Machine Learning based Memory Load Approximation

Alain Aoun

A Thesis

in

The Department

of

Electrical and Computer Engineering

Presented in Partial Fulfillment of the Requirements

for the Degree of

Doctor of Philosophy (Electrical and Computer Engineering) at

Concordia University

Montréal, Québec, Canada

July 2025

© Alain Aoun, 2025

Concordia University

School of Graduate Studies

This is to certify that the thesis prepared

By: **Mr. Alain Aoun**

Entitled: **Machine Learning based Memory Load Approximation**

and submitted in partial fulfillment of the requirements for the degree of

Doctor of Philosophy (Electrical and Computer Engineering)

complies with the regulations of this University and meets the accepted standards with respect to originality and quality.

Signed by the Final Examining Committee:

_____ Chair
Dr. Name of the Chair

_____ External Examiner
Dr. Jean Pierre David

_____ Examiner
Dr. Otmane Ait Mohamed

_____ Examiner
Dr. Sebastien Lebeux

_____ Examiner
Dr. Joey Paquet

_____ Supervisor
Dr. Sofiène Tahar

Approved by _____
Abdelwahab Hamou-Lhadj, Chair
Department of Electrical and Computer Engineering

July 2025 _____
Mourad Debbabi, Dean
Faculty of Engineering and Computer Science

Abstract

Machine Learning based Memory Load Approximation

Alain Aoun, Ph.D.

Concordia University, 2025

Modern computing applications demand ever-increasing performance and energy efficiency. However, conventional processor architectures frequently stall while waiting for data retrieval from memory, creating a bottleneck known as the *memory wall*. Over the past decades, various approaches such as speculative prefetching, load value prediction, and hardware caching have been proposed to mitigate this limitation. While these techniques yield moderate gains, they often rely on rigid hardware logic or simple pattern matching, which struggle with the irregular, data-driven workloads typical of contemporary multimedia and machine learning applications.

This thesis propose to use Machine Learning (ML) to speculate load values and reduce memory accesses. The proposed method is grounded in the principles of Approximate Computing (AC), where minor inaccuracies are accepted in exchange for improvements in performance or efficiency. To this end, we introduce an ML-based Load Value Approximation (ML-LVA) approach, which predicts the values of memory loads to reduce access latency. The ML-LVA is trained offline to generate a compact predictor that captures patterns in image and audio data, enabling accurate value prediction during runtime without the need for continual retraining. By learning

spatial correlations among adjacent data values, the proposed ML-LVA effectively anticipates memory contents, thereby reducing stalls and improving overall system performance in online deployment.

We have implemented the proposed ML-LVA framework both in software and hardware. The software variant targets existing processors lacking reconfigurability, as well as systems with tight area or power constraints that prohibit adding custom hardware. It operates as a callable subroutine designed for seamless integration without modifying the processor architecture. The software implementation was tested on an x86 processor in the GEM5 simulator. On the other hand, the hardware-based implementation integrates the proposed ML-LVA as a dedicated accelerator accessed via a custom instruction, offering tighter pipeline integration, lower latency, and enhanced efficiency for newly designed systems. The hardware-based ML-LVA was implemented in CVA6, which is an open source RISC-V processor. The synthesis results conducted in Cadence Innovus showed that the overhead of the added accelerator is marginal.

Experimental results conducted on audio and image processing workloads demonstrate that the proposed ML-LVA accelerates memory access by over 6×, resulting in application speedups up to 2.45×. Additionally, even when predicting up to 95% of loads, the output fidelity remains within perceptual thresholds. Subsequently, the proposed ML-LVA outperforms state-of-the-art LVAs in terms of performance and quality. The ML-LVA achieves these results with only a 5% area overhead and less than 1% power increase in silicon.

In loving memory of my grandmother, who passed away during my PhD,

and my Godfather, my guardian angels,

To my father, my mother, my sister and brothers.

Acknowledgments

First, I would like to thank my supervisor, Prof. Sofiène Tahar, for his valuable feedback, support, and encouragement throughout my PhD thesis. He was always available to share his experience and knowledge. His continuous belief and push through the proposed PhD project was a great essence in delivering the final output we present in this thesis. Moreover, during my study under his supervision, I learned a lot about research which helped me improving my abilities in this field. Also, I am grateful for the valuable feedback that I received from a colleague who became a dear friend, Dr. Mahmoud Masadeh. The priceless brainstorming sessions, the long meetings and sharing of his experience throughout my research were corner stones of my work. I am extremely grateful and forever beholden for his continuous availability and encouragement throughout my PhD journey.

I am also deeply grateful for Prof. Abdallah Kassem who was my supervisor at Notre Dame University for my undergraduate studies. Through his endless support and feedback, I acquired a lot of knowledge that was the essence for my graduate studies.

I would also like to express my gratitude for Dr. Jean Pierre David, Dr. Joey Paquet, Dr. Otmane Ait Mohamed, and Dr Sebastian Lebeux for serving on my advisory thesis committee and taking the time from their busy schedules to read and evaluate my thesis.

I am deeply thankful to my friends and colleagues at HVG, Elif, Kübra, and Oumaima, whose support made the PhD journey significantly more manageable. Thank you for listening patiently, thinking out loud with me, offering valuable advice, and always making the lab feel like home with a warm cup of tea and cookies at just the right time.

Finally, with all my love and gratitude, I want to thank my dearest family, Farid, Mona, Georges, Youssef and Lamia. Your unconditional love, endless patience, and unwavering support have been my anchor throughout this journey and every stage of my life. There were moments of doubt, exhaustion, and frustration, but your constant presence reminded me of my strength and my purpose. You believed in me when I struggled to believe in myself, and that made all the difference. Your sacrifices, encouragement, and quiet resilience have shaped who I am today. This PhD is not just my achievement—it is ours. I am forever grateful to you, and I carry your love with me always.

Last but by no mean the least, I would like to thank Michêle for the endless love, support and patient throughout my PhD study. Her constant encouragement made even the hardest days feel lighter, and I am truly grateful to have had her by my side.

Table of Contents

List of Figures

List of Tables

List of Abbreviations

AC	Approximate Computing
AMBA	Advanced Microcontroller Bus Architecture
ANC	Active Noise Cancellation
ASIC	Application-Specific Integrated Circuit
AXI	Advanced eXtensible Interface
BER	Bit-Error Rate
BW	Bandwidth
CAS	Column Address Strobe
CPU	Central Processing Unit
DDR	Double Data Rate
DRAM	Dynamic Random-Access Memory
ECC	Error-Correcting Code
ED	Error Distance
EDP	Energy-Delay Product
EEPROM	Electrically Erasable Programmable Read-Only Memory
FLOPS	Floating Point Operations per Second

FN	False Negative
FP	False Positive
FPGA	Field-Programmable Gate Array
GHB	Global History Buffer
GPDK	Cadence Generic Process Design Kit
GPU	Graphics Processing Unit
HD	High Definition
HPC	High-Performance Computing
ID	Instruction Decode
IoT	Internet of Things
ISA	Instruction Set Architecture
LHB	Local History Buffer
LLC	Last-Level Cache
LSB	Least Significant Bit
LVA	Load Value Approximation
LVS	Load Value Speculation
MAE	Mean Absolute Error
ML	Machine Learning
ML-LVA	Machine Learning-based Load Value Approximator
MSB	Most Significant Bit
MSE	Mean Squared Root

MSHR	Miss Status Holding Registers
NMAE	Normalized Mean Absolute Error
NRMSE	Normalized Root Mean Squared Error
OCR	Optical Character Recognition
PADP	Power-Area-Delay Product
PARSEC	Princeton Application Repository for Shared-Memory Computers
PC	Program Counter
PIM	Processing-in-Memory
PSNR	Peak Signal-to-Noise Ratio
PULP	Parallel Ultra Low Power
RAM	Random-Access Memory
RED	Relative Error Distance
RFVP	Rollback-Free Value Prediction
RISC-V	Reduced Instruction Set Computer Version 5
RM	Ring Modulation
RMSE	Root Mean Squared Error
ROB	Reorder Buffer
ROM	Read-Only Memory
SD	Standard Definition
SoC	System-on-Chip
SRAM	Static Random-Access Memory

tCK	Clock Cycle Time
TLB	Translation Lookaside Buffer
TN	True Negative
TP	True Positive
tRCD	Row to Column Delay
tRP	Row Precharge Time
VLSI	Very Large-Scale Integration

Chapter 1

Introduction

1.1 Motivation

The relentless pursuit of performance in computer architecture has always been linked to the challenge of memory latency, and the critical delay between a processor's demand for data and its eventual retrieval. This latency is not just a technical obstacle but a fundamental consequence of the von Neumann architecture [1], conceived in the mid-20th century, which tightly couples computation and memory. Early systems, such as the IBM 704 (1954), delivered a delicate equilibrium between processor speed and memory access times. However, the advent of Moore's Law in 1965 disrupted this balance, where the transistor densities began doubling roughly every 18–24 months leading to exponential gains in clock frequencies and instruction throughput. On the other hand, memory subsystems—constrained by the analog physics of charge storage and the resistive-capacitive delays of interconnects—could not keep pace. This gap in performance of memory led to slowing down computers. For instance, early processors when faced with a single cache miss or data dependency could halt the entire processor for dozens of cycles. For example, the Intel 8086's 6-stage pipeline, running at 5

MHz, could stall for more than 20 cycles (approximately 4 µs) on a memory access miss, a stark contrast to its average instruction throughput of roughly 200 ns. By the 1990s, this growing gap had been crystallized into what Wulf and McKee famously termed the "memory wall" [2]. This gap is a result of Central Processing Unit (CPU) performance improvements that far outstripped reductions in memory latency, leaving even the most advanced processors chronically starved for data. This phenomenon of memory wall still holds in the modern days. Figure 1.1 shows the performance improvement of the computational units, i.e., processors, Dynamic Random-Access Memory (DRAM) memory and processor-memory interlink bandwidth since the mid-90s. The improvement of Floating Point Operations per Second (FLOPS) of the computations units have increased at rate of 6000× in the last 20 years compared to a much smaller 100× in DRAM Bandwidth (BW). Subsequently, we can conclude an increasingly widening gap between performance improvement of the computational units and the one of the memory.

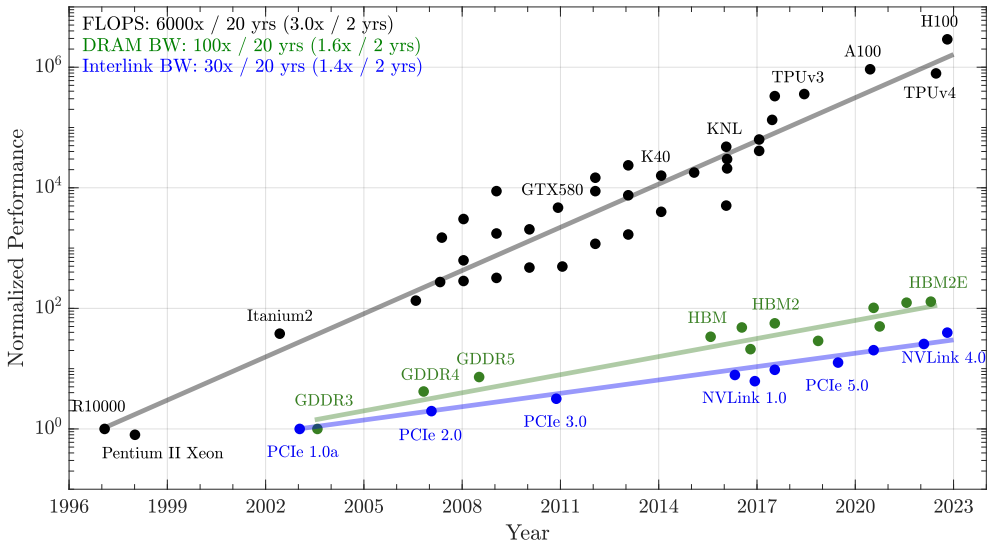


Figure 1.1: Normalized Performance over the Years [3]

To overcome the widening gap in performance, architects have redirected their interest to advancements in microarchitecture of computers to mitigate the memory wall. For instance, out-of-order execution aims at reordering instructions on the fly to hide the latency of long instructions. Architectures such as the Intel P6 [4] and AMD's K7 [5] further advanced this concept by integrating larger Reorder Buffers (ROBs) and employing register renaming to eliminate false dependencies, namely, write-after-read and read-after-write hazards. For instance, the AMD K7 utilized a 72-entry ROB along with multiple reservation stations, enabling dynamic reordering of loads, stores, and arithmetic operations. Modern CPUs offer more than 500 ROBs, however, on Last-Level Cache (LLC) misses, the ROBs can easily get saturated causing the CPU to stall [6]. Speculative techniques [7] further extend the benefits of these microarchitectural innovations. Hardware prefetchers, such as the stride-based mechanism in AMD's Zen 3 microarchitecture [8], work to anticipate sequential memory accesses. Additionally, Machine Learning (ML) [9] driven prefetchers such as DeepPrefetcher [10], leverage ML techniques to improve prefetching. For instance, the authors of [10] used deep neural networks to predict memory access patterns more accurately. However, over-aggressive prefetching can yield to cache pollution and increased miss rates yielding to overall performance degradation [11]. In a different approach to improve memory performance, AMD's 3D V-Cache [12] demonstrated that vertically stacking cache layers can effectively increase capacity and reduce the apparent access latency.

Nevertheless, even with all these advancements in microarchitecture, modern processors cannot fully address the memory wall. For instance, the developer of Blosc [13], a high performance data compressor, studied the performance improvement when the number of threads increases [13]. Figure 1.2 shows the memory bandwidth

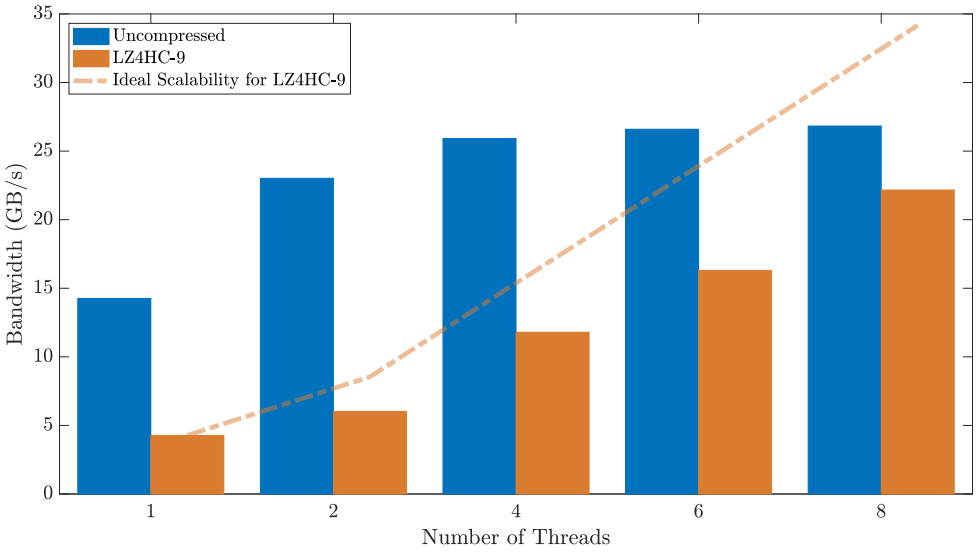


Figure 1.2: Memory Bandwidth when Increasing the Number of Thread versus the Ideal Scalability [13]

usage when increasing the number of threads used on the Intel Xeon E3-1245 v5 4-Core processor [14]. It highlights that when using uncompressed data, the bandwidth scalability does not scale linearly, i.e., ideal scalability. Consequently, the threads do not achieve maximum throughput, resulting in idle periods caused by data starvation. On the other hand, when the data is compressed with LZ4HC-9 compressor [15], a single thread requires less bandwidth, improving scalability as more threads are used. However, when operating with eight threads, the CPU receives 35% (12GB/s) less bandwidth than expected under ideal scalability, where no memory wall exists. Thus, even software techniques that aim to complement the advanced microarchitectural techniques do not fully address the memory wall. In the face of these persistent challenges, emerging paradigms such as Processing-In-Memory (PIM) [16] have been proposed to bridge the gap between processor speed and memory performance. PIM shifts repetitive computations, e.g., multiply and accumulate operations, from the CPU to memory or its vicinity to reduce data movement [17]. Although PIM minimizes processor-memory traffic, its computational speed is hindered by the slow access time

in a Random Access Memory (RAM). This fundamental limitation makes PIM a relatively slow process. In a different approach, researchers have explored the concept of *load value speculation* (LVS) [7] which involves adding a unit to the processor that predicts the value to be loaded from the memory, with minimal modifications to the Von Neumann architecture. Similar to branch prediction, if the LVS is wrong, the CPU reverts to its state before the prediction and flushes the pipeline. However, all LVS techniques aim to hide memory access delays while requiring a check to the correctness of the predictions. Thus, the bandwidth usage is not reduced but rather the latency is hidden. Another disadvantage of the LVS is the costly roll-back on a wrong speculation.

In a different approach, Approximate Computing (AC) [18] has reemerged as a pragmatic technique to further improve the performance of computers. Recognizing that many modern applications can tolerate minor inaccuracies, e.g., multimedia processing, machine learning and video games, AC intentionally relaxes precision requirements in order to reduce computational overhead, lower latency, and conserve energy. AC represents a paradigm shift from traditional exact computation by deliberately introducing controlled inaccuracies to boost energy efficiency, throughput, and overall system performance. AC is tailored for applications that can tolerate small deviations without significantly affecting perceived output quality. The underlying principle is to define error margins through rigorous analysis and quality-of-service guarantees so that performance gains outweight the slight loss in accuracy. For example, the work in [19] exploits the principle of AC to relax LVS. The costly roll-back in LVS can be avoided in error-tolerant applications by accepting the wrong (approximate) value predicted. The approximate version of the LVS is commonly referred to as *Load Value Approximation* (LVA). A drawback of existing LVA is the hardware overhead,

e.g., lookup table, in addition to the computation needed to compute indexing/hash values used by the predictor. Additionally, the existing LVA requires continuous access to the memory in order to maintain good quality.

Given that existing methods have failed to fully address the memory wall, in this thesis we aim to address this challenge by using ML and AC. The solution we propose can be a software- or hardware-based implementation. The predictor used in the proposed LVA is static and hence reduces the overhead of operating the predictor. In the rest of this chapter, we will present the evaluation techniques of AC designs. Thereafter, we present the state-of-the-art that is most relevant to our work, followed by a discussion on the limitations of existing methods. We then introduce the proposed methodology followed by the thesis contributions and the thesis organization.

