPU scheduling to evaluate the impact of various HDA topologies on the NN response time and accelerator's energy consumption. Our results demonstrated that, when compared to a Homogeneous design, for a maximum accuracy drop of 1.5%, the NN response time can be reduced by up to 18.2%, and the NN energy consumption can be reduced by up to 35.17%. Similarly, for a maximum accuracy drop of 1%, the NN response time can be decreased by up to 13.4%, and the energy consumption can be reduced by 31.8%.

In Chapter 4 we utilized formal properties of approximate DNN accelerators to develop two different frameworks that utilize approximation techniques. In Section 4.2 we presented a unified framework that supports reconfigurable approximate multiplications, enabling fine-grain optimizations. We utilized Parametric Signal Temporal Logic (PSTL) to express DNN properties and employed stochastic optimization to find approximate mappings that satisfy accuracy thresholds. We conducted an in-depth evaluation across multiple DNNs, datasets, and accuracy requirements, to obtain a better grasp of how strict constraints can still yield energy savings. When compared to other mapping methodologies, on the same multipliers, our framework achieves even more than $\times 2$ the energy gains without violating the defined accuracy thresholds. In the second part of Chapter 4 and specifically in Section 4.3, we proposed an automated framework that systematically explores pruning and quantization compression configurations based on fine-grain accuracy requirements through Signal Temporal Logic (STL). At the same time, the proposed framework avoids fine-tuning and retraining. The end user is offered three final modes of operation to control the quality of run-time inference at will. Extensive evaluation on the ImageNet dataset shows very big gains in energy consumption, even more than 30% in some cases when compared to the baseline.

In Chapter 5 we presented two frameworks that select on operating frequencies on two different types of heterogeneous embedded devices. In Section 5.1 we presented a framework for formal property exploration on clustered Clustered Multi-Processors (CMPs). Specifically, we utilized PSTL to express run-time constraints and mine multiple parameters in order to create optimized system- and application-aware operating points. Instead of having fixed values for power constraints, we enhanced the run-time manager to consider different threshold values based on the mined parameters. At run-time, the allocator utilizes these operating points to select the frequency in order to provide thermal safety and satisfy user requirements making reasonable trade-offs wherever necessary. Even though parameters vary for different systems, our method can be similarly applied to other CMPs. In Section 5.2, we presented a systematic approach for implementing a complex Convolution Neural Network (CNN) on a heterogeneous computing system for the Internet of Things (IoT). We developed a framework that explores the operating frequencies of the device, prunes the design space, and determines the optimal device configuration based on system objectives, such as minimizing power or maximizing performance. Our proposed framework demonstrated significant improvements over the default governor and operation mode of the system. On average, it achieved a 33.4% performance improvement, with up to 66.3% in certain cases, while reducing power consumption by an average of 42.8%, with up to 61.5% in certain cases.

REFERENCES

[1] Z.-G. Tasoulas, G. Zervakis, I. Anagnostopoulos, H. Amrouch, and J. Henkel, “Weight-oriented approximation for energy-efficient neural network inference accelerators,” *IEEE Transactions on Circuits and Systems I: Regular Papers*, vol. 67, no. 12, pp. 4670–4683, 2020.

[2] O. Spantidi, I. Anagnostopoulos, and G. Fainekos, “Efficient resource management of clustered multi-processor systems through formal property exploration,” in *2021 Design, Automation & Test in Europe Conference & Exhibition (DATE).* IEEE, 2021, pp. 1673–1678.

[3] O. Spantidi, G. Zervakis, I. Anagnostopoulos, H. Amrouch, and J. Henkel, “Positive/negative approximate multipliers for dnn accelerators,” in *2021 IEEE/ACM International Conference On Computer Aided Design (ICCAD).* IEEE, 2021, pp. 1–9.

[4] O. Spantidi, G. Zervakis, S. Alsalamin, I. Roman-Ballesteros, J. Henkel, H. Amrouch, and I. Anagnostopoulos, “Targeting dnn inference via efficient utilization of heterogeneous precision dnn accelerators,” *IEEE Transactions on Emerging Topics in Computing*, 2022.

[5] V. Mrazek, Z. Vasícek, L. Sekanina, M. A. Hanif, and M. Shafique, “Alwann: automatic layer-wise approximation of deep neural network accelerators without retraining,” in *2019 IEEE/ACM International Conference on Computer-Aided Design (ICCAD).* IEEE, 2019, pp. 1–8.

[6] O. Spantidi, G. Zervakis, I. Anagnostopoulos, and J. Henkel, “Energy-efficient dnn inference on approximate accelerators through formal property exploration,” *IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems*, vol. 41, no. 11, pp. 3838–3849, 2022.

[7] T.-J. Yang *et al.*, “Designing energy-efficient convolutional neural networks using energy-aware pruning,” in *Proceedings of the IEEE conference on computer vision*

and pattern recognition, 2017, pp. 5687–5695.

[8] O. Spantidi, I. Galanis, and I. Anagnostopoulos, "Frequency-based power efficiency improvement of cnns on heterogeneous iot computing systems," in *2020 IEEE 6th World Forum on Internet of Things (WF-IoT).* IEEE, 2020, pp. 1–6.

[9] E. Shamsa *et al.*, "Goal-driven autonomy for efficient on-chip resource management: Transforming objectives to goals," in *2019 Design, Automation & Test in Europe Conference & Exhibition (DATE).* IEEE, 2019, pp. 1397–1402.