1.2 State-of-the-Art

Various studies in the literature have investigated methods to address memory bottlenecks. Prior to approximate computing gaining popularity in recent years, researchers have focused on exploiting the localities in order to address this challenge. In the sequel, we will restrict the discussion of related work to those methods that are most relevant to the proposed solution in this thesis. We will present the approximate memory that aims to improve memory latency and energy consumption. We will also cover the Machine Learning (ML) based prefetcher which aims to predict the subsequent load value and hence improving the hit rate in the cache. Thereafter we present Load Value Speculation (LVS), a technique that aims to predict the load value and hence hide the latency. We conclude this section with the presentation of Load Value Approximation (LVA), a technique that improves the performance of LVS by tolerating error.

1.2.1 Approximate Memory

Approximate Memory consists of modifying the conventional exact memory where data can be stored approximately. For instance, the work in [20] explores an implementation of approximate DRAM. The authors of [20] propose the storage of data in a transposed fashion. Furthermore, they investigate the implementation of variable refresh rate where rows storing Least Significant Bits (LSBs) are refreshed less frequently compared to the rows storing the Most Significant Bits (MSBs). The variable refresh rate results in energy saving. In order to address the memory bottlenecks, the authors of [20] propose the load of a limited number of rows. When deploying this approach, the MSBs of multiple data elements are retrieved concurrently. Hence, more elements are retrieved faster. Furthermore, if only 16 rows out of 32 are loaded, i.e., loading the 16 MSBs and truncating the other 16 LSBs, the bandwidth would be reduced by 50%. Another approach of approximate memory in [21] proposes the compression of error-tolerant data in DRAMs. Compression and decompression in distinct memory regions define varying accuracy levels, that are controlled by a software-hardware integration. While both methods present compelling strategies for leveraging approximation in memory to reduce energy consumption and improve bandwidth efficiency, they also introduce notable limitations. For instance, in [20], the use of transposed storage and variable refresh rates is applied at the region level, allowing critical data to reside in safe, fully refreshed regions. This mitigates the risk of corruption and side channel attacks in precision-sensitive applications. However, the selective loading of only MSBs and the reduced refresh of the LSB regions may still affect the data quality in regions marked as approximate, and the complexity of managing transposed access patterns and refresh schedules introduces additional hardware overhead. Additionally, the introduction of approximate regions opens potential security vulnerabilities. However, malicious code

could exploit relaxed refresh policies or compressed representations to manipulate data silently by inducing bit flips in approximate regions. Similarly, the compression-based approach in [21] imposes latency and control challenges due to the need for dynamic decompression and region management. Its reliance on software-hardware coordination for setting approximation levels requires accurate workload characterization and tight system integration, which may limit portability and scalability. Both methods would benefit from more adaptive, runtime-aware control mechanisms that can fine-tune approximation levels based on observed application behavior and quality constraints.

1.2.2 ML-based Prefetching

Memory prefetching is a technique that predicts and loads data into the cache before it is needed, reducing access latency and improving processor performance. The authors of [10] developed DeepPrefetcher, a method that leverages ML to enhance memory access efficiency by intelligently predicting future data requests. DeepPrefetcher harnesses deep learning to identify complex correlations within memory access sequences. By analyzing execution traces, it learns nuanced dependencies that conventional prefetchers often overlook, allowing it to anticipate future accesses more effectively. This approach involves training a deep learning model on program memory traces, capturing both short-term and long-term relationships between memory accesses. DeepPrefetcher continuously refines its predictions by updating the model with new data, adapting dynamically to evolving workloads. This adaptability makes it particularly effective in scenarios where memory access patterns are irregular, non-repetitive, or workload-dependent. By leveraging its learned understanding of execution behavior, DeepPrefetcher reduces cache misses and improves data availability, leading to enhanced overall system performance. It is especially beneficial for modern

applications with complex memory access patterns, where conventional prefetching strategies struggle to keep pace. However, a common challenge among prefetchers is their potentially incorrect predictions, which can lead to cache *pollution*, i.e., the eviction of useful data by unnecessary prefetched data, and ultimately increase miss rates [11]. As a result, prefetchers can sometimes be counterproductive, as a higher number of incorrect prefetches may degrade the performance rather than improve it. Another drawback of DeepPrefetcher is its reliance on online learning, where the model must be continuously trained in real-time to maintain accurate predictions. This process requires dedicated computational resources, leading to additional hardware overhead and increased energy consumption.

1.2.3 Load Value Speculation

Load Value Speculation (LVS) aims to hide the memory latency by predicting the value to be loaded. In one of the earliest work in this area, the authors of [7] introduced the idea of *value locality*. The authors suggest that values stored in adjacent memory addresses are comparable in magnitude. For instance, the adjacent pixels of an image stored in memory are alike in value. Subsequently, the authors present a dynamic lookup table that speculates the value of a load with the aim of hiding the memory access latency. In case of wrong speculation, the processor rolls back and flushes the pipeline. Moreover, the lookup table is updated after every memory access. This concept was widely investigated by implementing other speculation techniques such as the work in [22] and [23]. Among the common techniques used in LVS are *i*) last value prediction, *ii*) stride-based prediction, *iii*) context-based prediction, *iv*) hybrid prediction, and *v*) perfect confidence [23]. Last value prediction assumes that a memory load instruction will retrieve the same value as in its most recent execution. On the

other hand, stride-based prediction estimates the next load value by identifying a consistent difference (stride) between successive values of a particular load instruction. Additionally, context-based prediction relies on the past four observed values of a load instruction to determine the most likely next value. Hybrid prediction dynamically selects the most accurate predictor between context-based and stride-based methods. Finally, perfect confidence utilizes the hybrid predictor but only applies the predicted value when it is certain to be correct, simulating an ideal confidence mechanism.

All LVS techniques access the memory to confirm the correctness of the speculation and roll-back in case of wrong speculation.

1.2.4 Load Value Approximation

Since roll-backs in LVS are expensive in terms of hardware requirement and loss in clock cycles, some researchers have proposed the idea of Load Value Approximation (LVA). These techniques speculate a value without a roll-back in case of a wrong prediction which results in approximation. For instance, the work in [19] proposed a load value approximation which relies on a dynamic predictor. The accuracy of the predictor can be improved by providing the value of the recent loads, which saves in bandwidth and energy for a cost in quality. The authors of [19] use a predictor with a Global History Buffer (GHB) to record recent load values and an approximator table indexed by a hash derived from the GHB along with the load instruction address to generate an estimated value on a cache miss. Additionally, the predictor includes a Local History Buffer (LHB), which records the sequence of recent values following patterns detected in the Global History Buffer (GHB). The approximator then estimates a value by averaging entries in the LHB, and thus allowing the processor to continue execution while the actual value is fetched in the background to train the predictor future

estimations. Unlike traditional value predictors that require an exact match and trigger rollbacks if even a slight discrepancy occurs, this approach uses a relaxed confidence window to tolerate minor differences, thereby eliminating rollbacks and enhancing energy efficiency. Furthermore, the design incorporates an approximation degree, which determines how many subsequent load misses can reuse the same predicted value before the actual data is fetched for retraining, thus effectively balancing accuracy with reduced memory fetches even when value delays occur. Although the LVA proposed in [19] introduces a solution that aims to address the memory wall, however it has the drawback of using a complex hardware to predict the load value. For instance the work in [19] requires the calculation of complex indexes and keeping track of LHB and GHB. Furthermore, it uses a dynamic predictor that requires a continuous memory access to maintain an acceptable low quality.

Rollback-Free Value Prediction (RFVP) [24] is an LVA technique designed for Graphics Processing Units (GPUs). RFVP reduces memory latency by predicting values for cache misses and selectively discarding some predictions. When a load is not discarded, the actual value retrieved from the memory updates the predictor used in the RFVP, improving its accuracy. The predictor relies on a value history table indexed using a hash derived from the program counter of the load instruction. Each entry stores three key elements, namely, the most recent value loaded from memory and two stride values that capture how the load value has changed over time. RFVP predicts a new load value by adding the last loaded value to one of the observed strides. To enhance stability, the predictor updates the stride entry only when the new stride is consistent with past patterns. Because GPU threads within a warp may not execute simultaneously, some threads can be inactive during a cache miss but may later request the same data. RFVP anticipates these accesses by precomputing

predictions for inactive threads, ensuring they receive meaningful approximations rather than outdated or random values. This approach maintains consistency and prevents errors due to unpredictable data. Similar to the LVA proposed in [19], RFVP uses of a dynamic predictor that updates through memory access to sustain an acceptable output quality. Furthermore, the RFVP predictor has a computation overhead, e.g., computation of a hash index, to approximate the load value while also requiring a lookup table that uses a complex updating method.

1.3 Problem Statement

Researchers have long investigated options to improve the performance of computers, e.g., out-of-order execution. However, most modern computers face a significant challenge in gaining speedup due to the memory wall. A limited number of researchers have explored approximation or prediction-based solutions to address memory bottlenecks. The most prominent techniques in approximating the load value are the work in [19, 24]. Existing LVA techniques rely heavily on memory access to update the predictor and hence maintain an acceptable quality. Moreover, existing methods of LVA require complex hardware that has substantial resource usage and/or require the computation of values needed to index/guide the predictor.

Alternatively, some researchers have shifted their focus towards exploring approximate memory to address the memory wall, e.g., the work in [20]. However, the approximate memory could potentially disrupt the functionality of the error correction code (ECC) in the memory. Additionally, an approximate memory has an increased complexity of the memory controller, which could yield in a significant overhead and a delay.

Given these limitations, we believe that an approach approach with minimal

overhead is better suited to achieve meaningful speedups while preserving a small hardware footprint. A key enabler for this is the use of a static predictor, which would avoid the need for continuous updates and complex runtime mechanisms. A statically designed model significantly reduces implementation complexity and allows for faster predictions, thereby decreasing latency and energy consumption. With the recent progress in ML algorithms and the increasing availability of high-performance computation resources, ML-based LVA solutions would offer a compelling alternative. Lightweight ML models, such as small neural networks or decision trees, can be trained offline to learn patterns in memory load value, enabling accurate approximations that do not require dynamic adaptation or rollbacks.

An advanced ML model can be trained once using representative workloads and a statically trained predictor used solely for runtime prediction that operates efficiently during execution. This approach eliminates the need for re-training at runtime and ensures consistent prediction performance. Such models can generalize across various applications, offering broader adaptability than traditional heuristic-based predictors. However, ML models are inherently resource-intensive, often requiring significant memory and computation. If not carefully designed, their demands can further burden the already strained memory system, thereby worsening the memory wall problem. To fully realize the benefits of ML in this context, it is essential to develop compact, statically trained models tailored for hardware efficiency, enabling low-overhead with minimal performance trade-offs.

1.4 Proposed Methodology

The objective of this thesis is to address the memory wall by using Machine Learning (ML) and Approximate Computing (AC). We argue that this challenge can

be addressed with a solution that satisfies three key criteria: *i*) improve performance, *ii*) maintain an acceptable quality, and *iii*) efficient usage of resources. Towards this goal, an ML model is trained to learn the patterns in the load values while recognizing that perfect accuracy is unfeasible. Thereafter, the trained ML model is used to predict the load value. Given the inherent inaccuracy of ML models, we accept the predicted value as-is resulting in an approximately computed load value. Subsequently, the ML model is trained to achieve a controlled quality that is deemed acceptable. We propose to develop a static predictor, where the ML model undergoes a single training phase and then is reused indefinitely. The proposed ML-based Load Value Approximator (ML-LVA) can be integrated into existing off-the-shelf hardware through a software implementation, or implemented as a dedicated unit in newly designed hardware. The proposed ML-LVA can be applied to any domain in which the principle of *value locality* holds. In this thesis, we focus on its implementation in multimedia applications, as these applications naturally exhibit value locality. For example, in an image, a given pixel typically has neighboring pixels with similar values. Subsequently, the proposed model in this thesis can be deployed in video games, augmented and virtual reality systems where performance is a key necessity while a slight loss due to approximation is tolerable. The details of the proposed methodology for developing and using a ML-LVA are depicted in Figure 1.3. The methodology consists of *Offline* and *Online* phases. The Offline phase is used to build the LVA predictor, while the Online phase represents the application's execution stage. The Offline phase requires three inputs: *i*) an Error Tolerant Application, *ii*) a Training Multimedia Dataset, and *iii*) an Approximation Level (n). Using these three inputs, the Offline stage produces an approximated error tolerant application as an output, which will be used by the Online phase to generate as an output the approximated multimedia

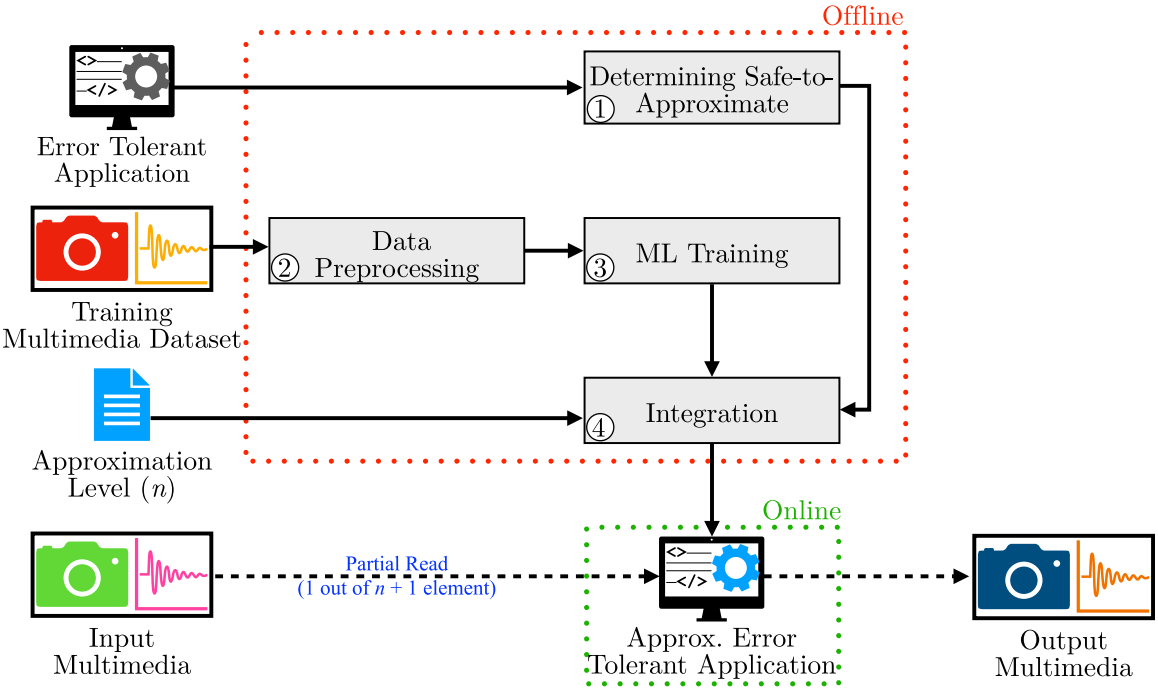


Figure 1.3: Methodology of the Proposed ML-LVA

result. The Offline phase consists of four main steps. In Step ① the error tolerant application is profiled to determine the "safe-to-approximate" load instructions. This is a critical step in the process since approximating load values that do not tolerate error could lead to catastrophic scenarios. For instance, loading a loop boundary or memory addresses cannot be approximated. On the other hand, loading multimedia data, e.g., pixel of an image, is error-tolerant and thus can be determined as "safe-to-approximate". The identification of safe-to-approximate load instructions can be performed at either the high-level programming or low-level (assembly) code. For applications written in a high-level language, the programming language must provide a mechanism to explicitly declare variables as safe-to-approximate, and the compiler must be extended to recognize these declarations and generate the corresponding approximations. Alternatively, when the application is provided as assembly code, the analysis can be performed directly at the instruction level. Although in this thesis

the identification of safe-to-approximate load instructions is performed manually, this process could be automated in the future using, for example, some machine learning technique, which can evaluate the impact of approximation and automatically determine the safe-to-approximate instructions. In parallel, in Step ② we perform a data preprocessing on the training multimedia dataset. This step is crucial as raw data cannot be fed to an ML model to perform training. Additionally, in this step, we split the data into training and testing data in order to validate the quality of the ML model. Thereafter, we perform the ML training in Step ③. The trained model will serve as the load value predictor, i.e., the ML-LVA. In Step ④, we integrate the ML-LVA into the error-tolerant application, where a portion of the safe-to-approximate load instructions is replaced. The ratio of replacement is defined by the approximation level (n), meaning that we retain 1 out of $n + 1$ instructions as exact while the others are approximated. Every approximated instruction becomes a call to a subroutine, invoking a custom instruction that can be implemented in software or in hardware.

In the software-based approach, this custom instruction is realized as a conventional function call to a software routine that performs load value prediction using the trained ML model. This solution is intended for deployment in existing off-the-shelf hardware platforms, where hardware modifications are either impractical or impossible. As such, it enables the integration of approximate load value prediction without requiring architectural changes. To evaluate this method, the software-based implementation of the ML-LVA was deployed in GEM5 [25], simulating an x86-based architecture. In this evaluation, the "*Timing Simple CPU*" [26] was used, and various CPU frequencies, cache settings and DRAM configurations were tested to assess the impact of the memory hierarchy on performance gains. In contrast, in the hardware-based implementation, the custom instruction is backed by a dedicated hardware accelerator that executes the

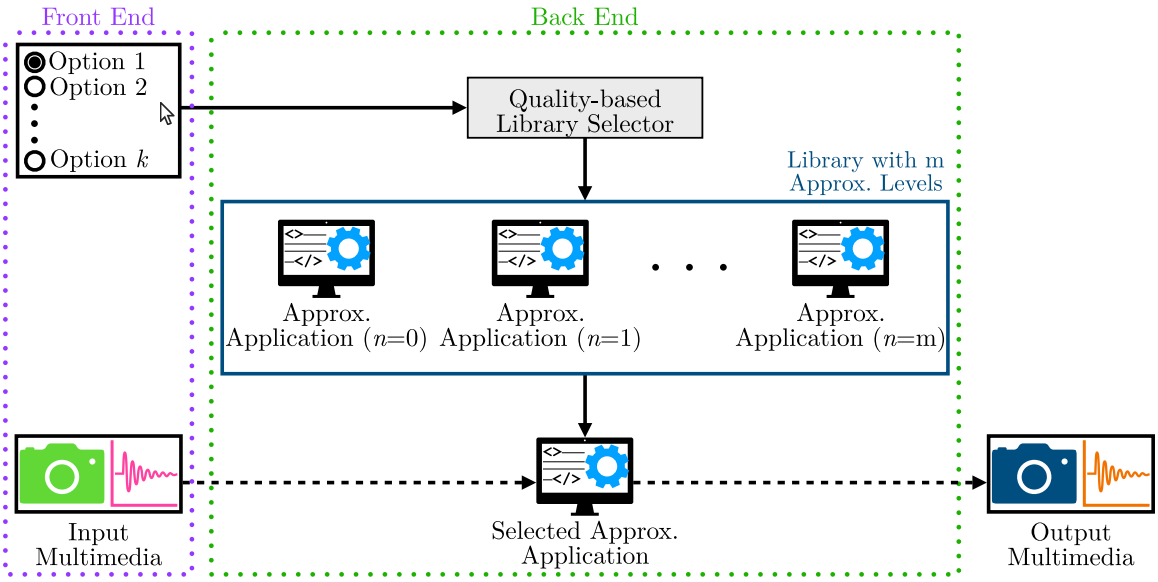


Figure 1.4: Deployment Framework for the Proposed Methodology

ML prediction logic directly in silicon. This accelerator acts as a coprocessor integrated into a newly designed processor pipeline. In this thesis, the hardware ML-LVA was implemented within the CVA6 design [27], a RISC-V processor [28], extending its instruction set architecture (ISA) to include a custom opcode. When the approximated instruction is encountered during execution, it triggers the custom instruction, which routes the request to the ML-LVA accelerator. This tightly coupled design eliminates software overhead and significantly reduces latency, making it suitable for high-performance systems where hardware design flexibility is available. By supporting both software and hardware realizations, the ML-LVA framework offers flexibility for deployment across a wide spectrum of systems—from commodity platforms to next-generation processor designs. In both the software and hardware implementations, we investigate the ML-LVA framework for 8-bit applications. Nonetheless, certain design choices have been made to ensure that the methodology can scale effectively to larger designs, such as 16-bit or 32-bit applications.

The proposed methodology can be deployed by a user as shown in Figure 1.4. First,

the back end developer creates multiple versions of the same approximated application with each version corresponding to a specific approximate level (n). Subsequently, in the front end, the user selects the desired quality from an option menu, similar to how a user selects video quality, such as *Standard Definition* (SD), *High Definition* (HD), or 4K, in a streaming service. Additionally, the user provides the input multimedia to the application. A selector function in the back end then picks the application variant from the library that best matches the user's chosen quality level. For instance, the user might select SD to prioritize performance at the cost of reduced output quality. Internally, the developer maps this SD setting to, for example, an approximation level of 4, i.e., $n = 4$, which corresponds to an 80% approximation. Finally, the selected application processes the input multimedia to produce the output. The deployment, shown in Figure 1.4, could be further improved by compiling a single version of the application into an executable file –commonly referred to as the *binary* or *binary code*– and then passing the approximate level (n) dynamically. For example, instead of compiling a separate binary for each approximation level, a single application binary code can be compiled to accept the approximation level (n) as a runtime argument. This value can be passed via the argument vector, i.e., *argv*, when launching the application. At runtime, the program retrieves this argument to determine how aggressively to approximate. The mechanism for accessing the argument may vary depending on the underlying architecture. For instance, on x86 it may be fetched from the stack, while on RISC-V it may be accessed through a register.

To validate the quality and effectiveness of the proposed ML-LVA methodology we evaluate it using a diverse set of multimedia applications. These include image processing, i.e., image blending, image inversion, and image binarization, as well as audio processing tasks such as audio blending, audio inversion, and audio binarization.

These applications were selected due to their error-tolerant nature and their dependence on memory-intensive operations, making them ideal benchmarks for assessing the impact of approximate load value prediction. Experimental results indicate that the trained ML-LVA was able to achieve a quality that outperforms state-of-the-art. Similarly, both the software- and hardware-based implementations of proposed ML-LVA outperform existing LVA techniques where substantial speedups both application overall as well as in memory loads were measured.

1.5 Thesis Contributions

The main contribution of this thesis is overcoming the memory wall by leveraging Machine Learning (ML) and Approximate Computing (AC) for memory load value approximation. The proposed methodology addresses this challenge by reducing the memory bandwidth needed by a given application and approximating (predicting) the values that were not loaded from the memory. The predictor used in thesis is generated by training an ML model. The key contributions of this work are outlined below, with references to relevant publications included in the Biography section at the end of the thesis:

- A methodology for machine learning-based load value approximator (ML-LVA) which presents a complete design and deployment framework that includes identifying "safe-to-approximate" load instructions, performing offline training of an ML model to predict these values, and integrating the predictor into error-tolerant applications. The output quality is controlled via a tunable approximation level. The methodology supports both software- and hardware-based realizations and enables quality-performance trade-offs through user-configurable settings [Bio-Cf4].

- A static machine learning model that predicts unfetched memory load values to reduce access latency. By adopting a static approach, the proposed ML-LVA eliminates the deployment overhead of a dynamic predictor. The proposed ML-LVA consistently produced high-quality results across six multimedia applications, with output errors typically below 10%. In some cases, the output was indistinguishable from the original. For classification based-applications, accuracy and precision exceeded 98% at 50% approximation and remained above 80% even at 95% approximation [Bio-Cf1, Bio-Cf5].

- A software-based implementation of the proposed methodology deploying the ML-LVA as a callable subroutine on commercial hardware. Safe-to-approximate load instructions are replaced with subroutine calls to enable approximate execution via the trained ML model. We evaluate this solution on six diverse multimedia applications using the GEM5 simulator configured for an x86 architecture. The implementation delivered overall application speedups of up to 1.23×, and more than 6× faster value loading in multiple cases [Bio-Jr1].

- A hardware-based implementation integrating the ML-LVA predictor as an accelerator within the CVA6 RISC-V processor [27] using a custom instruction. Implemented in the form of a lightweight ROM, this design enables fast, low-overhead inference without complex logic operations. We evaluate this approach on four multimedia applications, achieving overall speedups up to 1.21× and memory load gains of 1.47×. Synthesis results show that incorporating ML-LVA into CVA6 incurs modest overheads: area increases by 5.09% and power consumption rises by only 0.79% [Bio-Jr3].

1.6 Thesis Organization

The rest of this thesis is organized as follows: in Chapter 2, we introduce the preliminary notions needed to ease the understanding the remainder of this thesis. This chapter focuses on defining the key evaluation metrics used throughout the work, including quality metrics for assessing the effectiveness of the ML-LVA, and performance metrics for analyzing its impact on resource usage. Additionally, we present the mathematical background underlying the target applications, along with an explanation of how the ML-LVA influences their behavior.

In Chapter 3, we present the dataset used along with the training process and the quality of the proposed ML-LVA. We also provide the details of the input needed by the ML-LVA to predict the load value along with the prediction method. Furthermore, we evaluate the quality of the ML-LVA when simulated in audio and image blending applications. Thus, in Chapter 3 we will expand and present Steps ② and ③ of the proposed methodology shown in Figure 1.3.

In Chapter 4, we present Steps ① and ④ of the proposed methodology, namely, the software-based implementation of the proposed ML-LVA. We present in details how we determine a safe-to-approximate instructions. Furthermore, we expand the work in Step ④ by outlining the process of transforming the ML model into an optimized software solution. Furthermore, we present the measuring environment used to evaluate the efficiency of the software-based implementation which consists of simulating an x86 processor in GEM5 with various cache and DRAM configurations. Thereafter, we present the speedup achieved in memory access operations as well as the application when the software-based implementation is deployed. The performance analysis was performed using six multimedia applications, namely, image blending, image inversion, image binarization, audio blending, audio inversion and audio binarization.

Chapter 5 describes how Step ④ of the proposed methodology shown in Figure 1.3 can be adopted for a hardware-based implementation. In this chapter, we present the RISC-V CPU used and the modifications performed in order to integrate the ML-LVA. We present the custom assembly instructions that were added to the Instruction Set Architecture (ISA) permitting the usage of the ML-LVA. Furthermore, we evaluate the benefits of these new instructions in terms of speedup in memory access and application performance using multimedia applications. We provide experimental results on multimedia applications, including image blending, image inversion, audio blending and audio inversion.

We conclude this thesis in Chapter 6, where we provide closing remarks and future directions.

Chapter 2

Preliminary

2.1 Introduction

This chapter describes the metrics used to evaluate the proposed Load Value Approximation (LVA) techniques. Rather than presenting empirical results, the chapter focuses on defining the evaluation methodology and contextualizing the approximation within practical application domains. The aim is to lay the foundation for the detailed analysis that follows in the subsequent chapters.

We first introduce the mathematical formulations of the metrics used to assess the impact of LVA. These include measures related to performance, accuracy, and resource efficiency. By formalizing these concepts, the section ensures a consistent and precise basis for analyzing the effectiveness of the proposed LVA across various implementation contexts, whether hardware or software. We then present the six multimedia applications selected as representative case studies for applying LVA. This chapter describes the structure of each application and highlights the specific mathematical operations that are suitable candidates for approximation. Understanding the underlying operations permits the reader to understand how they

tolerate imprecision and how their approximation can be leveraged by the ML-LVA to enhance performance. Together, these preliminaries provide the necessary background for interpreting the evaluation results presented in later chapters.

2.2 Evaluating Approximate Computing

The proposed ML-LVA relies on Approximate Computing (AC). Therefore, it is important to understand the assessment methods used to identify the quality and gains of an AC design. In this section, we will delve into the evaluating techniques of AC.

The usability of an AC design can be determined if the delivered quality falls within an acceptable range of tolerable error. The generated quality of an AC design can be measured using different metrics that can broadly be classified into two categories: *i*) arithmetic error metrics that quantify the numerical deviation, and *ii*) classification metrics that assess correctness in binary decisions. Some of these metrics include [29]:

- **Error Distance (ED)** is the arithmetic distance between the exact value (E_v) and the approximate value (A_v) by a given set of inputs. Hence the ED can be written as:
$$ED = E_v - A_v \tag{1}$$
- **Relative Error Distance (RED)** is the ratio of the relative ED with respect to the exact value (E_v):
$$RED = \frac{ED}{E_v} \tag{2}$$
- **Mean Absolute Error (MAE)** is the average of the absolute values of all ED in space, i.e., n number of instances:
$$MAE = \left(\frac{\sum |ED|}{n}\right) \tag{3}$$

- **Normalized Mean Absolute Error (NMAE)** is measured to have a better analysis for the worst-case scenario error. $NMAE$ is normalized using ED_{max}, the maximum ED in space, e.g., $ED_{max} = 255$ for 8-bit applications, and computed as:

$$NMAE = \frac{MAE}{ED_{max}} \tag{4}$$

- **Mean Squared Error (MSE)** is the average of the ED squared:

$$MSE = \frac{\sum ED^2}{n} \tag{5}$$

- **Root Mean Squared Error (RMSE)** is the square root of MSE:

$$RMSE = \sqrt{MSE} \tag{6}$$

- **Normalized Root Mean Squared Error (NRMSE)** is computed in a similar fashion to NMAE:

$$NRMSE = \frac{RMSE}{ED_{max}} \tag{7}$$

- **Peak Signal-to-Noise Ratio (PSNR)** evaluates the quality of an image or video by comparing the original signal to the noise introduced by the new design. For 8-bit applications, the PSNR is computed as [30]:

$$PSNR = 20 \ \log_{10}\left(\frac{255}{RMSE}\right) \tag{8}$$

- **Bit-Error Rate (BER)** is the percentage of faulty bits in the output. The BER is different from all previously discussed error metrics since it disregards the arithmetic error. The BER can be expressed in terms of False Negative (FN),

False Positive (FP), True Negative (TN) and True Positive (TP) as:

$$BER = \frac{FN + FP}{FN + FP + TN + TP} \tag{9}$$

- **Accuracy** is the overall proportion of correct predictions made out of all predictions. Similar to BER, the accuracy also does not measure arithmetic error. Accuracy can be computed as:

$$Accuracy = 1 - BER = \frac{TN + TP}{FN + FP + TN + TP} \tag{10}$$

- **Precision** is the fraction of predicted positive instances that are indeed correct, emphasizing the accuracy in making positive predictions. Similar to BER and the accuracy metrics, precision does not measure arithmetic error. Precision is computed as:

$$Precision = \frac{TP}{TP + FP} \tag{11}$$

The selection criterion of the error metric is driven by the type of application. For instance, in a system that generates a true or false response, the classification metrics are the suitable ones.

In addition to error metrics, physical design metrics are essential when evaluating the practicality and efficiency of an AC design. These metrics capture the tangible costs of implementing a design in hardware and are typically expressed in terms of area (A), delay (D), and power (P). While an approximate design may tolerate some degradation in output quality, it must still offer meaningful gains in one or more physical dimensions, otherwise, the trade-off is unjustified.

To assess the overall hardware efficiency and compare different designs fairly,

composite metrics are often employed. These metrics combine multiple resource constraints into a single value, helping designers analyze trade-offs and identify optimal points along the performance-cost spectrum [31]:

- **Power-Area-Delay Product (PADP)**: This metric captures the overall hardware cost by multiplying the three primary physical factors. It is particularly useful when all three resources—power (P), area (A), and delay (D)—are equally important in the target application.

$$PADP = P \times A \times D \tag{12}$$

- **Energy-Delay Product (EDP)**: This metric evaluates the trade-off between energy consumption (E) and performance. It is especially relevant in battery-powered and energy-sensitive systems, where reducing both energy and execution time is critical. Since energy is the product of power and time ($E = P \times D$), EDP can be rewritten as:

$$EDP = E \times D = P \times D^2 \tag{13}$$

The choice between PADP, EDP, or other derived metrics often depends on the specific goals and constraints of the system being designed. For instance, in mobile and embedded devices, minimizing energy and latency is often prioritized, making EDP a suitable metric. In contrast, for high-performance computing systems where throughput is more critical, delay (execution time) or PADP might be emphasized more. Moreover, when approximation is applied at the software level, execution time becomes an even more significant factor. Faster execution not only improves

responsiveness but can also indirectly reduce total energy consumption, especially in systems where power scales with utilization. Therefore, speedup, defined as the ratio of execution time between the exact and approximate versions, is a key metric in these contexts. It provides a direct measure of how much computational efficiency is gained through approximation.

Ultimately, the effectiveness of an AC design must be evaluated in terms of both quality loss using error metrics and resource savings by analyzing the physical metrics. A good approximation strikes the right balance between these two aspects, delivering meaningful hardware and performance benefits while keeping the output within acceptable quality bounds.

2.3 Multimedia Applications

To evaluate the applicability of LVA in practical scenarios, six multimedia applications have been selected as representative case studies in this thesis. These applications span both image and audio processing tasks, relying on simple yet computationally relevant operations. Each is well-suited to approximate computing due to the inherent error tolerance in human perception. Multimedia workloads often tolerate minor deviations in data values without significantly degrading the perceived quality of the output, making them ideal for exploring trade-offs between performance and accuracy. Moreover, multimedia applications frequently process large volumes of data in parallel, which amplifies the performance benefits of techniques like LVA that reduce memory latency and computational effort. By covering a diverse set of operations—ranging from arithmetic-heavy blending to memory-bound thresholding and inversion—this selection offers a comprehensive and realistic benchmark for assessing the effectiveness

of LVA in real-world, error-resilient domains. In the sequel, we outline the applications and their corresponding mathematical operations. The multimedia applications used in this thesis are:

1. **Image Blending [32]**

   Image blending combines two images by multiplying their corresponding pixel values, optionally scaled by a blending factor. This operation is common in graphics design and visual effects. For example, for 8-bit images, the blending operation is defined as:

$$I_{\text{blend}}(x, y) = \alpha \cdot \sqrt{I_1(x, y) \cdot I_2(x, y)} \tag{14}$$

   where I_1 and I_2 are the input images, (x, y) denotes the spatial coordinates of a pixel in the image, and $\alpha \in [0, 1]$ is a blending coefficient. The square root operation ensures that the resulting values remain within the 8-bit range. Since this operation requires the calculation of the square root –a computation intensive operation– the overall speedup in the application is expected to be significantly less compared to the one in the memory access since a smaller portion is spent on the memory load.

2. **Ring Modulation (Audio Blending) [33]**

   Ring modulation blends two audio signals by pointwise multiplication of their 8-bit unsigned sample values that ranges from 0 to 255. The formula for this operation, similar to image blending, is:

$$s_{\text{mod}}(t) = \alpha \cdot \sqrt{s_1(t) \cdot s_2(t)} \tag{15}$$

where s_1 and s_2 are the two audio streams, t denotes the discrete time index, and α is the scaling factor applied to control the intensity of the effect. Similar to image blending, this operation involves two back-to-back loads, one for each audio signal. The approximation could help accelerate the processing of long audio signals by reducing the cost of these loads, leading to a reduced execution time. However, similar to the image blending, as the operation includes the calculation of the square root, the speedup achieved in the application due to the LVA is expected to be significantly less than the one observed in the memory access.

3. **Image Thresholding (Image Binarization) [34]**

   Image thresholding simplifies a grayscale image by converting it to a binary image using a fixed intensity threshold:

$$I_{\text{thresh}}(x, y) = \begin{cases} 255, & I(x, y) > T \\ 0, & I(x, y) \leq T \end{cases} \tag{16}$$

   where I is the input image, (x, y) denotes the spatial coordinates of a pixel in the image, and T is the selected threshold. This operation only requires a single load per sample followed by a simple comparison, making it more dependent on memory access compared to image blending or ring modulation. Thus, while approximating, a higher speedup in the memory access will have a higher impact on the performance improvement observed at the application level.

4. **Infinite Clipping (Audio Binarization) [35]**

   Infinite clipping transforms an audio signal by reducing each sample to a binary representation of its polarity. Given that the samples are 8-bit unsigned, they

are first centered around zero. The operation is defined as:

$$s_{\text{clip}}(t) = \begin{cases} 255, & s(t) > T \\ 0, & s(t) \leq T \end{cases} \tag{17}$$

where s is the audio stream t denotes the discrete time index, and T is the threshold. The resulting values are either 0 or 255, depending on the comparison. Similar to image thresholding, this operation involves a single load and simple computation. Thus, the benefit of the LVA reverberates more at the application level.