[10] H. Kwon, L. Lai, T. Krishna, and V. Chandra, "Herald: Optimizing heterogeneous dnn accelerators for edge devices," *arXiv e-prints*, pp. arXiv–1909, 2019.

[11] N. P. Jouppi, C. Young, N. Patil, D. Patterson, G. Agrawal, R. Bajwa, S. Bates, S. Bhatia, N. Boden, A. Borchers *et al.*, "In-datacenter performance analysis of a tensor processing unit," in *Proceedings of the 44th annual international symposium on computer architecture*, 2017, pp. 1–12.

[12] J. L. Hennessy and D. A. Patterson, "A new golden age for computer architecture," *Communications of the ACM*, vol. 62, no. 2, pp. 48–60, 2019.

[13] C.-J. Wu *et al.*, "Machine learning at Facebook: Understanding inference at the edge," in *2019 IEEE International Symposium on High Performance Computer Architecture (HPCA).* IEEE, 2019, pp. 331–344.

[14] H. Amrouch, G. Zervakis, S. Salamin, H. Kattan, I. Anagnostopoulos, and J. Henkel, "Npu thermal management," *IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems*, vol. 39, no. 11, pp. 3842–3855, 2020.

[15] G. Zervakis, I. Anagnostopoulos, S. Salamin, O. Spantidi, I. Roman-Ballesteros, J. Henkel, and H. Amrouch, "Thermal-aware design for approximate dnn accelerators," *IEEE Transactions on Computers*, vol. 71, no. 10, pp. 2687–2697, 2022.

[16] O. Spantidi and I. Anagnostopoulos, "How much is too much error? analyzing the impact of approximate multipliers on dnns," in *2022 23rd International Symposium*

on Quality Electronic Design (ISQED). IEEE, 2022, pp. 1–6.

[17] V. Mrazek, Z. Vasicek, and L. Sekanina, "Evoapproxlib: Extended library of approximate arithmetic circuits," in *Proc. Workshop Open-Source EDA Technol.(WOSET)*, 2019, p. 10.

[18] G. Zervakis, O. Spantidi, I. Anagnostopoulos, H. Amrouch, and J. Henkel, "Control variate approximation for dnn accelerators," *arXiv preprint arXiv:2102.09642*, 2021.

[19] Z. Vasicek, "Formal methods for exact analysis of approximate circuits," *IEEE Access*, vol. 7, pp. 177 309–177 331, 2019.

[20] V. Mrazek, R. Hrbacek, Z. Vasicek, and L. Sekanina, "Evoapprox8b: Library of approximate adders and multipliers for circuit design and benchmarking of approximation methods," in *Design, Automation & Test in Europe Conference & Exhibition (DATE), 2017.* IEEE, 2017, pp. 258–261.

[21] S. S. Sarwar, S. Venkataramani, A. Ankit, A. Raghunathan, and K. Roy, "Energy-efficient neural computing with approximate multipliers," *ACM Journal on Emerging Technologies in Computing Systems (JETC)*, vol. 14, no. 2, pp. 1–23, 2018.

[22] J. Lee, C. Kim, S. Kang, D. Shin, S. Kim, and H.-J. Yoo, "Unpu: A 50.6 tops/w unified deep neural network accelerator with 1b-to-16b fully-variable weight bit-precision," in *IEEE International Solid-State Circuits Conference-(ISSCC).* IEEE, 2018.

[23] S. Venkataramani *et al.*, "Efficient ai system design with cross-layer approximate computing," *Proceedings of the IEEE*, vol. 108, no. 12, pp. 2232–2250, 2020.

[24] J. Han and M. Orshansky, "Approximate computing: An emerging paradigm for energy-efficient design," in *2013 18th IEEE European Test Symposium (ETS).* IEEE, 2013, pp. 1–6.

[25] G. Zervakis, H. Amrouch, and J. Henkel, "Design automation of approximate circuits with runtime reconfigurable accuracy," *IEEE Access*, 2020.

[26] A. Dokhanchi, H. B. Amor, J. V. Deshmukh, and G. Fainekos, "Evaluating perception systems for autonomous vehicles using quality temporal logic," in *Int. Conf. Runtime Verification*. Springer, 2018, pp. 409–416.

[27] K. Wang *et al.*, "Haq: Hardware-aware automated quantization with mixed precision," in *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition*, 2019, pp. 8612–8620.

[28] B. Jacob *et al.*, "Quantization and training of neural networks for efficient integer-arithmetic-only inference," in *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*, 2018.

[29] J. Choi *et al.*, "Pact: Parameterized clipping activation for quantized neural networks," *arXiv preprint arXiv:1805.06085*, 2018.

[30] H. Yang *et al.*, "Automatic neural network compression by sparsity-quantization joint learning: A constrained optimization-based approach," in *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition*, 2020, pp. 2178–2188.

[31] G. Fainekos *et al.*, "Robustness of specifications and its applications to falsification, parameter mining, and runtime monitoring with s-taliro," in *International Conference on Runtime Verification*. Springer, 2019, pp. 27–47.

[32] M. Guevara *et al.*, "Strategies for anticipating risk in heterogeneous system design," in *2014 IEEE 20th International Symposium on High Performance Computer Architecture (HPCA)*. IEEE, 2014, pp. 154–164.

[33] T. Marinakis *et al.*, "Meeting power constraints while mitigating contention on clustered multi-processor system," *IEEE Embedded Systems Letters*, 2019.

[34] S. Pagani *et al.*, "Thermal safe power (tsp): Efficient power budgeting for heterogeneous manycore systems in dark silicon," *IEEE Transactions on Computers*, vol. 66, no. 1, pp. 147–162, 2017.

[35] H. Amrouch *et al.*, "Optimizing temperature guardbands," in *Design, Automation &*

Test in Europe Conference & Exhibition (DATE), 2017. IEEE, 2017, pp. 175–180.

[36] I. Galanis, I. Anagnostopoulos, P. Gurunathan, and D. Burkard, “Environmental-based speed recommendation for future smart cars,” *Future Internet*, vol. 11, no. 3, p. 78, 2019.

[37] I. Galanis, S. S. N. Perala, and I. Anagnostopoulos, “Edge computing and efficient resource management for integration of video devices in smart grid deployments,” in *IoT for Smart Grids.* Springer, 2019, pp. 115–132.

[38] A. Mohan, K. Gauen, Y.-H. Lu, W. W. Li, and X. Chen, “Internet of video things in 2030: A world with many cameras,” in *Circuits and Systems (ISCAS), 2017 IEEE International Symposium on.* IEEE, 2017.

[39] S. S. N. Perala, I. Galanis, and I. Anagnostopoulos, “Fog computing and efficient resource management in the era of internet-of-video things (iovt),” in *2018 IEEE International Symposium on Circuits and Systems (ISCAS).* IEEE, 2018, pp. 1–5.

[40] https://www.samsung.com/semiconductor/minisite/exynos/products/mobileprocessor/exynos-9-series-9820/, [Online;].

[41] S. Venkataramani, X. Sun, N. Wang, C.-Y. Chen, J. Choi, M. Kang, A. Agarwal, J. Oh, S. Jain, T. Babinsky, N. Cao, T. Fox, B. Fleischer, G. Gristede, M. Guillorn, H. Haynie, H. Inoue, K. Ishizaki, M. Klaiber, S.-H. Lo, G. Maier, S. Mueller, M. Scheuermann, E. Ogawa, M. Schaal, M. Serrano, J. Silberman, C. Vezyrtzis, W. Wang, F. Yee, J. Zhang, M. Ziegler, C. Zhou, M. Ohara, P.-F. Lu, B. Curran, S. Shukla, V. Srinivasan, L. Chang, and K. Gopalakrishnan, “Efficient ai system design with cross-layer approximate computing,” *Proceedings of the IEEE*, vol. 108, no. 12, pp. 2232–2250, 2020.

[42] V. Mrazek, S. S. Sarwar, L. Sekanina, Z. Vasicek, and K. Roy, “Design of power-efficient approximate multipliers for approximate artificial neural networks,” in *Proceedings of the 35th International Conference on Computer-Aided Design*, 2016, pp. 1–7.

[43] M. S. Ansari, V. Mrazek, B. F. Cockburn, L. Sekanina, Z. Vasicek, and J. Han, "Improving the accuracy and hardware efficiency of neural networks using approximate multipliers," *IEEE Transactions on Very Large Scale Integration (VLSI) Systems*, vol. 28, no. 2, pp. 317–328, 2019.

[44] V. Mrazek, R. Hrbacek, Z. Vasicek, and L. Sekanina, "Evoapproxsb: Library of approximate adders and multipliers for circuit design and benchmarking of approximation methods," in *Design, Automation & Test in Europe Conference & Exhibition (DATE), 2017.* IEEE, 2017, pp. 258–261.

[45] V. Mrazek, L. Sekanina, and Z. Vasicek, "Using libraries of approximate circuits in design of hardware accelerators of deep neural networks," in *2020 2nd IEEE International Conference on Artificial Intelligence Circuits and Systems (AICAS).* IEEE, 2020, pp. 243–247.

[46] M. A. Hanif, F. Khalid, and M. Shafique, "Cann: Curable approximations for high-performance deep neural network accelerators," in *2019 56th ACM/IEEE Design Automation Conference (DAC).* IEEE, 2019, pp. 1–6.

[47] M. Riaz, R. Hafiz, S. A. Khaliq, M. Faisal, H. T. Iqbal, M. Ali, and M. Shafique, "Caxcnn: Towards the use of canonic sign digit based approximation for hardware-friendly convolutional neural networks," *IEEE Access*, vol. 8, pp. 127 014–127 021, 2020.

[48] I. Hammad, L. Li, K. El-Sankary, and W. M. Snelgrove, "Cnn inference using a preprocessing precision controller and approximate multipliers with various precisions," *IEEE Access*, vol. 9, pp. 7220–7232, 2021.

[49] G. Park, J. Kung, and Y. Lee, "Design and analysis of approximate compressors for balanced error accumulation in mac operator," *IEEE Trans. Circuits Syst. I, Reg. Papers*, vol. 68, no. 7, pp. 2950–2961, 2021.

[50] R. Ding, Z. Liu, T.-W. Chin, D. Marculescu, and R. D. Blanton, "Flightnns: Lightweight quantized deep neural networks for fast and accurate inference," in

Proceedings of the 56th Annual Design Automation Conference 2019, 2019, pp. 1–6.

[51] Q. Lu, W. Jiang, X. Xu, Y. Shi, and J. Hu, "On neural architecture search for resource-constrained hardware platforms," *arXiv preprint arXiv:1911.00105*, 2019.