5. **Image Negatives (Image Inversion) [36]**

   Image negatives transform an image into its negative by subtracting each pixel value from the maximum intensity:

$$I_{\text{neg}}(x, y) = 255 - I(x, y) \tag{18}$$

where I is the input image, and (x, y) denotes the spatial coordinates of a pixel in the image. This operation is visually intuitive and widely employed in photographic effects and preprocessing. Its reliance on a straightforward subtraction operation makes it another potential candidate to test the proposed LVA. Similar to thresholding, image negatives involve a single load followed by a straightforward subtraction. Consequently, the potential speedup from LVA is higher, contributing to a larger reduction in the overall execution time, i.e., higher speedup.

6. **Audio Polarity Inversion [37]**

   Polarity inversion flips the audio waveform around its midpoint. In the 8-bit unsigned format, this is implemented by inverting the sample around 128:

$$s_{\text{inv}}(t) = 255 - s(t) \tag{19}$$

   where s is the audio stream and t denotes the discrete time index. This operation generates an inverted waveform, which is perceptually indistinguishable in numerous playback environments, rendering it suitable for evaluating approximation in subtraction or data loading. Similar to image inversion, polarity inversion entails a single load and a straightforward subtraction. Consequently, the speedup observed at the application level will be substantial.

In this thesis, specific parameters were set for each application to align with practical use cases. For instance, for the blending tasks, the blending factor α was set to 1, reflecting typical settings in image editing [32] and audio mixing tools [38]. For image thresholding, the threshold was selected using Otsu's method [39], a standard approach in optical character recognition (OCR) to eliminate noise in scanned images [40]. Simple inversion operations, such as image negatives and polarity inversion, also benefit significantly from LVA due to their minimal arithmetic requirements and single-load nature. In particular, audio polarity inversion is commonly used in active noise cancellation (ANC) systems, where inverting the polarity of noise allows it to destructively interfere with the original signal, thus canceling it out [41].

These applications were selected to represent a broad range of image and audio processing tasks with varying computational complexity and inherent error tolerance. Together, they provide a meaningful and practical basis for evaluating the effectiveness

of the proposed LVA methodology. By covering diverse operations, from simple inversions to more complex thresholding and blending tasks, the evaluation reflects the practical implications of LVA on both quality and performance across real-world scenarios.

Chapter 3

ML-based Load Value Predictor

3.1 Introduction

This chapter describes the Machine Learning (ML) techniques used to enhance the predictive capabilities of the Load Value Approximator (LVA) developed in this thesis. The focus is on a static, pre-trained model designed to operate efficiently during runtime without requiring continual updates or retraining. The primary role of this model is to provide load value accurate predictions based on historical patterns, enabling the system to make informed decisions in real time. Although handcrafted statistical models may be employed to predict load values, such methods do not scale effectively to larger design spaces, e.g., 16-bit data widths, due to their inability to generalize for unforeseen data patterns. In contrast, the ML-based model presented in this thesis provides a generic methodology that can be applied across different systems, offering greater adaptability and predictive robustness across diverse application domains.

In many conventional approaches, prediction mechanisms rely on hand-crafted rules or actively trained models that update dynamically based on new data. While these methods can adapt to changes over time, they also introduce a number of practical limitations. Online learning systems require additional computational resources,

complex hardware and mechanisms to deliver a good quality. Furthermore, due to the heuristic nature of the online trained system, the prediction of load values can have a substantial delay. These issues make active online learning methods unsuitable for systems that must operate under strict latency or energy constraints.

To address these challenges, this work adopts a static machine learning model—a model trained offline using a carefully curated dataset, then deployed in a fixed form at runtime. This approach offers several key advantages. First, it decouples the training process from the system's operation, allowing the use of powerful offline tools and computing resources during model development. Second, it ensures consistency and predictability in the model's behavior, which is especially valuable in embedded or real-time environments. Finally, by avoiding runtime training, the system's complexity is significantly reduced, making it easier to maintain, verify, and integrate with other components.

The decision to use a static model is further supported by the nature of the problem space. In this scenario, the input data patterns exhibit a degree of regularity and stability, making it possible to train a model that generalizes well across future scenarios. Extensive profiling and benchmarking during the design phase allowed for the creation of a representative training dataset that captures the essential characteristics needed for accurate predictions. As a result, the deployed model maintains high performance without the need for adaptation or retraining.

In the sequel, we will present the training methodology for the proposed ML-based LVA, followed by the dataset selection, then the quality assessment of the trained ML-LVA.

3.2 Training Method

Existing methods in developing LVA techniques primarily rely on the history of previously observed values, the program counter (PC), and hash functions, such as XOR-based schemes, to aid in forming accurate predictions. In [42], we explored this design space by training a machine learning model based on these principles. Specifically, the model was provided with the load memory address, the program counter, and separately computed hash values corresponding to the load memory address, the program counter, the store values, and the store addresses. Each of these hash values was supplied as a distinct feature to the model to capture a rich context for prediction. However, despite the extensive feature set, the model consistently failed to deliver satisfactory prediction quality across a range of applications. For instance, when applied to the *Canneal* benchmark from the Princeton Application Repository for Shared-Memory Computers (PARSEC) suite [43], the classification accuracy reached only 36.57%, while the regression-based trained model resulted in a root mean square error (RMSE) of 59.41. These metrics clearly indicate that the model was not able to approximate load values effectively. Furthermore, the approach incurred substantial runtime overhead, largely due to the necessity of computing and maintaining multiple separate hash values at every load and store operation, adding complexity and cost to the system.

In light of these limitations, we concluded that an effective LVA prediction technique should prioritize two fundamental goals: *i*) minimize prediction overhead, and *ii*) achieve acceptable prediction quality across a wide range of applications without excessive feature engineering. To this end, we developed a more lightweight and targeted approach, wherein the current load value is predicted solely based on the previous load value produced by the same instruction. This design choice entirely

eliminates the need for computing hash values, significantly streamlining the prediction mechanism and reducing the computational burden at runtime.

To train and evaluate the ML-based LVA we chose to use a multimedia dataset. The rationale behind this selection lies in the inherent properties of multimedia applications, where they often exhibit strong value locality, meaning that adjacent or sequential memory values tend to follow similar patterns or trends. For example, image processing tasks, such as blending, filtering, or inversion, pixel values that are spatially or temporally close, typically have correlated magnitudes. By leveraging this property, we ensure that the predictor is exposed to realistic scenarios, where value locality is prevalent. Successfully approximating load values in such a context would indicate that the predictor is capable of handling a broad class of multimedia, e.g., image and audio applications, and could potentially extend to other domains with similar locality patterns.

The methodology to generate the ML-LVA is illustrated in Figure 3.1. The overall process is divided into two primary steps conducted within a simulated environment. In Step ②, we instrument the load behavior by simulating the execution of load instructions, capturing the dynamic sequence of load values generated by each instruction instance. This sequence forms the training dataset, effectively encoding the load context in terms of past observed values. The output of Step ② is thus a structured dataset mapping historical load values to the subsequent target value.

In Step ③, we train the machine learning model using the Extra Trees algorithm [44]. Extra Trees was selected based on its favorable characteristics for our task as it offers fast training times, robustness to noisy inputs, and strong predictive accuracy when using only simple features, such as the previous load value, as input. Our previous exploration in [45] corroborated the effectiveness of Extra Trees for

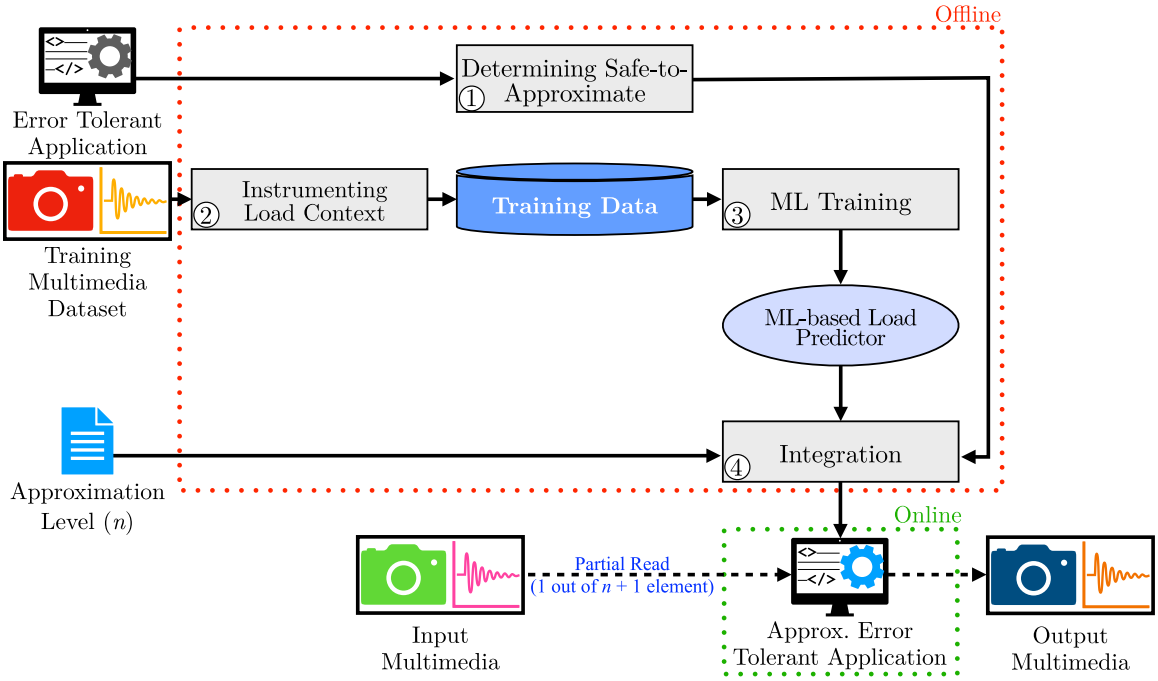


Figure 3.1: Training Method of the Proposed ML-LVA

this prediction setting. The output of Step ③ is a trained ML-LVA. In Step ④, the application is modified to predict the load value as shown in Figure 3.2. The example in the figure depicts how the prediction is applied in our proposed LVA where a square and a circle represent an exact and approximate load value, respectively. The approximation (prediction) is based on the preceding value regardless of whether it is exact or approximate. For instance, the first approximate value (A_1) predicted by the LVA is based on the exact value (E) loaded from the memory. Thereafter, the second approximate value (A_2) is based on its preceding value A_1 which is predicted/generated by the proposed LVA. This prediction sequence is repeated n times, i.e., A_1 to A_n. After predicting n approximate values, an exact value (E) is loaded again from the memory and the sequence is repeated. In this chapter, we test the quality of the proposed ML-LVA by simulating its behavior in predicting the value of load instructions, without integrating it into a full application. On the other hand, the details of determining

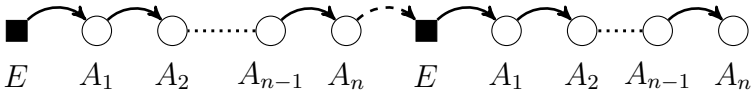


Figure 3.2: Prediction Sequence

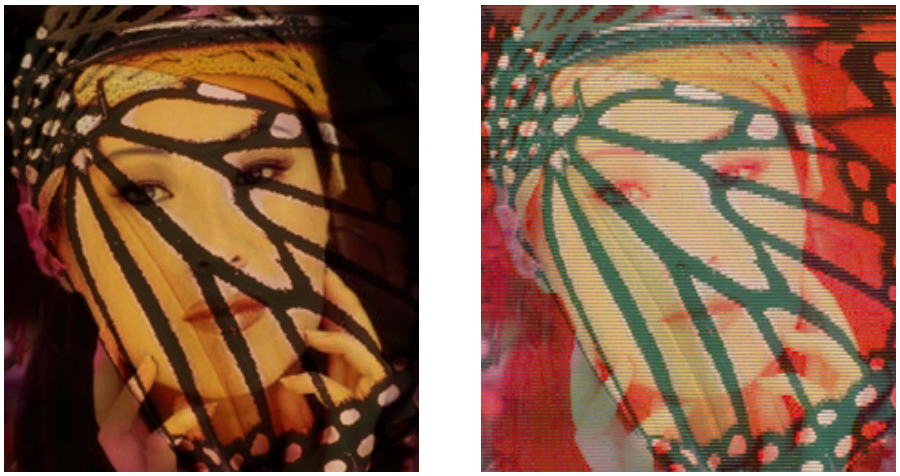
the safe-to-approximate load instructions are discussed in Chapter 4. Furthermore, specifics of the deployment of Step ④ are addressed in Chapter 4 and Chapter 5 for the software- and hardware-based implementation, respectively.

3.3 Dataset Selection and ML Training

The selection of appropriate datasets plays a critical role in training and evaluating the ML-LVA developed in this work. Given the objective to design a predictor that can effectively approximate load values in multimedia applications, it was essential to choose datasets that not only represent diverse multimedia content but also exhibit strong value locality characteristics, which is an important property for effective load value approximation. In this section, we describe the datasets used for training and evaluation, along with the rationale behind their selection and their relevance to the predictor's design goals.

3.3.1 Dataset Selection

Multimedia applications, such as image and audio processing tasks, typically demonstrate high value locality. That is, adjacent data elements, such as neighboring pixels in an image or successive audio samples in a sound file, tend to have related or correlated values. This phenomenon emerges naturally from the physical properties of the underlying content. For instance, colors in an object or continuous sound waves in an audio stream change gradually rather than abruptly. Leveraging this observation,



(a) (b)

Figure 3.3: Image Blending: (a) Exact and (b) ML-LVA Model with Poor Training

the dataset selection focused on sourcing multimedia datasets where value locality is naturally present. This ensures that the training process of the load predictor, i.e., the ML-LVA, is exposed to realistic patterns commonly encountered in practical applications and thus improving its generalization to real-world multimedia workloads.

In ML, the choice of the training dataset is a decisive factor in determining the quality of the resulting model. A dataset that lacks sufficient diversity or representative features can severely restrict the model's ability to generalize, leading to degraded predictions. Figure 3.3 illustrates this effect by comparing an exact image blending output with the result produced by an ML-LVA model trained on a poorly chosen dataset. In this example, the benchmark images known as "*Set5*" (with 5 images) and "*Set14*" (with 14 images) [46] are used. The ML-LVA is trained using *Set14* and the testing was performed using *Set5*. From this example, we can notice that the absence of representative training data results in noticeable quality loss, underscoring the importance of careful dataset selection. To ensure robust training, a combination of three publicly available datasets was selected for the image, namely, the *Flowers* [47],

the *Cars* [48], and the *Places* [49] datasets. Each dataset was chosen to introduce different types of images, thus promoting a broad exposure to diverse patterns. The *Flowers* dataset consists of 8,189 images of flowers across multiple species, captured under varying lighting conditions, backgrounds, and compositions. This dataset is particularly suitable for training purposes because it contains a rich variety of textures and colors while still maintaining consistent value locality properties at the pixel level. From this dataset, 4,094 images were used for training the ML model, referred to as the "*Training Multimedia*" as depicted in Figure 3.1, while the remaining 4,095 images were reserved for evaluation as "*Input Multimedia*" during the predictor deployment and testing. This evenly distributed dataset ensures a balanced evaluation, enabling an assessment of the predictor's ability to generalize beyond the training set.

To further diversify the "*Input Multimedia*" during evaluation, two additional datasets were incorporated. The *Cars* dataset provides 8,041 images featuring various types of vehicles captured from different angles and environments. *Car* images introduce distinct object contours, color distributions, and background variations compared to flower images, enabling the evaluation of the predictor's robustness across heterogeneous visual patterns. In addition, 16 images were included from the *Places* dataset, a large dataset composed of images depicting various indoor and outdoor scenes. Although only a small subset of the *Places* dataset was used, these images introduce further variability in textures, lighting conditions, and spatial arrangements. This added diversity challenges the predictor to maintain prediction quality when faced with less predictable or more complex visual structures. By combining these three datasets, the "*Input Multimedia*" set spans a wide range of natural imagery, enhancing the comprehensiveness of the predictor evaluation.

In addition to image datasets, an audio dataset was incorporated to introduce a

different modality of multimedia content into the training and evaluation pipeline. Audio streams, like images, exhibit strong value locality, where neighboring audio samples typically have correlated amplitude values, especially in continuous speech, background music, or ambient sounds. For this purpose, the Babylon 5 audio dataset [50] was selected. This dataset consist of 614.45 minutes distributed over 2,436 recordings from the television series "Babylon 5" encompassing a variety of audio types such as dialogues, ambient effects, and music tracks, thus providing a diverse set of audio patterns for training and evaluation. From this dataset, 1,255 audio files were used as "*Training Multimedia*" (Figure 3.1) to generate load value sequences for the model training. The remaining 1,181 audio files were used as "*Input Multimedia*" for evaluating the predictor. The audio files were split according to the file size to ensure an even distribution between the training and the application datasets. We chose this method instead of dividing by the number of audio files, as the lengths of the audio vary between the files.

3.3.2 ML Training

Separate ML-LVAs were trained for image and audio data, respectively, in order to specialize the predictor behavior for each type of multimedia content. To clarify the structure of these models, Tables 3.1 and 3.2 show the mapping of the trained ML models for image and audio models, respectively, detailing the input features, output responses, and the delta between input and output values. This mapping highlights the specific data characteristics each model processes, emphasizing the specialized nature of the predictors for image and audio workloads. From these tables, we can notice that the predicted value in an image is close in range compared to the input. On the other hand, the model trained on audio shows a larger difference in values between

Table 3.1: Mapping of the Trained ML Model Developed using Images

F§	R‡	Δ*	F§	R‡	Δ*	F§	R‡	Δ*	F§	R‡	Δ*
0	1	1	64	64	0	128	128	0	192	191	-1
1	2	1	65	65	0	129	129	0	193	192	-1
2	3	1	66	66	0	130	130	0	194	193	-1
3	4	1	67	67	0	131	131	0	195	194	-1
4	5	1	68	68	0	132	132	0	196	195	-1
5	5	0	69	69	0	133	133	0	197	196	-1
6	6	0	70	70	0	134	134	0	198	197	-1
7	7	0	71	71	0	135	135	0	199	198	-1
8	8	0	72	72	0	136	136	0	200	199	-1
9	9	0	73	73	0	137	137	0	201	200	-1
10	10	0	74	74	0	138	138	0	202	201	-1
11	11	0	75	75	0	139	139	0	203	202	-1
12	12	0	76	76	0	140	140	0	204	203	-1
13	13	0	77	77	0	141	141	0	205	204	-1
14	14	0	78	78	0	142	142	0	206	205	-1
15	15	0	79	79	0	143	143	0	207	206	-1
16	16	0	80	80	0	144	144	0	208	207	-1
17	17	0	81	81	0	145	145	0	209	208	-1
18	18	0	82	82	0	146	146	0	210	209	-1
19	19	0	83	83	0	147	147	0	211	210	-1
20	20	0	84	84	0	148	148	0	212	211	-1
21	21	0	85	85	0	149	149	0	213	212	-1
22	22	0	86	86	0	150	150	0	214	213	-1
23	23	0	87	87	0	151	150	-1	215	214	-1
24	24	0	88	88	0	152	151	-1	216	215	-1
25	25	0	89	89	0	153	152	-1	217	216	-1
26	26	0	90	90	0	154	153	-1	218	217	-1
27	27	0	91	91	0	155	154	-1	219	218	-1
28	28	0	92	92	0	156	155	-1	220	219	-1
29	29	0	93	93	0	157	156	-1	221	220	-1
30	30	0	94	94	0	158	157	-1	222	221	-1
31	31	0	95	95	0	159	158	-1	223	222	-1
32	32	0	96	96	0	160	159	-1	224	223	-1
33	33	0	97	97	0	161	160	-1	225	224	-1
34	34	0	98	98	0	162	161	-1	226	225	-1
35	35	0	99	99	0	163	162	-1	227	226	-1
36	36	0	100	100	0	164	163	-1	228	227	-1
37	37	0	101	101	0	165	164	-1	229	228	-1
38	38	0	102	102	0	166	165	-1	230	229	-1
39	39	0	103	103	0	167	166	-1	231	230	-1
40	40	0	104	104	0	168	167	-1	232	231	-1
41	41	0	105	105	0	169	168	-1	233	232	-1
42	42	0	106	106	0	170	169	-1	234	233	-1
43	43	0	107	107	0	171	170	-1	235	234	-1
44	44	0	108	108	0	172	171	-1	236	235	-1
45	45	0	109	109	0	173	172	-1	237	236	-1
46	46	0	110	110	0	174	173	-1	238	237	-1
47	47	0	111	111	0	175	174	-1	239	238	-1
48	48	0	112	112	0	176	175	-1	240	239	-1
49	49	0	113	113	0	177	176	-1	241	240	-1
50	50	0	114	114	0	178	177	-1	242	241	-1
51	51	0	115	115	0	179	178	-1	243	242	-1
52	52	0	116	116	0	180	179	-1	244	243	-1
53	53	0	117	117	0	181	180	-1	245	244	-1
54	54	0	118	118	0	182	181	-1	246	245	-1
55	55	0	119	119	0	183	182	-1	247	246	-1
56	56	0	120	120	0	184	183	-1	248	247	-1
57	57	0	121	121	0	185	184	-1	249	248	-1
58	58	0	122	122	0	186	185	-1	250	249	-1
59	59	0	123	123	0	187	186	-1	251	250	-1
60	60	0	124	124	0	188	187	-1	252	251	-1
61	61	0	125	125	0	189	188	-1	253	252	-1
62	62	0	126	126	0	190	189	-1	254	253	-1
63	63	0	127	127	0	191	190	-1	255	253	-2

§Input **F**eature

‡Output **R**esponse

*Delta between Input and Output

Table 3.2: Mapping of the Trained ML Model Developed using Audio Data

F§	R‡	Δ*	F§	R‡	Δ*	F§	R‡	Δ*	F§	R‡	Δ*
0	0	0	64	73	9	128	128	0	192	183	-9
1	2	1	65	73	8	129	129	0	193	184	-9
2	20	18	66	75	9	130	129	-1	194	184	-10
3	26	23	67	75	8	131	130	-1	195	186	-9
4	21	17	68	76	8	132	131	-1	196	185	-11
5	24	19	69	77	8	133	132	-1	197	188	-9
6	22	16	70	78	8	134	133	-1	198	187	-11
7	25	18	71	79	8	135	134	-1	199	189	-10
8	23	15	72	80	8	136	135	-1	200	189	-11
9	27	18	73	80	7	137	136	-1	201	191	-10
10	25	15	74	81	7	138	136	-2	202	191	-11
11	29	18	75	82	7	139	137	-2	203	192	-11
12	27	15	76	83	7	140	138	-2	204	194	-10
13	29	16	77	84	7	141	139	-2	205	194	-11
14	28	14	78	85	7	142	140	-2	206	194	-12
15	33	18	79	85	6	143	141	-2	207	196	-11
16	30	14	80	86	6	144	142	-2	208	196	-12
17	33	16	81	87	6	145	143	-2	209	198	-11
18	32	14	82	88	6	146	143	-3	210	198	-12
19	36	17	83	89	6	147	144	-3	211	199	-12
20	34	14	84	90	6	148	145	-3	212	200	-12
21	38	17	85	91	6	149	146	-3	213	202	-11
22	36	14	86	92	6	150	147	-3	214	202	-12
23	39	16	87	92	5	151	148	-3	215	203	-12
24	37	13	88	93	5	152	149	-3	216	202	-14
25	41	16	89	94	5	153	149	-4	217	206	-11
26	38	12	90	95	5	154	150	-4	218	205	-13
27	42	15	91	96	5	155	151	-4	219	207	-12
28	42	14	92	97	5	156	152	-4	220	207	-13
29	41	12	93	98	5	157	153	-4	221	208	-13
30	44	14	94	98	4	158	154	-4	222	209	-13
31	44	13	95	99	4	159	154	-5	223	210	-13
32	46	14	96	100	4	160	155	-5	224	210	-14
33	45	12	97	101	4	161	156	-5	225	212	-13
34	47	13	98	102	4	162	157	-5	226	212	-14
35	49	14	99	103	4	163	158	-5	227	214	-13
36	49	13	100	104	4	164	159	-5	228	214	-14
37	49	12	101	104	3	165	160	-5	229	215	-14
38	51	13	102	105	3	166	161	-5	230	217	-13
39	50	11	103	106	3	167	161	-6	231	215	-16
40	53	13	104	107	3	168	162	-6	232	219	-13
41	53	12	105	108	3	169	163	-6	233	217	-16
42	54	12	106	109	3	170	164	-6	234	220	-14
43	55	12	107	109	2	171	165	-6	235	218	-17
44	56	12	108	110	2	172	166	-6	236	222	-14
45	57	12	109	111	2	173	167	-6	237	221	-16
46	58	12	110	112	2	174	167	-7	238	224	-14
47	58	11	111	113	2	175	168	-7	239	222	-17
48	59	11	112	114	2	176	169	-7	240	226	-14
49	59	10	113	115	2	177	170	-7	241	224	-17
50	61	11	114	116	2	178	170	-8	242	229	-13
51	61	10	115	116	1	179	172	-7	243	226	-17
52	62	10	116	117	1	180	172	-8	244	230	-14
53	64	11	117	118	1	181	173	-8	245	230	-15
54	65	11	118	119	1	182	174	-8	246	233	-13
55	65	10	119	120	1	183	175	-8	247	234	-13
56	66	10	120	121	1	184	176	-8	248	236	-12
57	67	10	121	122	1	185	177	-8	249	235	-14
58	68	10	122	123	1	186	177	-9	250	238	-12
59	68	9	123	123	0	187	179	-8	251	238	-13
60	70	10	124	124	0	188	179	-9	252	238	-14
61	70	9	125	125	0	189	181	-8	253	231	-22
62	71	9	126	126	0	190	181	-9	254	238	-16
63	72	9	127	127	0	191	182	-9	255	244	-11

§Input **F**eature
‡Output **R**esponse
*Delta between Input and Output

the input feature and output response where the largest difference is 23. By training distinct models, each predictor is exposed to the unique data characteristics within its corresponding domain. For the image predictor, the training data exhibits trends, such as smooth color gradients, and repetitive textures, all of which demonstrate strong spatial value locality. For the audio predictor, the training data contains features such as gradual amplitude changes, periodic waveforms, and occasional sharp transitions, reflecting strong temporal locality. Thus, all selected datasets maintain the intrinsic value locality necessary for effective load value approximation. By maintaining separate predictors, the evaluation can more precisely assess the model's ability to learn and exploit domain-specific locality patterns. Furthermore, the usage of dedicated predictors allows for a more focused optimization for each modality, ensuring that the design and performance tuning of the predictor align with the distinct requirements of image and audio processing tasks. The datasets used reflect realistic multimedia application scenarios, where processing tasks often involve either image or audio content independently. Consequently, the evaluation of these separately trained predictors provides meaningful insights into the practical deployment of load value approximation techniques in diverse multimedia workloads.

In summary, the dataset selection strategy was carefully designed to align with the goals of minimizing prediction overhead while maintaining high-quality approximations across diverse multimedia applications. By utilizing a mix of image and audio datasets, partitioning the datasets systematically for training and evaluation, and focusing on sources with strong value locality, the ML-LVA is trained and evaluated under realistic and challenging conditions. This strategy forms the foundation for the subsequent quality analysis discussed in the rest of this chapter and the deployment of the ML-LVA in software or hardware in the subsequent chapters.

3.4 Quality Assessment

In this section, we simulate the output quality when using ML-LVA on six multimedia applications, namely, multiplication-based image blending [32], multiplication-based audio blending known as Ring Modulation (RM) [33], audio binarization known as infinite clipping [35], image binarization known as image thresholding [34], polarity inversion of audio [37], and image inversion known as image negatives [36]. This section is divided into subsections for each multimedia application, where we analyze the quality of the results in terms of common quality metrics, including PSNR, NMAE, NRMSE, accuracy, and precision. Additionally, we discuss how approximation affects each application differently, considering their specific nature and requirements.

3.4.1 Image Processing

The first set of experiments focuses on image processing applications, which are fundamental in many areas, such as photography, medical imaging, and computer vision. These applications often involve operations that manipulate pixel values, such as blending, inversion, and binarization. In the context of approximation, the goal is to test the impact of reduced computational load through LVA on image quality.

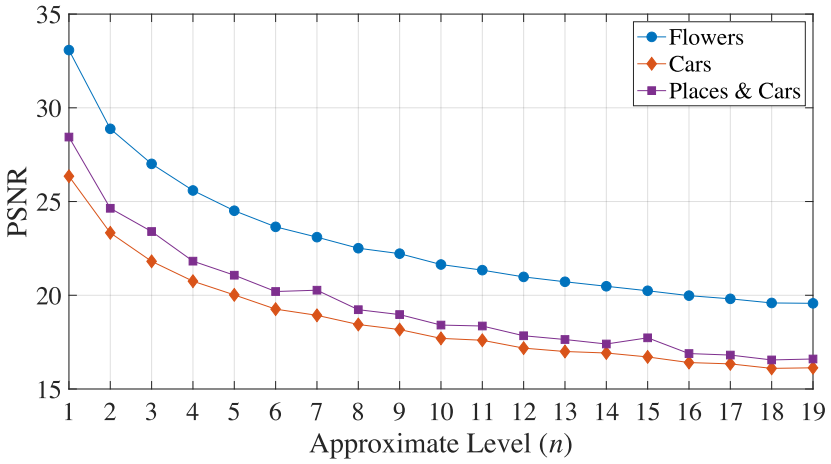
Each image processing experiment is evaluated across a range of approximation levels, from $n = 1$ to $n = 19$, where n corresponds to an approximation percentage of load value ranging from 50% to 95%. The experiments involve different types of image transformations, each of which has different computational requirements and quality impacts.

3.4.1.1 Image Blending

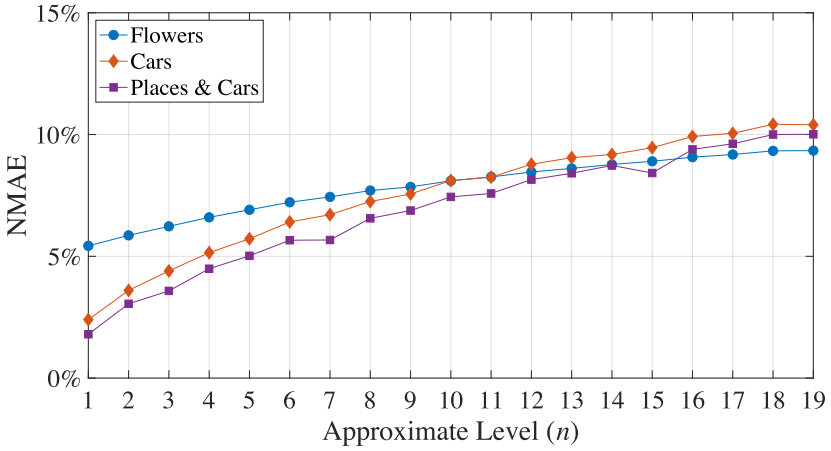
The quality metrics observed when experimenting with the multiplication based image blending are shown in Figure 3.4. From Figure 3.4(a), we can notice that the PSNR

ranges from 16.13 dB to 33.08 dB where for a smaller n, i.e., less approximation, a higher PSNR was achieved. Researches evaluated human perception of quality in relation to PSNR values. They categorized the quality as *Excellent* when the PSNR exceeds 37 dB, *Good* between 31 dB and 37 dB, *Fair* from 25 dB to 31 dB, and *Poor* when PSNR falls between 20 dB and 25 dB. Any value below 20 dB is classified as *Bad* quality [51, 52]. Based on this classification, the PSNR results obtained across the three image datasets remain within acceptable quality levels, particularly within the *Fair* to *Good* range, for the initial approximation levels. As the approximation level increases from 1 to 19, the PSNR values show a gradual decline. Nevertheless, when the ML is tested on the *Flowers* dataset, the PSNR delivered an acceptable quality even when the approximate level reached 15. On the other hand, when blending *Cars* or *Cars* with *Places*, up to an approximate level of 4 or 5, i.e., 80% approximation, the PSNR consistently stays above 20 dB, indicating that the visual quality remains within acceptable bounds. This suggests that modest levels of approximation can be safely applied without significantly degrading perceptual quality. Subsequently, as n increases, the PSNR decreases, but still maintains acceptable quality for most use cases. The NMAE and NRMSE follow a similar trend, with increasing errors at higher approximation levels. From Figure 3.4(b), we can notice that the NMAE was as low as 1.8% and slightly surpassed 10% in few cases only. Furthermore, from Figure 3.4(c), we can notice that the NRMSE ranged from 4% to 15.6%.

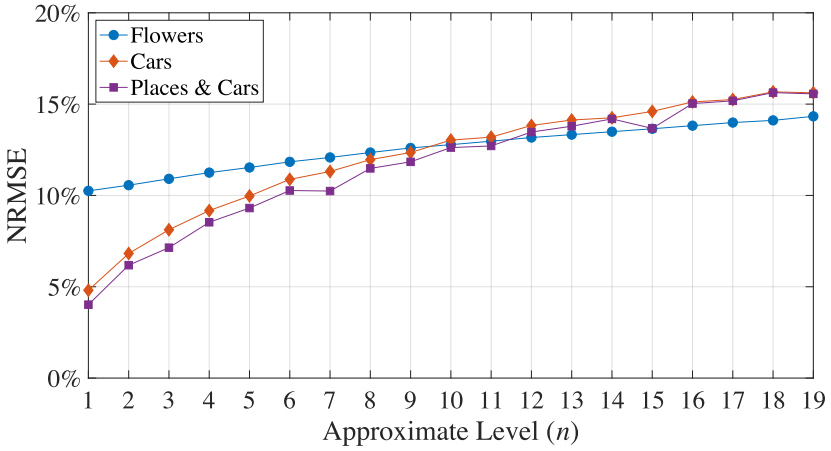
When comparing the quality of the blending images from the *Cars* dataset with the blending of the one with the *Flowers*, we can notice the blending quality slightly differs. From Figure 3.4, we notice that the quality when applied to images from the *Flowers* dataset, the measured metrics had a linear trend while the one with the *Cars* had a logarithmic trend. Moreover, we can notice that for a lower approximate level,



(a)



(b)

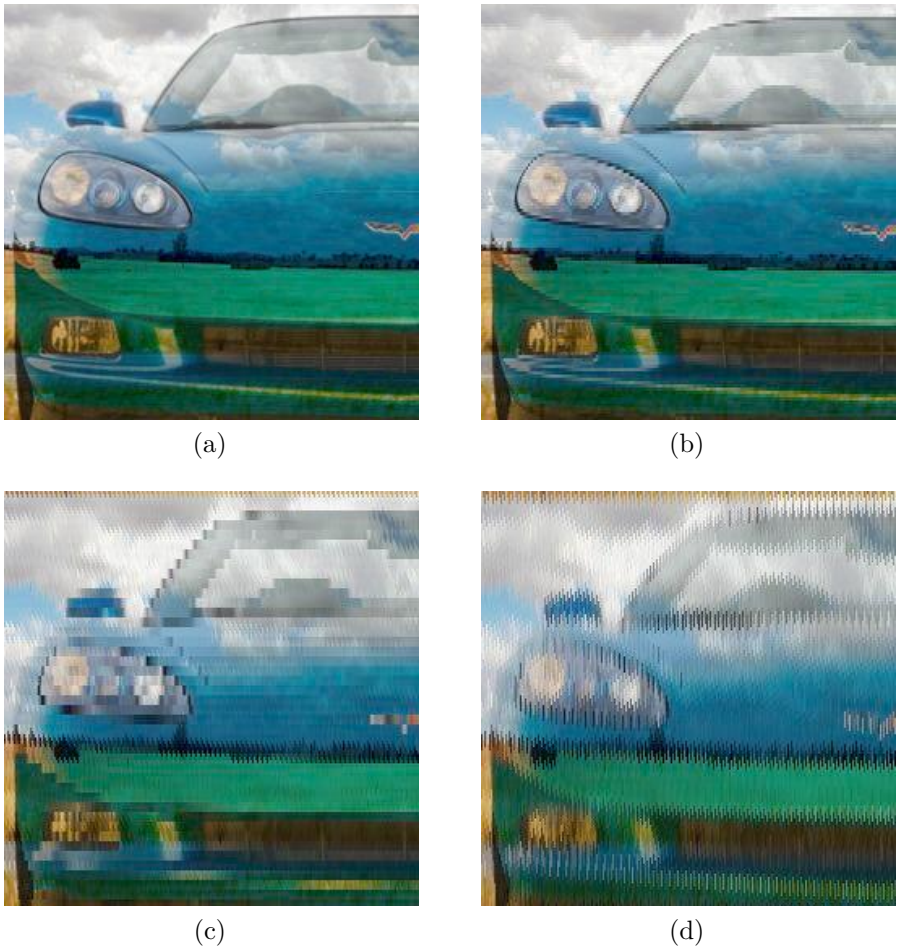


(c)

Figure 3.4: (a) PSNR, (b) NMAE, and (c) NRMSE for Various Image Blending Sets

i.e., less approximation, the quality of blending images from the *Cars* dataset was better compared to the one from the *Flowers* dataset. This can be attributed to the distinct nature of the images in the *Cars* dataset, which might have different contrast and texture compared to the *Flowers* dataset. Additionally, since the ML-LVA was trained using a portion of the *Flowers* dataset while achieving on average a better quality with the *Cars* dataset, this demonstrates the extensibility of the trained model to any image, even if they do not resemble the one used in the training phase.