[52] W. Jiang *et al.*, "Device-circuit-architecture co-exploration for computing-in-memory neural accelerators," *IEEE Trans. Comput.*, 2020.

[53] H. Kwon, P. Chatarasi, M. Pellauer, A. Parashar, V. Sarkar, and T. Krishna, "Understanding reuse, performance, and hardware cost of dnn dataflow: A data-centric approach," in *International Symposium on Microarchitecture*, 2019, pp. 754–768.

[54] H. Kwon, L. Lai, M. Pellauer, T. Krishna, Y.-H. Chen, and V. Chandra, "Heterogeneous dataflow accelerators for multi-dnn workloads," in *2021 IEEE International Symposium on High-Performance Computer Architecture (HPCA)*. IEEE, 2021, pp. 71–83.

[55] L. Yang *et al.*, "Co-exploration of neural architectures and heterogeneous asic accelerator designs targeting multiple tasks," *arXiv preprint arXiv:2002.04116*, 2020.

[56] A. Samajdar, Y. Zhu, P. Whatmough, M. Mattina, and T. Krishna, "Scale-sim: Systolic cnn accelerator simulator," *arXiv preprint arXiv:1811.02883*, 2018.

[57] A. Lottarini *et al.*, "Master of none acceleration: a comparison of accelerator architectures for analytical query processing," in *Proceedings of the 46th International Symposium on Computer Architecture*, 2019.

[58] L. Song, F. Chen, Y. Zhuo, X. Qian, H. Li, and Y. Chen, "Accpar: Tensor partitioning for heterogeneous deep learning accelerators," in *2020 IEEE International Symposium on High Performance Computer Architecture (HPCA)*. IEEE, 2020.

[59] M. P. Véstias, R. P. Duarte, J. T. De Sousa, and H. C. Neto, "A configurable architecture for running hybrid convolutional neural networks in low-density fpgas,"

IEEE Access, vol. 8, pp. 107 229–107 243, 2020.

[60] K.-C. Chen, M. Ebrahimi, T.-Y. Wang, and Y.-C. Yang, "Noc-based dnn accelerator: A future design paradigm," in *Proceedings of the 13th IEEE/ACM International Symposium on Networks-on-Chip*, 2019, pp. 1–8.

[61] M. Gao, X. Yang, J. Pu, M. Horowitz, and C. Kozyrakis, "Tangram: Optimized coarse-grained dataflow for scalable nn accelerators," in *Proceedings of the Twenty-Fourth International Conference on Architectural Support for Programming Languages and Operating Systems*, 2019, pp. 807–820.

[62] Y. S. Shao *et al.*, "Simba: Scaling deep-learning inference with multi-chip-module-based architecture," in *Proceedings of the 52nd Annual IEEE/ACM International Symposium on Microarchitecture*, 2019, pp. 14–27.

[63] E. Qin *et al.*, "Sigma: A sparse and irregular gemm accelerator with flexible interconnects for dnn training," in *2020 IEEE International Symposium on High Performance Computer Architecture (HPCA)*. IEEE, 2020, pp. 58–70.

[64] S. Han, X. Liu, H. Mao, J. Pu, A. Pedram, M. A. Horowitz, and W. J. Dally, "Eie: Efficient inference engine on compressed deep neural network," *ACM SIGARCH Computer Architecture News*, vol. 44, no. 3, pp. 243–254, 2016.

[65] L. Jiang, M. Kim, W. Wen, and D. Wang, "Xnor-pop: A processing-in-memory architecture for binary convolutional neural networks in wide-io2 drams," in *2017 IEEE/ACM International Symposium on Low Power Electronics and Design (ISLPED)*. IEEE, 2017, pp. 1–6.

[66] Z. Dong, Z. Yao, A. Gholami, M. W. Mahoney, and K. Keutzer, "Hawq: Hessian aware quantization of neural networks with mixed-precision," in *Proceedings of the IEEE/CVF International Conference on Computer Vision*, 2019, pp. 293–302.

[67] T. Wang, K. Wang, H. Cai, J. Lin, Z. Liu, H. Wang, Y. Lin, and S. Han, "Apq: Joint search for network architecture, pruning and quantization policy," in *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern*

Recognition, 2020, pp. 2078–2087.

[68] M. F. F. Khan, M. M. Kamani, M. Mahdavi, and V. Narayanan, "Learning to quantize deep neural networks: A competitive-collaborative approach," in *2020 57th ACM/IEEE Design Automation Conference (DAC)*. IEEE, 2020, pp. 1–6.

[69] Z. Dong, Y. Gao, Q. Huang, J. Wawrzynek, H. K. So, and K. Keutzer, "Hao: Hardware-aware neural architecture optimization for efficient inference," in *2021 IEEE 29th Annual International Symposium on Field-Programmable Custom Computing Machines (FCCM)*. IEEE, 2021, pp. 50–59.

[70] C. Gong, Z. Jiang, D. Wang, Y. Lin, Q. Liu, and D. Z. Pan, "Mixed precision neural architecture search for energy efficient deep learning," in *2019 IEEE/ACM International Conference on Computer-Aided Design (ICCAD)*. IEEE, 2019, pp. 1–7.

[71] E. Baek, D. Kwon, and J. Kim, "A multi-neural network acceleration architecture," in *2020 ACM/IEEE 47th Annual International Symposium on Computer Architecture (ISCA)*. IEEE, 2020, pp. 940–953.

[72] G. Armeniakos, G. Zervakis, D. Soudris, and J. Henkel, "Hardware approximate techniques for deep neural network accelerators: A survey," *ACM Comput. Surv.*, mar 2022. [Online]. Available: https://doi.org/10.1145/3527156

[73] B. Hoxha, A. Dokhanchi, and G. Fainekos, "Mining parametric temporal logic properties in model based design for cyber-physical systems," *Int. J. Softw. Tools Technol. Transf.*, vol. 20, pp. 79–93, 2018.

[74] E. Bartocci, J. Deshmukh, A. Donzé, G. Fainekos, O. Maler, D. Ničković, and S. Sankaranarayanan, "Specification-based monitoring of cyber-physical systems: a survey on theory, tools and applications," in *Lectures on Runtime Verification*. Springer, 2018, pp. 135–175.

[75] E. Asarin, A. Donzé, O. Maler, and D. Nickovic, "Parametric identification of temporal properties," in *Runtime Verification*, S. Khurshid and K. Sen, Eds.

Springer Berlin Heidelberg, 2012.

[76] S. Han *et al.*, "Learning both weights and connections for efficient neural networks," *arXiv preprint arXiv:1506.02626*, 2015.

[77] H. Li *et al.*, "Pruning filters for efficient convnets," *arXiv preprint arXiv:1608.08710*, 2016.

[78] W. Wen *et al.*, "Learning structured sparsity in deep neural networks," *Advances in neural information processing systems*, vol. 29, 2016.

[79] Y. He *et al.*, "Amc: Automl for model compression and acceleration on mobile devices," in *Proceedings of the European conference on computer vision (ECCV)*, 2018, pp. 784–800.

[80] F. Tung and G. Mori, "Clip-q: Deep network compression learning by in-parallel pruning-quantization," in *Proceedings of the IEEE conference on computer vision and pattern recognition*, 2018, pp. 7873–7882.

[81] S. Han *et al.*, "Deep compression: Compressing deep neural networks with pruning, trained quantization and huffman coding," *arXiv preprint arXiv:1510.00149*, 2015.

[82] Y. He *et al.*, "Filter pruning via geometric median for deep convolutional neural networks acceleration," in *Proceedings of the IEEE/CVF conference on computer vision and pattern recognition*, 2019, pp. 4340–4349.

[83] P. Hu *et al.*, "Opq: Compressing deep neural networks with one-shot pruning-quantization," in *Proceedings of the AAAI Conference on Artificial Intelligence*, vol. 35, no. 9, 2021, pp. 7780–7788.

[84] S. Stuijk *et al.*, "A predictable multiprocessor design flow for streaming applications with dynamic behaviour," in *Digital System Design: Architectures, Methods and Tools (DSD), 2010 13th Euromicro Conference on.* IEEE, 2010, pp. 548–555.

[85] A. Weichslgartner *et al.*, "Daarm: Design-time application analysis and run-time mapping for predictable execution in many-core systems," in *Hardware/Software Codesign and System Synthesis (CODES+ ISSS), 2014 International Conference*

on. IEEE, 2014, pp. 1–10.

[86] B. K. Reddy *et al.*, "Online concurrent workload classification for multi-core energy management," in *2018 Design, Automation & Test in Europe Conference & Exhibition (DATE)*.

[87] ——, "Inter-cluster thread-to-core mapping and dvfs on heterogeneous multi-cores," *IEEE Transactions on Multi-Scale Computing Systems*, vol. 4, no. 3, pp. 369–382, 2017.

[88] J. S. Koduri *et al.*, "Spa: Simple pool architecture for application resource allocation in many-core systems," in *2018 Design, Automation & Test in Europe Conference & Exhibition (DATE)*.

[89] R. Cochran *et al.*, "Pack & cap: adaptive dvfs and thread packing under power caps," in *2011 44th Annual IEEE/ACM International Symposium on Microarchitecture (MICRO)*. IEEE, 2011, pp. 175–185.

[90] R. Piscitelli *et al.*, "Design space pruning through hybrid analysis in system-level design space exploration," in *Design, Automation & Test in Europe Conference & Exhibition (DATE), 2012*.

[91] D. Olsen *et al.*, "Performance-aware resource management of multi-threaded applications on many-core systems," in *Proceedings of the on Great Lakes Symposium on VLSI 2017*, 2017, pp. 119–124.

[92] B. Donyanavard *et al.*, "Sparta: Runtime task allocation for energy efficient heterogeneous manycores," in *2016 International Conference on Hardware/Software Codesign and System Synthesis (CODES+ ISSS)*. IEEE, 2016, pp. 1–10.

[93] A. K. Singh *et al.*, "Resource and throughput aware execution trace analysis for efficient run-time mapping on mpsocs," *IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems*.

[94] E. Shamsa *et al.*, "Goal formulation: Abstracting dynamic objectives for efficient on-chip resource allocation," in *2018 IEEE Nordic Circuits and Systems Conference*

(NORCAS): NORCHIP and International Symposium of System-on-Chip (SoC). IEEE, 2018, pp. 1–4.

[95] "Odroid-xu3," https://www.hardkernel.com/shop/odroid-xu3/.

[96] F. Tsimpourlas, L. Papadopoulos, A. Bartsokas, and D. Soudris, "A design space exploration framework for convolutional neural networks implemented on edge devices," *IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems*, vol. 37, no. 11, pp. 2212–2221, 2018.