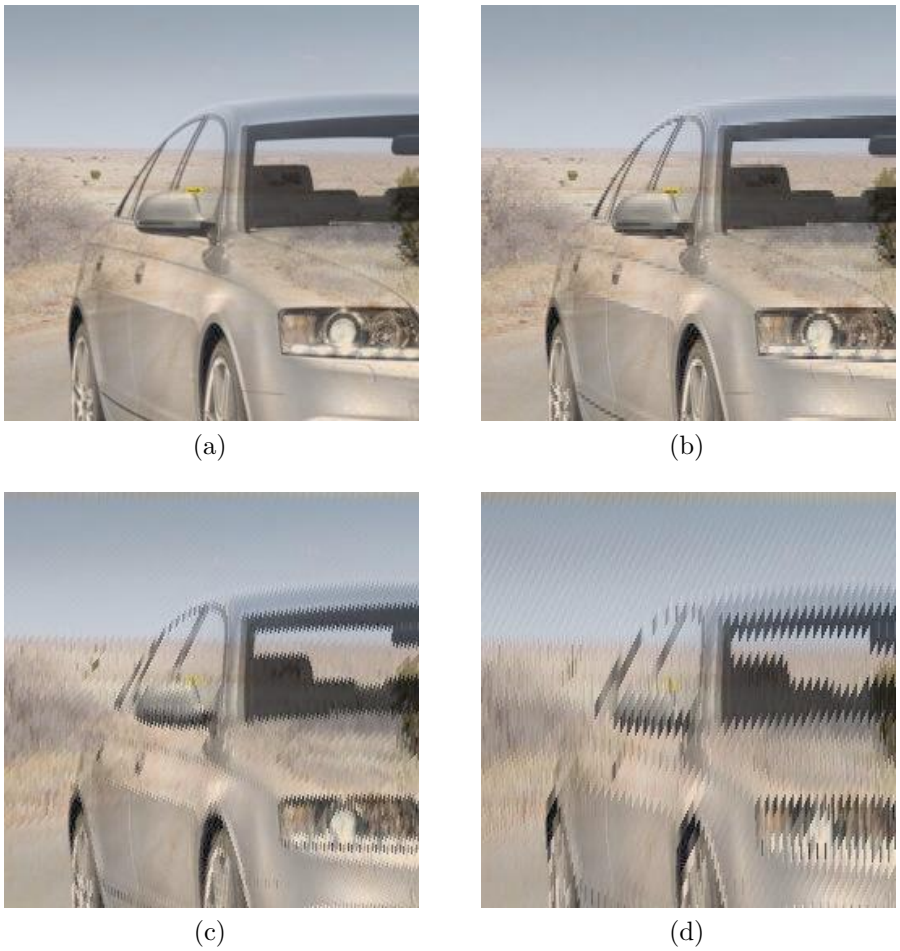
To further investigate the quality of the trained model, we test 16 images from



(a) (b)

(c) (d)

Figure 3.5: Image Blending Example 1: (a) Exact, (b) 50% Approximation (n=1), (c) 80% Approximation (n=5), and (d) 90% Approximation (n=9)



(a) (b)

(c) (d)

Figure 3.6: Image Blending Example 2: (a) Exact, (b) 50% Approximation (n=1), (c) 80% Approximation (n=5), and (d) 90% Approximation (n=9)

the *Places* dataset and 129 images from the *Cars* dataset, where the quality is shown in Figure 3.4, i.e., Places & Cars. The results demonstrate that even when mixing contexts of images, the quality was similar to the one of blending *Cars* only.

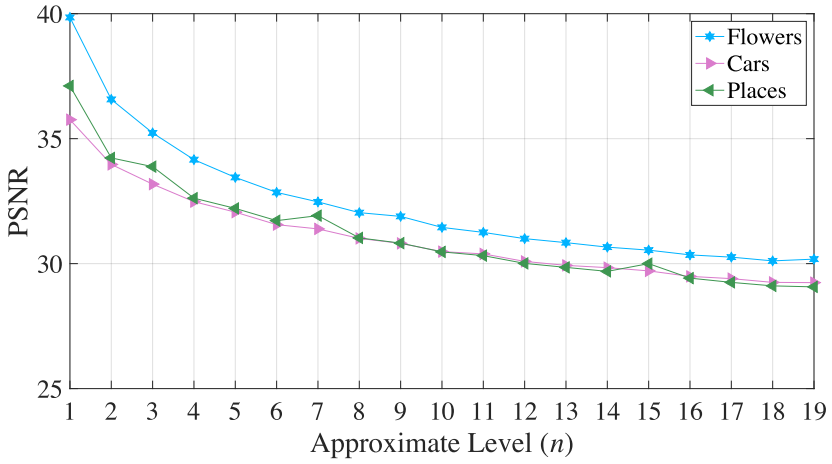
Analyzing the quality objectively, we can notice from Figures 3.5 and 3.6 that for various approximate levels, the pixels are in general predicted accurately, i.e., the color of the pixels are predicted accurately. Additionally, we can notice that for a higher approximation, sharp edges that consist of significant color changes could be less accurately approximated, whereas less sharp edges and shapes are predicted

accurately. Finally, we notice that for a 50% approximation, the quality loss is barely noticeable. At 80% approximation, the reduction in quality can still be deemed acceptable in scenarios where systems operate under limited memory bandwidth. In such cases, the additional performance gains achieved through approximation outweigh the minor perceptual loss, providing a favorable trade-off between efficiency and output fidelity. However, at 90% approximation the quality loss becomes more visible yet still consumable if the performance gain is the ultimate goal.

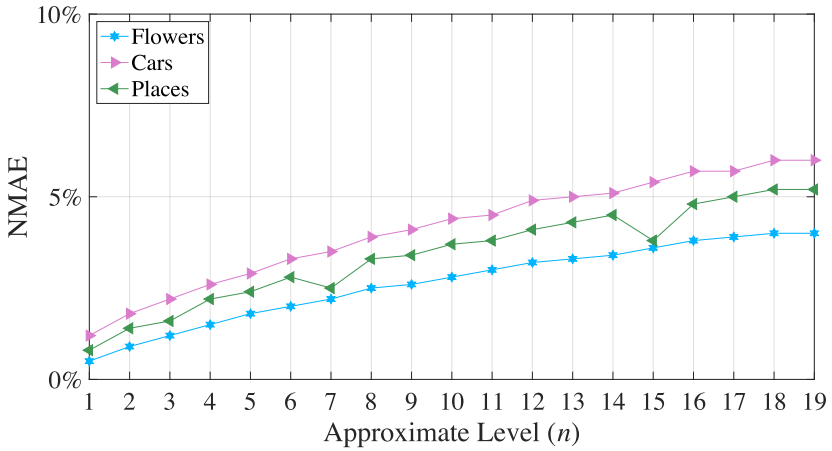
3.4.1.2 Image Inversion

The quality analysis of image inversion is shown in Figure 3.7. Compared to image blending, image inversion is more resilient to approximation, as demonstrated by the consistently higher PSNR values across all approximation levels where the quality is in the *Fair* to *Excellent* ranges. Moreover, we can notice that the inversion of images from the *Flowers* dataset attained a better quality compared to the other datasets. Nonetheless, the resulting quality is comparable among the various sets.

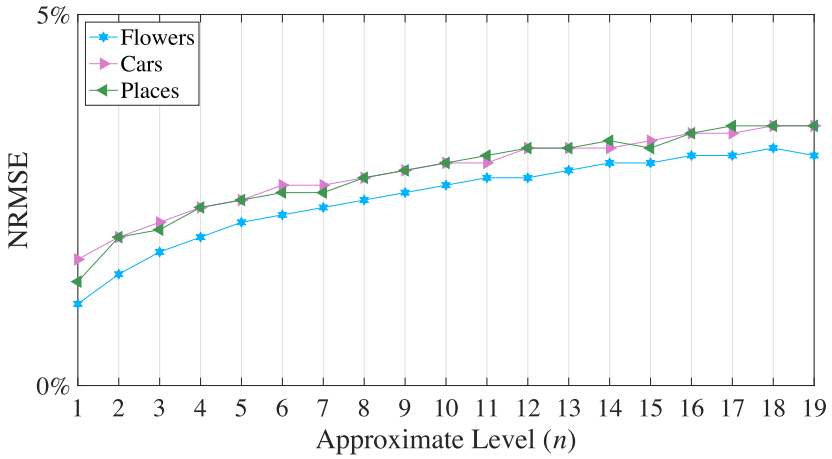
From Figure 3.7(a), PSNR values range from 29.07 dB to 39.85 dB. This suggests that image inversion, being a less complex operation, is less sensitive to approximation and retains a high quality even with a large reduction in computational load. On the other hand, the NMAE and NRMSE values for image inversion are also lower than those for image blending, as shown in Figures 3.7(b) and 3.7(c), respectively. The NMAE ranged from 0.5 to 6.0 and the NRMSE from 1.1 to 3.5, where a lower approximation level achieved a lower error. Subsequently, compared to image blending, the image inversion delivered superior results in terms of PSNR, NMAE and NRMSE. Finally, when analyzing the quality subjectively, we can notice from Figure 3.8 that similar to image blending, at 50% the quality loss is barely noticeable and at 80%



(a)

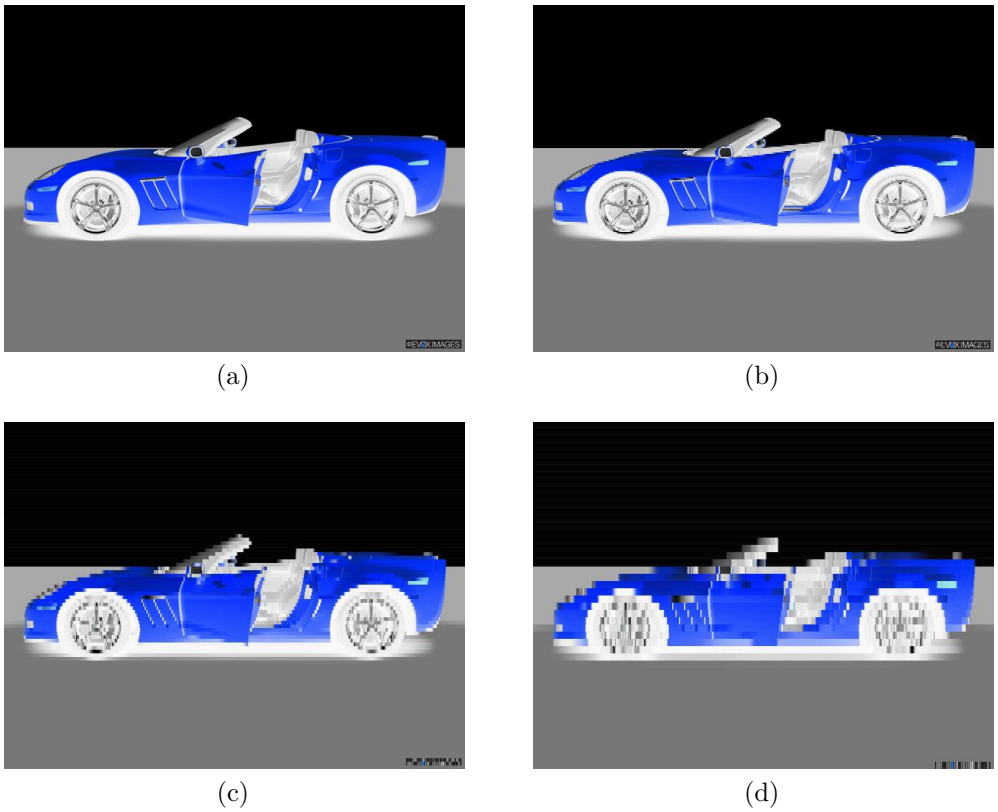


(b)



(c)

Figure 3.7: (a) PSNR, (b) NMAE, and (c) NRMSE for Various Image Inversion Sets



(a) (b) (c) (d)

Figure 3.8: Image Inversion Example: (a) Exact, (b) 50% Approximation (n=1), (c) 80% Approximation (n=4), and (d) 90% Approximation (n=9)

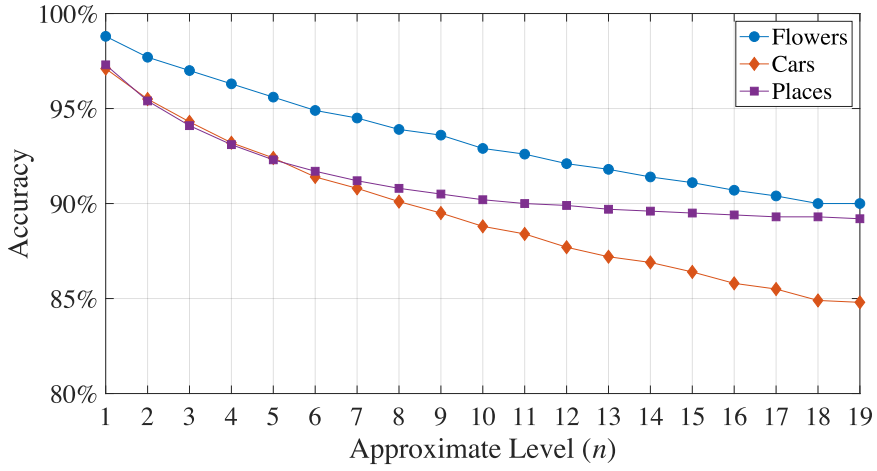
the quality can be considered acceptable. For a 90% approximation the quality is drastically reduced, however the output is still consumable.

3.4.1.3 Image Binarization

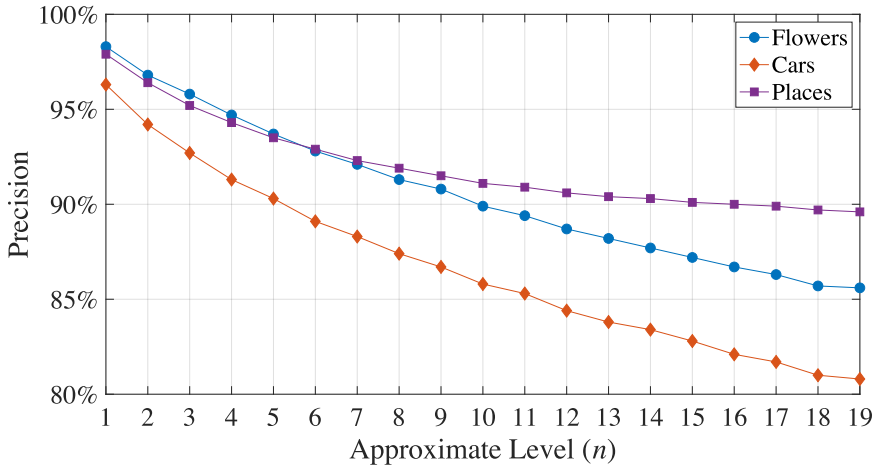
In the context of image binarization, where the output is a binary value consisting solely of two possible pixel values, the usage error-rate based error metrics, i.e., accuracy and precision, as evaluation metrics is more appropriate than magnitude-based error metrics such as PSNR, NMAE, or NRMSE. The accuracy and precision of the binarized images are shown in Figure 3.9. The accuracy drops to 84.8% and precision drops to 80.8% for an approximation of 95%, i.e., $n = 19$, as shown in Figures 3.9(a) and 3.9(b), respectively. However, at the 50% approximation, i.e., $n = 1$, both accuracy and

precision are high at 98.8% and 98.3%, respectively. Subsequently, with a quality loss of less than 2% with a 50% approximation, the trade-off in quality is deemed acceptable. Furthermore, since for a 95% approximation, i.e., $n = 19$, the quality loss was less than 20%, which can be also deemed acceptable.

From the example of image binarization shown in Figure 3.10, we can notice a similar trend in quality to the previous applications. Finally, we can suggest from all the examples presented throughout the various applications, that a higher approximation level, e.g., more than 80% approximation, can be used if the performance is the

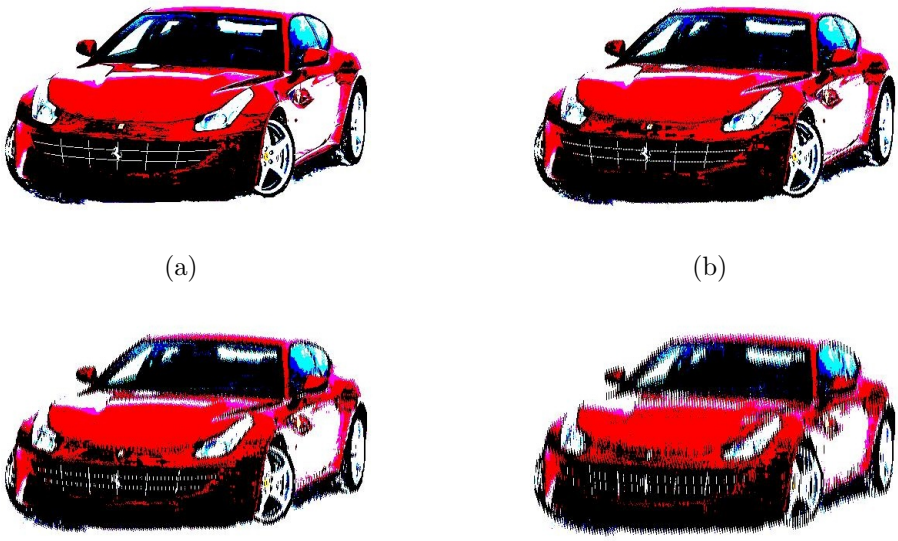


(a)



(b)

Figure 3.9: (a) Accuracy, and (b) Precision for Various Image Binarization Sets



(a) (b)

(c) (d)

Figure 3.10: Image Binarization Example: (a) Exact, (b) 50% Approximation (n=1), (c) 80% Approximation (n=4), and (d) 90% Approximation (n=9)

ultimate goal, given that at high approximation, the human perception can easily identify the quality loss.