[97] J. Hyun, J. Park, K. Y. Kim, S. Yu, and W. Baek, "Ceml: a coordinated runtime system for efficient machine learning on heterogeneous computing systems," in *European Conference on Parallel Processing.* Springer, 2018, pp. 781–795.

[98] R. Mammadli, F. Wolf, and A. Jannesari, "The art of getting deep neural networks in shape," *ACM Transactions on Architecture and Code Optimization (TACO)*, vol. 15, no. 4, p. 62, 2019.

[99] D. Stamoulis, E. Cai, D.-C. Juan, and D. Marculescu, "Hyperpower: Power-and memory-constrained hyper-parameter optimization for neural networks," in *2018 Design, Automation & Test in Europe Conference & Exhibition (DATE).* IEEE, 2018, pp. 19–24.

[100] A. E. Eshratifar and M. Pedram, "Energy and performance efficient computation offloading for deep neural networks in a mobile cloud computing environment," in *Proceedings of the 2018 on Great Lakes Symposium on VLSI.* ACM, 2018, pp. 111–116.

[101] Y. Kang, J. Hauswald, C. Gao, A. Rovinski, T. Mudge, J. Mars, and L. Tang, "Neurosurgeon: Collaborative intelligence between the cloud and mobile edge," in *ACM SIGARCH Computer Architecture News*, vol. 45, no. 1. ACM, 2017, pp. 615–629.

[102] S. Cass, "Taking ai to the edge: Google's tpu now comes in a maker-friendly package," *IEEE Spectrum*, vol. 56, no. 5, pp. 16–17, 2019.

[103] G. Zervakis, K. Tsoumanis, S. Xydis, D. Soudris, and K. Pekmestzi, "Design-efficient approximate multiplication circuits through partial product perforation," *IEEE Trans. Very Large Scale Integr. (VLSI) Syst.*, vol. 24, no. 10, pp. 3105–3117, Oct 2016.

[104] V. Leon, G. Zervakis, S. Xydis, D. Soudris, and K. Pekmestzi, "Walking through the energy-error pareto frontier of approximate multipliers," *IEEE Micro*, vol. 38, no. 4, pp. 40–49, 2018.

[105] N. Karmarkar and R. M. Karp, "An efficient approximation scheme for the one-dimensional bin-packing problem," in *Annual Symposium on Foundations of Computer Science (sfcs 1982)*, 1982, pp. 312–320.

[106] C. Szegedy, W. Liu, Y. Jia, P. Sermanet, S. Reed, D. Anguelov, D. Erhan, V. Vanhoucke, and A. Rabinovich, "Going deeper with convolutions," in *Conf. Computer Vision and Pattern Recognition (CVPR)*, 2015, pp. 1–9.

[107] K. He, X. Zhang, S. Ren, and J. Sun, "Deep residual learning for image recognition," in *Proceedings of the IEEE conference on computer vision and pattern recognition*, 2016, pp. 770–778.

[108] M. Sandler, A. Howard, M. Zhu, A. Zhmoginov, and L.-C. Chen, "Mobilenetv2: Inverted residuals and linear bottlenecks," in *Conf. Computer Vision and Pattern Recognition (CVPR)*, 2018.

[109] X. Zhang *et al.*, "Shufflenet: An extremely efficient convolutional neural network for mobile devices," in *Conf. Computer Vision and Pattern Recognition (CVPR)*, 2018, pp. 6848–6856.

[110] A. Krizhevsky, G. Hinton *et al.*, "Learning multiple layers of features from tiny images," 2009.

[111] J. Stallkamp, M. Schlipsing, J. Salmen, and C. Igel, "Man vs. computer: Benchmarking machine learning algorithms for traffic sign recognition," *Neural networks*, vol. 32, pp. 323–332, 2012.

[112] A. Mogelmose, M. M. Trivedi, and T. B. Moeslund, “Vision-based traffic sign detection and analysis for intelligent driver assistance systems: Perspectives and survey,” *IEEE Transactions on Intelligent Transportation Systems*, vol. 13, no. 4, pp. 1484–1497, 2012.

[113] M. Abadi, A. Agarwal, P. Barham, E. Brevdo, Z. Chen, C. Citro, G. S. Corrado, A. Davis, J. Dean, M. Devin *et al.*, “Tensorflow: Large-scale machine learning on heterogeneous distributed systems,” *arXiv preprint arXiv:1603.04467*, 2016.

[114] F. Vaverka, V. Mrazek, Z. Vasicek, and L. Sekanina, “Tfapprox: Towards a fast emulation of dnn approximate hardware accelerators on gpu,” in *Design, Automation & Test in Europe Conference & Exhibition (DATE)*, 2020, p. 294–297.

[115] N. Irtija, I. Anagnostopoulos, G. Zervakis, E. E. Tsiropoulou, H. Amrouch, and J. Henkel, “Energy efficient edge computing enabled by satisfaction games and approximate computing,” *IEEE Transactions on Green Communications and Networking*, 2021.

[116] D. Sikeridis, E. E. Tsiropoulou, M. Devetsikiotis, and S. Papavassiliou, “Wireless powered public safety iot: A uav-assisted adaptive-learning approach towards energy efficiency,” *Journal of Network and Computer Applications*, vol. 123, pp. 69–79, 2018.

[117] J. S. Park *et al.*, “9.5 a 6k-mac feature-map-sparsity-aware neural processing unit in 5nm flagship mobile soc,” in *IEEE Int. Solid- State Circuits Conference (ISSCC)*, vol. 64, 2021, pp. 152–154.

[118] R. Banner, Y. Nahshan, and D. Soudry, “Post training 4-bit quantization of convolutional networks for rapid-deployment,” in *Advances in Neural Information Processing Systems*, 2019, pp. 7950–7958.

[119] R. Krishnamoorthi, “Quantizing deep convolutional networks for efficient inference: A whitepaper,” *arXiv preprint arXiv:1806.08342*, 2018.

[120] J. Fang, A. Shafiee, H. Abdel-Aziz, D. Thorsley, G. Georgiadis, and J. H. Hassoun,

"Post-training piecewise linear quantization for deep neural networks," in *European Conference on Computer Vision*, 2020.

[121] Y. Nahshan *et al.*, "Loss aware post-training quantization," *arXiv preprint arXiv:1911.07190*, 2019.

[122] Y. Li, R. Gong, X. Tan, Y. Yang, P. Hu, Q. Zhang, F. Yu, W. Wang, and S. Gu, "Brecq: Pushing the limit of post-training quantization by block reconstruction," *arXiv preprint arXiv:2102.05426*, 2021.

[123] B. Kim, D. Lee, Y. Ro, Y. Jeon, S. J. Kwon, B. Park, and D. Oh, "Q-rater: Non-convex optimization for post-training uniform quantization," *arXiv preprint arXiv:2105.01868*, 2021.

[124] P. Wang, Q. Chen, X. He, and J. Cheng, "Towards accurate post-training network quantization via bit-split and stitching," in *International Conference on Machine Learning.* PMLR, 2020, pp. 9847–9856.

[125] A. Vega, A. Amarnath, J.-D. Wellman, H. Kassa, S. Pal, H. Franke, A. Buyuktosunoglu, R. Dreslinski, and P. Bose, "Stomp: A tool for evaluation of scheduling policies in heterogeneous multi-processors," *arXiv:2007.14371*, 2020.

[126] T. Nghiem, *et al.*, "Monte-carlo techniques for falsification of temporal properties of non-linear hybrid systems," in *Proceedings of the 13th ACM International Conference on Hybrid Systems: Computation and Control.* ACM Press, 2010, pp. 211–220.

[127] O. Maler *et al.*, "Monitoring temporal properties of continuous signals," in *Proceedings of FORMATS-FTRTFT*, 2004.

[128] G. E. Fainekos *et al.*, "Robustness of temporal logic specifications," in *Formal Approaches to Testing and Runtime Verification*, ser. LNCS, vol. 4262. Springer, 2006, pp. 178–192.

[129] E. Bartocci *et al.*, "Specification-based monitoring of cyber-physical systems: A survey on theory, tools and applications," in *Lectures on Runtime Verification -*

Introductory and Advanced Topics, ser. LNCS. Springer, 2018, vol. 10457, pp. 128–168.

[130] E. Asarin *et al.*, “Parametric identification of temporal properties,” in *Runtime Verification.* Springer, 2012, pp. 147–160.

[131] G. Fainekos *et al.*, “Robustness of specifications and its applications to falsification, parameter mining, and runtime monitoring with s-taliro,” in *Runtime Verification (RV)*, 2019.

[132] G. Zervakis *et al.*, “Vader: Voltage-driven netlist pruning for cross-layer approximate arithmetic circuits,” *IEEE Transactions on VLSI Systems*, 2019.

[133] K. Samal, M. Wolf, and S. Mukhopadhyay, “Attention-based activation pruning to reduce data movement in real-time ai: A case-study on local motion planning in autonomous vehicles,” *IEEE Journal on Emerging and Selected Topics in Circuits and Systems*, vol. 10, no. 3, pp. 306–319, 2020.

[134] O. Maler *et al.*, “Monitoring temporal properties of continuous signals,” in *Formal Techniques, Modelling and Analysis of Timed and Fault-Tolerant Systems.* Springer, 2004, pp. 152–166.

[135] B. Hoxha, A. Dokhanchi, and G. Fainekos, “Mining parametric temporal logic properties in model-based design for cyber-physical systems,” *International Journal on Software Tools for Technology Transfer*, vol. 20, no. 1, pp. 79–93, 2018.

[136] H. Abbas, B. Hoxha, G. Fainekos, and K. Ueda, “Robustness-guided temporal logic testing and verification for stochastic cyber-physical systems,” in *Annu. IEEE Int. Conf. Cyber Technology in Automation, Control and Intelligent*, 2014, pp. 1–6.

[137] C. Szegedy, V. Vanhoucke, S. Ioffe, J. Shlens, and Z. Wojna, “Rethinking the inception architecture for computer vision,” in *Conf. Computer Vision and Pattern Recognition (CVPR)*, 2016, pp. 2818–2826.

[138] A. G. Howard, M. Zhu, B. Chen, D. Kalenichenko, W. Wang, T. Weyand, M. Andreetto, and H. Adam, “Mobilenets: Efficient convolutional neural networks

for mobile vision applications," *preprint arXiv:1704.04861*, 2017.

[139] B. Zoph, V. Vasudevan, J. Shlens, and Q. V. Le, "Learning transferable architectures for scalable image recognition," in *Conf. Computer Vision and Pattern Recognition (CVPR)*, 2018, pp. 8697–8710.