3.4.2 Audio Processing

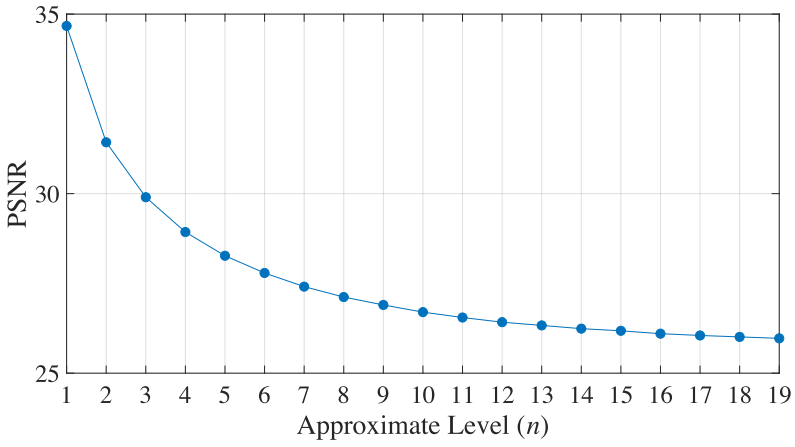
The second set of experiments involves audio processing applications, which are integral in fields like music production, speech recognition and headphones design. Audio processing typically requires operations that manipulate sound waves, such as blending, inversion, and binarization. The impact of deploying the proposed ML-LVA on these operations is investigated in this section, using the same range of n values as in the image processing experiments.

3.4.2.1 Audio Blending

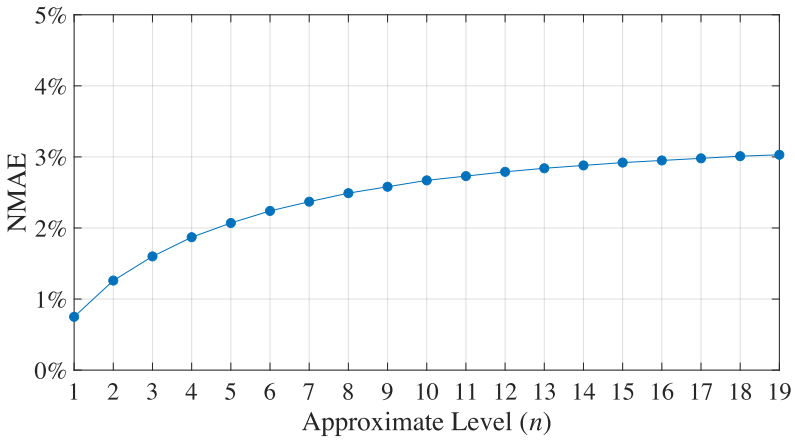
In this experiment, a Ring Modulation (RM) was employed to blend two audio signals under varying levels of approximation. The results are presented in Figure 3.11. The corresponding PSNR values range from 34.67 dB at the lowest approximation level ($n = 1$), i.e., 50% of load values are approximated, to 25.97 dB at the highest level ($n = 19$), where approximately 95% of the load values were predicted, as shown in Figure 3.11(a). Notably, even under substantial approximation, e.g., 95% approximation of load values, the PSNR consistently remains above the 25 dB threshold, which is typically considered indicative of acceptable quality [53]. This demonstrates that similar to image blending, the audio blending has a significant tolerance to high levels of approximation without significant perceptual degradation when using the proposed ML-LVA.

The change in NMAE and NRMSE for the various approximation levels is depicted in Figures 3.11(b) and 3.11(c), respectively. Both metrics exhibit a gradual increase with rising approximation, with NMAE varying from 0.7% to 3.0% and NRMSE increasing from 1.6% to 4.6%. Despite this growth, since the error values remain less than 5%, this further supports the observation that the perceived audio quality remains largely unaffected even at elevated approximation levels.

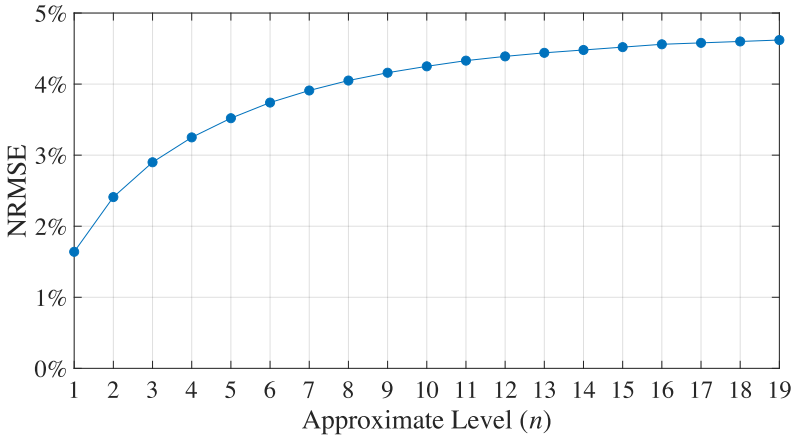
When compared to image blending, audio blending appears marginally more resilient to approximation. This may be due to the inherent characteristics of audio signals, which exhibit greater nuance and better value locality, enabling more accurate prediction with the trained ML-LVA. The temporal continuity and smoother transitions in audio data also help maintain prediction accuracy, reducing the impact of using the ML-LVA and allowing for improved performance with minimal degradation.



(a)



(b)



(c)

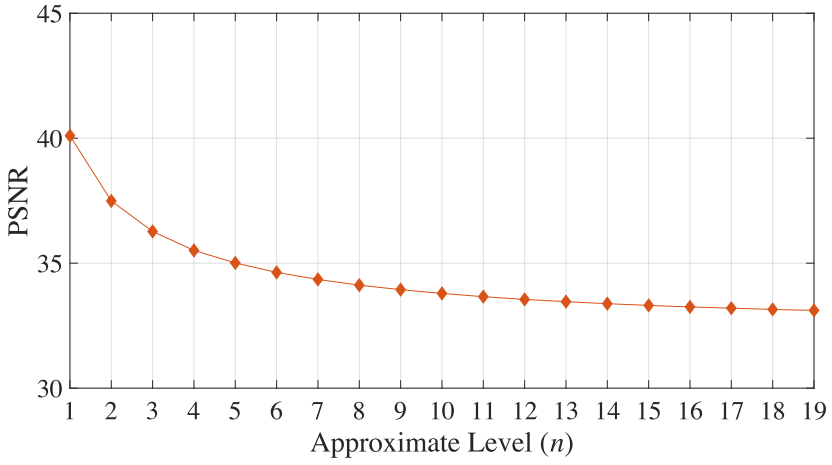
Figure 3.11: (a) PSNR, (b) NMAE, and (c) NRMSE for Various Approximate Levels of Audio Blending

3.4.2.2 Audio Inversion

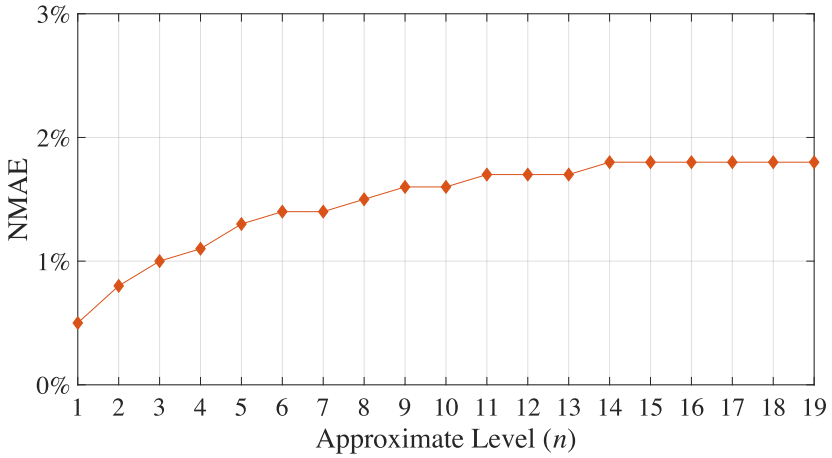
The results of applying the proposed ML-LVA to audio inversion are presented in Figure 3.12. As the approximate level (n) increases, a gradual reduction in PSNR and an increase in NMAE and NRMSE is observed. Nevertheless, the performance remains within acceptable bounds across all the tested approximate levels. As shown in Figure 3.12(a), the PSNR declines from 40.10 dB at minimal approximation, i.e., $n = 1$, to 33.11 dB at the highest approximation level, i.e., $n = 19$, reflecting a controlled deterioration of signal fidelity under approximation. Importantly, the PSNR consistently remains well above the 31 dB threshold (*Good* quality), indicating that the core structure of the binarized signal is preserved even under a significant approximate level, e.g., 95% approximation.

On the other hand, the NMAE and NRMSE, exhibit a modest increase with higher approximation levels. The NMAE rises from 0.5% to 1.8%, while the NRMSE varies from 1.1% to 2.3% as shown in Figures 3.12(b) and 3.12(c), respectively. These increments are gradual and relatively minor, further confirming that the approximation introduces limited distortion into the binarized audio representation. Notably, with 90% approximation of load values, i.e., $n = 19$, both NMAE and NRMSE remain low, i.e., less than 3%, highlighting the robustness of the inversion process against intermediate levels of approximation.

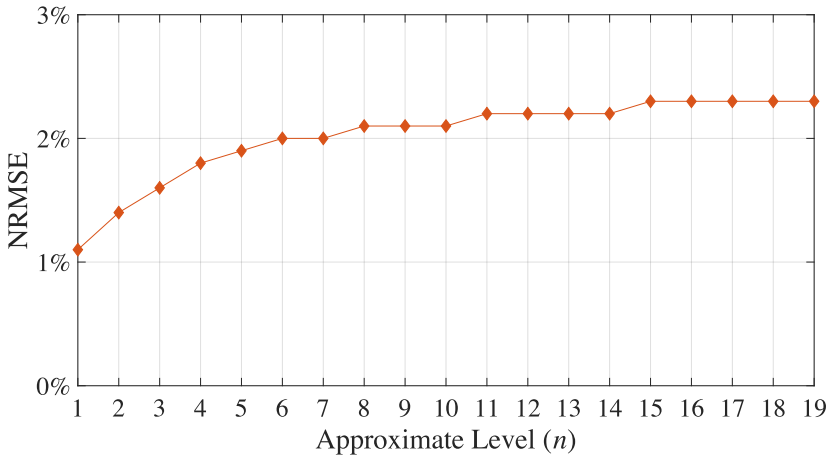
Overall, the results of this experiment demonstrate that the ML-LVA can effectively predict the load values in the audio inversion task. While increasing the approximation level introduces a slight increase in approximation errors, the overall quality of the inverted audio signal remains high across all levels. These findings validate that the proposed ML-LVA is a viable technique for reducing the resources needed when performing an audio inversion, ensuring minimal degradation in signal quality even



(a)



(b)



(c)

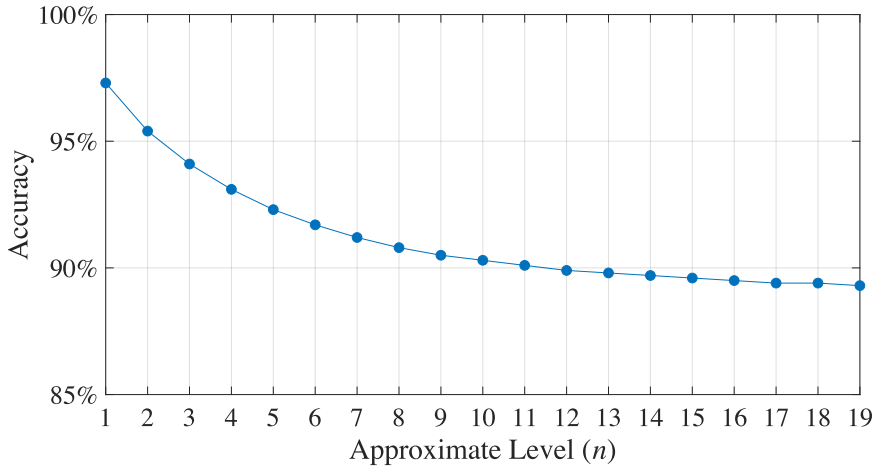
Figure 3.12: (a) PSNR, (b) NMAE, and (c) NRMSE for Various Approximate Levels of Audio Inversion

at higher approximation levels. Thus, the experiment successfully highlights the practical applicability of ML-LVA in audio processing tasks requiring accurate load value prediction.

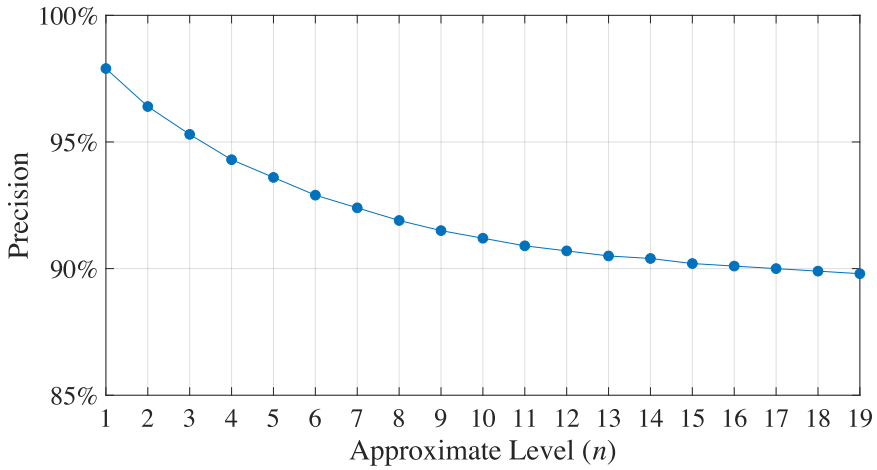
3.4.2.3 Audio Binarization

For audio binarization, where the output signal is reduced to just two discrete amplitude levels, it is more meaningful to assess performance using error-rate based metrics such as accuracy and precision rather than relying on magnitude-based error measures used in the previous audio applications, i.e., PSNR, NMAE, or NRMSE. The results of the audio binarization are depicted in Figure 3.13. These results reflect the performance of the binarization process across varying levels of approximation, from $n = 1$ to $n = 19$. As the approximation level increases, both accuracy and precision gradually decrease, which is expected due to the introduction of approximation errors. For the least approximation level ($n = 1$), i.e., 50% approximation, the accuracy is 97.3%, and the precision is 97.9%, indicating a high level of fidelity between the original and binarized signals. As the approximation level increases, we observe a steady decline in both metrics. At $n = 2$, the accuracy diminishes to 95.4%, and precision decreases to 96.4%, indicating that even at elevated approximation level, i.e., 66.67% approximation, a noticeable reduction in quality is observed. Nonetheless, the accuracy and precision indicate that the error in the output is still tolerable. As the approximation level continues to increase, the accuracy reaches 89.3% and the precision 89.8% for $n = 19$.

Despite this decrease in both accuracy and precision as the approximation level approaches its maximum, the performance remains within an acceptable range. The accuracy and precision metrics exhibit a clear, consistent trend of degradation as the approximation level increases, but the values do not drop to levels that would



(a)



(b)

Figure 3.13: (a) Accuracy, and (b) Precision for Various Approximate Levels Audio Binarization

undermine the overall quality of the binarized output. This behavior suggests that the audio binarization method remains robust and can tolerate higher levels of approximation without significant loss in quality and hence suitable for usage with the proposed ML-LVA.

3.4.3 Comparison with Related Work

We compare the proposed ML-LVA to the state-of-the-art LVA proposed in [19, 24]. Since these related work use different error metrics, we will compare to each of

these work separately. For instance, the authors of [19] used the NMAE as an error metric when analyzing the effect of their LVA when applied to the PARSEC benchmark [43]. We compare the average prediction quality of the proposed ML-LVA across all applications and datasets evaluated in the previous section with the average quality reported for the applications studied in [19]. The comparison is summarized in Table 3.3. We can notice that the NMAE of the work in [19] is more than double for the various approximate levels. Since the proposed LVA delivers a better quality than the LVA proposed in [19] for the various approximate levels, we can conclude that the LVA we propose is superior.

Table 3.3: Comparison of NMAE of the Proposed ML-LVA with [19]

Approximate Level (n)	**LVA** [19]	**Proposed ML-LVA**
1	5.81%	1.98%
3	7.25%	3.17%
5	8.97%	3.98%
9	11.06%	5.01%
17	13.78%	6.30%

To further establish a common reference, we evaluate the quality of the proposed ML-LVA using the same input data as the PARSEC multimedia applications, specifically the *simlarge* dataset [43]. Subsequently, the *simlarge* input data for three PARSEC multimedia applications, namely BodyTrack, Ferret and x264, are used as input to the applications chosen in this thesis, i.e., Blending, Binarization and Inversion of images. BodyTrack and Ferret operate on image inputs, whereas x264 processes video streams. Since the applications used in this thesis do not directly handle video content, we extracted individual frames from the x264 input and applied them as image data to our workloads. This adaptation ensures that our evaluation remains consistent with the datasets used in related work while staying aligned with

Table 3.4: Comparison of NMAE of the Proposed ML-LVA with [19] when using the *simlarge* Dataset

Approximate Level (n)	**LVA** [19]	**Proposed ML-LVA**
1	10.52%	1.81%
3	10.90%	3.32%
5	11.87%	4.42%
9	12.64%	6.02%
17	13.79%	7.75%

the scope of the chosen applications. This allows for a direct comparison under identical data conditions. Table 3.4 compares the average quality obtained for the three PARSEC multimedia applications evaluated in [19] with the prediction quality achieved by the proposed ML-LVA when applied to the *simlarge* input using the chosen multimedia applications in this thesis. From Tables 3.3 and 3.4 it becomes clear that the prediction accuracy of the proposed ML-LVA remains stable across datasets, with no significant fluctuations between different inputs. This deterministic behavior contrasts with the variability reported in [19], where prediction quality could vary considerably from one application set to another. Thus, beyond providing higher accuracy, our proposed ML-LVA ensures predictable performance regardless of the input dataset.