[140] K. Simonyan and A. Zisserman, "Very deep convolutional networks for large-scale image recognition," *arXiv preprint arXiv:1409.1556*, 2014.

[141] J. Cralley *et al.*, "Tltk: A toolbox for parallel robustness computation of temporal logic specifications," in *International Conference on Runtime Verification*. Springer, 2020, pp. 404–416.

[142] J. A. Nelder and R. Mead, "A simplex method for function minimization," *The computer journal*, vol. 7, no. 4, pp. 308–313, 1965.

[143] H. Sharma *et al.*, "Bit fusion: Bit-level dynamically composable architecture for accelerating deep neural network," in *2018 ACM/IEEE 45th Annual International Symposium on Computer Architecture (ISCA)*. IEEE, 2018, pp. 764–775.

[144] N. Zmora *et al.*, "Neural network distiller: A python package for dnn compression research," *arXiv preprint arXiv:1910.12232*, 2019.

[145] "S-taliro: Temporal logic falsification of cyber-physical systems," https://sites.google.com/a/asu.edu/s-taliro/s-taliro, 2015.

[146] L.-N. Pouchet, "Polybench: The polyhedral benchmark suite," *URL: http://www. cs. ucla. edu/pouchet/software/polybench*, 2012.

[147] J. D. McCalpin, "A survey of memory bandwidth and machine balance in current high performance computers," *IEEE TCCA Newsletter*, 1995.

[148] https://www.nvidia.com/en-us/autonomous-machines/embedded-systems/, [Online;].

VITA

Graduate School
Southern Illinois University Carbondale

Ourania Spantidi

Southern Illinois University
Master of Science, Computer Science, December 2018

Special Honors and Awards:

- Best Research Video Award at the Design Automation Conference 2021 and 2022
- Student Participation Grant Winner at the International Symposium on Circuits and Systems 2022

Dissertation Paper Title:
Resource-Aware Optimization Techniques for Machine Learning Inference on Heterogeneous Embedded Systems

Major Professor: Dr. I. Anagnostopoulos

Publications:

Journals

(J5) O. Spantidi, G. Zervakis, I. Anagnostopoulos, J. Henkel. Energy-efficient DNN Inference on Approximate Accelerators Through Formal Property Exploration, *IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems (TCAD)* (2022).

(J4) O. Spantidi, G. Zervakis, S. Salamin, I. Roman-Ballesteros, J. Henkel, H. Amrouch, I. Anagnostopoulos. Targeting DNN Inference via Efficient Utilization of Heterogeneous Precision DNN Accelerators, *IEEE Transactions on Emerging Topics in Computing (TETC)* (2022).

(J3) G. Zervakis, I. Anagnostopoulos, S. Salamin, O. Spantidi, I. Roman-Ballesteros, J. Henkel, H. Amrouch. Thermal-Aware Design for Approximate DNN Accelerators, *IEEE Transactions on Computers (TC)* (2022).

(J2) S. Kundan, O. Spantidi, I. Anagnostopoulos. Online frequency-based performance and power estimation for clustered multi-processor systems, *Computers & Electrical Engineering* (2021).

(J1) C. Mousas, D. Anastasiou, O. Spantidi. The effects of appearance and motion of virtual characters on emotional reactivity, *Computers in Human Behavior* (2018).

Conferences

(C9) O. Spantidi, I. Anagnostopoulos. Automated Energy-Efficient DNN Compression under Fine-Grain Accuracy Constraints, *Design, Automation & Test in Europe Conference & Exhibition (DATE)* 2023.

(C8) O. Spantidi, I. Anagnostopoulos. How much is too much error? Analyzing the impact of approximate multipliers on DNNs, *23rd International Symposium on Quality Electronic Design (ISQED)* 2022.

(C7) O. Spantidi, T. Marinakis, I. Anagnostopoulos. Fair Scheduling Through Collaborative Filtering on Multicore Systems, *IEEE International Symposium on Circuits and Systems (ISCAS)* 2022.

(C6) O. Spantidi, G. Zervakis, I. Anagnostopoulos, H. Amrouch, J. Henkel. Positive/Negative Approximate Multipliers for DNN Accelerators, *IEEE/ACM 40th International Conference On Computer Aided Design (ICCAD)* 2021.

(C5) G. Zervakis, O. Spantidi, I. Anagnostopoulos, H. Amrouch, J. Henkel. Control Variate Approximation for DNN Accelerators, *IEEE/ACM Design Automation Conference (DAC)* 2021.

(C4) O. Spantidi, I. Anagnostopoulos, G. Fainekos. Efficient Resource Management of Clustered Multi-Processor Systems Through Formal Property Exploration, *Design, Automation & Test in Europe Conference & Exhibition (DATE)* 2021.

(C3) S. Salamin, G. Zervakis, O. Spantidi, I. Anagnostopoulos, J. Henkel, H. Amrouch. Reliability-Aware Quantization for Anti-Aging NPUs, *Design, Automation & Test in Europe Conference & Exhibition (DATE)* 2021.

(C2) J. Cralley, O. Spantidi, B. Hoxha, G. Fainekos. TLTk: A Toolbox for Parallel Robustness Computation of Temporal Logic Specifications, *International Conference on Runtime Verification (RV)* 2020.

(C1) O. Spantidi, I. Galanis, I. Anagnostopoulos. Frequency-based Power Efficiency Improvement of CNNs on Heterogeneous IoT Computing Systems, *6th World Forum on Internet of Things (WF-IoT)* 2020.