The authors of [24] proposed the Rollback Free Value Predictor (RFVP) and used the NRMSE as an error metric when evaluating their method on selected applications from the Rodina [54], Mars [55] and Nvidia SDK [56] benchmarks. The comparison with the proposed LVA is summarized in Table 3.5. Even though the authors of [24] targeted GPU architectures with different approximate levels, we can fairly compare the quality results of their proposed model and the LVA we propose. In fact, the quality will only vary based on the percentage of instances that are approximated, e.g., if 50% of load values are approximated sequentially on a CPU or in parallel on a GPU the proposed LVA will yield the same quality. Table 3.5 shows the normalized

Table 3.5: Comparison of NRMSE of the Proposed ML-LVA with [24]

Approximate Level (n)	**RFVP** [24]	**Proposed ML-LVA**
1	12.21%	4.00%
3	18.65%	5.81%
4	23.46%	6.45%
9	31.25%	8.21%

root mean squared error (NRMSE) for the various approximate levels. From Table 3.5 we can notice that for any approximate level, our model provides at least 3.75× better quality and hence outperforms the work in [24].

3.5 Summary

In this chapter, we presented a comprehensive framework for the ML-LVA tailored to multimedia applications, with a focus on image and audio data. The design and evaluation of the proposed LVA were structured around two main objectives: ensuring efficient load value approximation, i.e., a predictor that requires minimal overhead to operate, while maintaining high-quality performance across a diverse range of multimedia content. To support this, we curated a dataset suite comprising three image datasets—*Flowers*, *Cars*, and *Places*, and one audio dataset, *Babylon 5*. These datasets were chosen for their strong value locality, enabling effective model training and robust predictor performance tailored to the characteristics of image and audio domains. This diversity helped ensure that the LVA generalized well to real-world workloads.

In our quality evaluation, the ML-LVA was applied to six multimedia tasks, namely, image blending, audio blending, image inversion, audio inversion, image binarization and audio binarization. Across all tasks, we assessed the output quality using metrics such as PSNR, NMAE, NRMSE, accuracy, and precision. The results demonstrated

that even at high approximation levels, our LVA preserves an acceptable quality. Image tasks showed minimal degradation at moderate levels, while audio tasks were especially resilient, maintaining perceptual quality under more aggressive approximation. The experiments also revealed varying tolerance across applications. For instance, image inversion was more robust to approximation than image blending, and audio blending tolerated approximation with very little quality loss. Despite a predictable decline in accuracy with higher approximation, the multimedia applications remained usable, validating the practical effectiveness of ML-LVA.

Overall, the ML-LVA achieves a strong balance between computational efficiency and output quality. This efficiency is achieved through the use of a lightweight history-based prediction mechanism, which avoids the added complexity and hardware overhead used in prior work. Its adaptability to both image and audio domains, combined with its performance under approximation, makes it a compelling solution for real-time or resource-constrained multimedia systems. While these results are more conceptual for the quality of the trained model and generated predictor, an integration of this load predictor with the target application is mandatory to assess the impact on the performance.

In the subsequent chapters, we will therefore explore in detail how the proposed ML-LVA can be integrated into both software- and hardware-based systems. In particular, the next chapter will provide an in-depth examination of the ML-LVA implementation within a software runtime environment. Understanding the deployment details of the ML-LVA is crucial for assessing its real-world compatibility, overhead, and scalability. While this chapter focused on prediction accuracy, the next chapter will provide a comprehensive performance analysis, including speedup effects and workload behavior under actual execution conditions.

Chapter 4

Software Implementation of the ML-LVA

4.1 Introduction

Following the development of the machine learning (ML) model that is capable of predicting the load value and testing its accuracy through simulation, this chapter presents its practical realization through a software-based implementation. The objective of this phase is to transform the conceptual design of the ML-LVA into a fully operational system capable of functioning within real-world environments. The software implementation integrates the key components of the ML-LVA into a cohesive framework, designed to meet the requirements of efficiency, scalability, and adaptability.

A software-based implementation offers several important advantages. Foremost among them is the ability to deploy the proposed ML-LVA model onto existing hardware platforms, e.g., off-the-shelf processors, without necessitating modifications to the underlying hardware. In this work, we evaluate the ML-LVA on an x86

platform using the GEM5 simulator [25]. The x86 architecture was chosen due to its widespread adoption, its compatibility with GEM5, and its use in previously proposed LVA [19]. Furthermore, x86 provides a stable and well-documented ISA, making it a suitable baseline for evaluating new design concepts such as the ML-LVA. Additionally, the GEM5 simulator was selected for its cycle-accurate modeling capabilities, extensive configurability, and widespread use in academic research, allowing for detailed architectural analysis and reproducible experimentation.

In addition to its relevance for general-purpose platforms, a software-based deployment is also highly beneficial in the context of application-specific integrated circuits (ASIC) or other custom hardware environments, where available silicon area is often constrained, and the addition of new dedicated hardware units is either impractical or impossible, the integration of the ML-LVA through software provides a viable and efficient solution. This approach ensures that the benefits of the ML-LVA can be realized with minimal disruption to existing system architectures, thereby enhancing system flexibility and reducing development time and cost.

The rest of this chapter is structured as follows. First, we present the methodology employed to achieve the software-based implementation, with an emphasis on the translation process of the ML model into a modular and integrable software structure. Additionally, we present in details how the "safe-to-approximate" load instructions are identified giving its importance in achieving a software-based implementation of the proposed ML-LVA. Subsequently, we present the testing environment, highlighting the configurations used to evaluate the impact of the software integration on system behavior. Thereafter, we present a comprehensive performance analysis, examining the operational efficiency, resulting from the software-based implementation of the ML-LVA when deployed in six multimedia applications.

4.2 Proposed Methodology

This section describes the methodology used to integrate the ML-LVA model discussed in the previous chapter into software. Building on the training and design details previously presented, we focus here on translating the trained model into a form suitable for implementation within the system's software stack. Key considerations include model representation, and the mapping of decision logic to software constructs for efficient execution. Additionally, we explain how the set of load instructions that are safe-to-approximate is determined, forming the basis for applying LVA selectively and effectively. With both the determination of safe-to-approximate instructions and the translation from high-level model to optimized software, a complete implementation of the application can be generated. Thus, in this section, we present Steps ① and ④ of the methodology shown in Figure 1.3.

Figure 4.1 depicts the process of determining the safe-to-approximate load instructions and how we integrate in software the trained model within the error tolerant application. In Step ①, we perform the first step towards determining the safe-to-approximate load instructions by profiling the load instructions and determining the effect on the program. Thereafter, in Step ②, we determine the control flow independent load instructions. This is a crucial step since if the control flow is affected by the approximation, e.g., approximating (predicting) the loop boundary read from the memory, could result in crashes, such as segmentation faults, infinite loops, execution of unintended code, stack corruption and/or breaking the logic of the program. Thus, based on Step ②, we determine the *safe-to-approximate* load instructions by determining those that are not part of the control flow.

Furthermore, as shown in Figure 4.1, in Step ⑤, we compile the high level description of the ML model into an optimized assembly code. Optimizing the

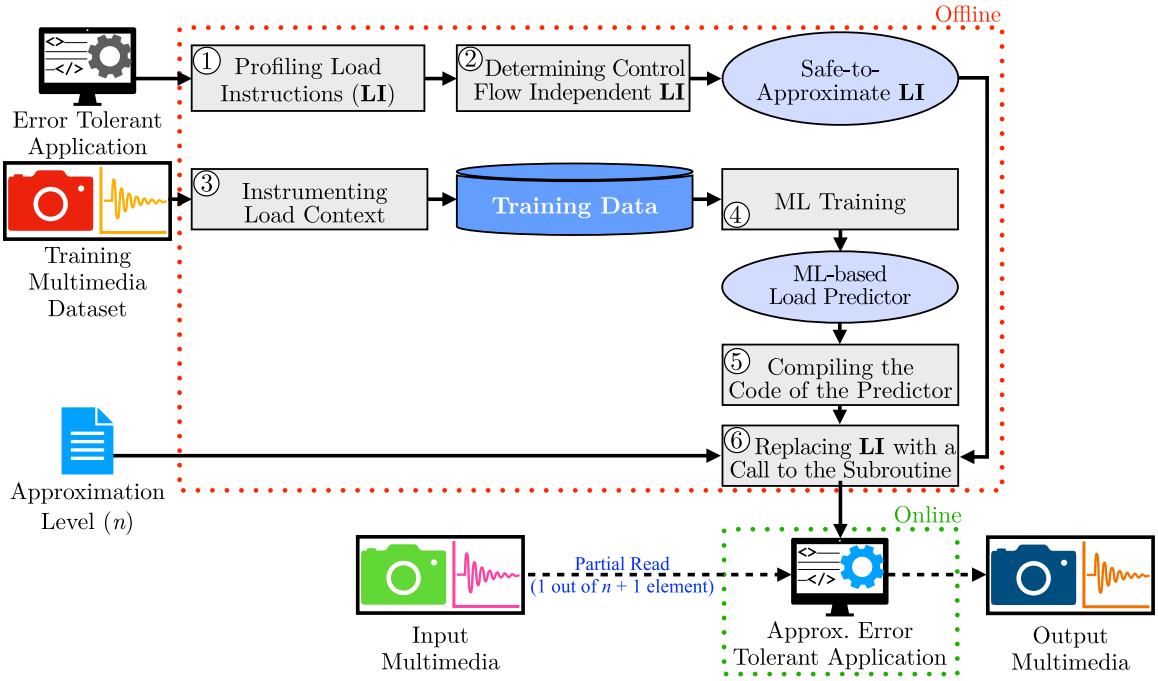


Figure 4.1: Methodology to Implement the Proposed LVA in Software

implementation of an ML-model in assembly is not a straightforward task. For instance, in a tree-based prediction, the tree will consist of *if-else* conditions that translate to conditional branching in assembly. However, branching can consume a large number of cycles and cause the predictor to require a substantial number of clock cycles to predict. On the other hand, since in this thesis the proposed LVA targets a 1-byte load, the predicted value can be from 0 to 255, i.e., 256 unique values. Subsequently, we choose to implement the predictor in a subroutine that uses an unconditional branch. Since the predictor is static, for a given history value, i.e., preceding value, the predicted value will always be the same. Subsequently, we extract the 256 possible predictions from the ML model in order to implement it in assembly. The implementation in assembly consists of using the history value, i.e., preceding value, as a multiplier for the jump address. Although the predictor could be implemented as a lookup array, we adopt a subroutine-based implementation since it scales more effectively for

applications that operate on larger data widths, e.g., 16-bit and 32-bit inputs. For instance, in a 16-bit application, an array-based design may introduce conflict misses in the data cache, which would undermine the purpose of the ML-LVA, as its objective is to alleviate memory bottlenecks. In contrast, a subroutine resides in the instruction cache, which is read-only and therefore experiences lower traffic and simpler access patterns compared to the data cache, which must handle both reads and writes. As a result, the subroutine-based approach reduces the likelihood of cache conflicts allowing the scalability to larger data width. It is noteworthy that the extension of the proposed implementation will lead to a larger size of the code since the instances that need to be covered by the subroutine are much larger. Finally, in Step ⑥ we replace the safe-to-approximate load instructions with a call to the subroutine. The replacement is done by hand since a new program has to be generated where its flow requires an attentive modifications. In the future, compilers could be adapted to accept the list of safe-to-approximate load instructions, allowing this process to be fully automated.

4.3 Implementation of the Predictor

To evaluate the practical feasibility of the proposed ML-LVA, we implement and test it on an x86-based platform. The choice of x86 is motivated by several factors. First, x86 remains one of the most widely adopted ISA, making it highly relevant for both academic research and industry applications. Its mature ecosystem and widespread use across desktop, server, and embedded domains allow for results that are more broadly applicable. Second, x86 is fully supported by the GEM5 simulator, which we use in this work to conduct detailed cycle-accurate simulation and gain system-level insight into predictor behavior. Although the proposed ML-LVA could be implemented on alternative ISAs such as ARM or RISC-V, in this work we limit our scope to x86

Listing 4.1: Assembly Subroutine of the ML-based Predictor for x86 Architecture

```
1  ; ECX contains the history value, i.e.,~the preceding value.
2  ; Multiply ECX~(history value) by 6 since MOV and RET are 6 bytes
3  ; Jump to Base Address~(vals) + ECX
4  ; Move the predicted value to ECX and exit the subroutine
5  predictor:
6         imul ecx , ecx, 6
7         lea  ecx , [vals + ecx]
8         jmp  ecx
9  vals:
10        mov  ecx , 41       ; if history value = 0
11        ret
12        mov  ecx , 18       ; if history value = 1
13        ret
14        mov  ecx , 108      ; if history value = 2
15        ret
16
17        . . .
18        # < skipped portion of the code>
19        . . .
20
21        mov  ecx , 238      ; if history value = 253
22        ret
23        mov  ecx , 240      ; if history value = 254
24        ret
25        mov  ecx , 220      ; if history value = 255
26        ret
```

due to its widespread popularity, its extensive support in GEM5, and its prior use in related work on load value prediction [19]. By targeting x86, we ensure that the ML-LVA can be realistically evaluated in a widely recognized and practical computing context, thereby enhancing the relevance and comparability of our results.

The implementation of the `predictor` subroutine in x86 assembly is shown in Listing 4.1. In this snippet it is assumed that the history value, i.e., preceding value, to be used for the prediction is available in the `ecx` register. In this snippet, if the history value is 0, we must execute the instructions on lines #10 and #11 to predict a value of 41 and exit the subroutine. Alternatively, if the history value is 254, we must

execute the code shown on lines #21 and #22 to predict a value of 240 and exit the subroutine. Since the instruction `mov ecx,#Pred_Val` and `ret` are 5 and 1 byte(s), respectively, we must skip 6 bytes multiplied by the history value, i.e., 6 × `ecx`, to branch to the targeted portion of the code and predict the load value. Subsequently, the history value in `ecx` is multiplied by 6 and the resulting value is stored in `ecx`. Thereafter, we add the address of the label `vals` to the value in `ecx`, i.e., which is 6× history value, where the resulting value is used as a jump address in `jmp ecx`.

Subsequently, we use the optimized compilation of the ML-based predictor in Step ⑤ as shown in Figure 4.1 to integrate the predictor in the application by replacing the safe-to-approximate load instructions with a call to the optimized subroutine. This will generate an application that can be utilized in the online phase where it will perform a partial memory read, i.e., reading 1 out of $n+1$ element, where the rest is predicted using the incorporated predictor. Subsequently, this application will generate the approximated output multimedia.

4.4 Testing Environment

To evaluate the performance impact of the proposed ML-LVA when deployed in software, we use the GEM5 simulator [25] a widely adopted, open-source platform that offers detailed cycle-accurate simulation of processor microarchitectures. By running the assembly codes in GEM5, we ensure that measurements are isolated and unaffected by any background processes or system-level disturbances that could otherwise interfere with timing accuracy. This level of fidelity is essential for our analysis, as it allows us to precisely measure changes in execution timing and microarchitectural behavior resulting from the integration of the proposed ML-LVA—particularly at the memory and application levels, where small variations in load latency can significantly affect the

overall speedup. Although several alternatives exist for simulating processor behavior they generally lack features or does not provide cycle-accurate analysis. One commonly used alternative is QEMU [57], a high-performance functional emulator capable of running full system software stacks. However, QEMU focuses primarily on functional correctness and emulation speed rather than timing accuracy. It does not model pipeline behavior, cache hierarchies, or memory access latency at a cycle-accurate level, making it unsuitable for evaluating fine-grained architectural modifications or microarchitectural optimizations such as the ML-LVA. Other lightweight simulators or instruction set emulators fall into a similar category as they can validate program behavior but lack the timing detail required for precise performance analysis.

Another option is hardware prototyping using platforms such as FPGAs or dedicated emulation systems. While these can provide very high fidelity and even cycle-accurate execution, they often require significant design effort, time, and access to sophisticated commercial platforms such as Cadence Protium [58] and Palladium [59]. These systems offer advanced capabilities for hardware emulation but come with high costs and complex setup procedures, making them less accessible for early-stage architectural exploration. Additionally, implementing a full out-of-order processor pipeline and memory hierarchy on FPGAs or emulators can be challenging, potentially limiting the scope or realism of the simulation.

In contrast, GEM5 strikes an effective balance between accuracy and flexibility. It supports a range of CPU models, including detailed in-order and out-of-order pipelines, and allows full control over cache configurations, memory types, e.g., DDR3 [60] and DDR4 [61], and CPU frequency settings. These capabilities are critical for our study, as they enable us to simulate a range of system configurations that reflect current commercial processor trends. In this thesis the CPU configuration used in GEM5

is based on the most recent trends in commercially available computers. We apply the proposed LVA when varying the cache settings, the type of DRAM, e.g., DDR3 and DDR4, and the frequency of the CPU. For instance, the latest generation of Intel processors [62] has mainly two cache configurations, where low to medium-end processors have the same cache settings while high-end models differentiate in their cache settings [62]. Furthermore, all Intel processors have *efficient* (**E**) and *performance* (**P**) cores where the cache hierarchy is also different. The cache settings of the various Intel processors and cores are summarized in Table 4.1 [62].

In this thesis, we use an acronym to reference the cache configuration of the **E** cores of the **L**ow-end Intel processor as **LE** cache. We apply a similar format to all three other configurations, namely, **LP**, **HE** and **HP**. On the other hand, GEM5 only accepts cache sizes and associativity that are of the power of two. Subsequently, the **LP** and **HP** caches cannot be modeled in GEM5. For this purpose, we created variations based on the Intel cache configurations, which sizes are power of two. The various cache settings used in this thesis are summarized in Table 4.2. For instance, we created **LP0** and **LP1** which are variations of the **LP** where the 10-way set associativity is modified to 8-way and the 1.25MB L2 cache is transformed to 1MB. Furthermore, since the L1 Data cache is 48KB which is a middle value between two

Table 4.1: Cache Settings of the Intel Processor [62]

Description	**L1 Data**	**L1 Instruction**	**L2**
Low/Medium-End Intel Processor – **E** Cores (**LE**)	32KB 8-way	64KB 8-way	2MB 16-way
High-End Intel Processor – **E** Cores (**HE**)	32KB 8-way	64KB 8-way	4MB 16-way
Low/Medium-End Intel Processor – **P** Cores (**LP**)	48KB 12-way	32KB 12-way	1.25MB 10-way
High-End Intel Processor – **P** Cores (**HP**)	48KB 12-way	32KB 12-way	2MB 16-way

Table 4.2: Cache Configurations used to Test the Proposed LVA

Description	L1 Data	L1 Instruction	L2
LE	32KB 8-way	64KB 8-way	2MB 16-way
HE	32KB 8-way	64KB 8-way	4MB 16-way
LP0	32KB 16-way	32KB 16-way	1MB 8-way
LP1	64KB 16-way	32KB 16-way	1MB 8-way
HP0	32KB 16-way	32KB 16-way	2MB 16-way
HP1	64KB 16-way	32KB 16-way	2MB 16-way

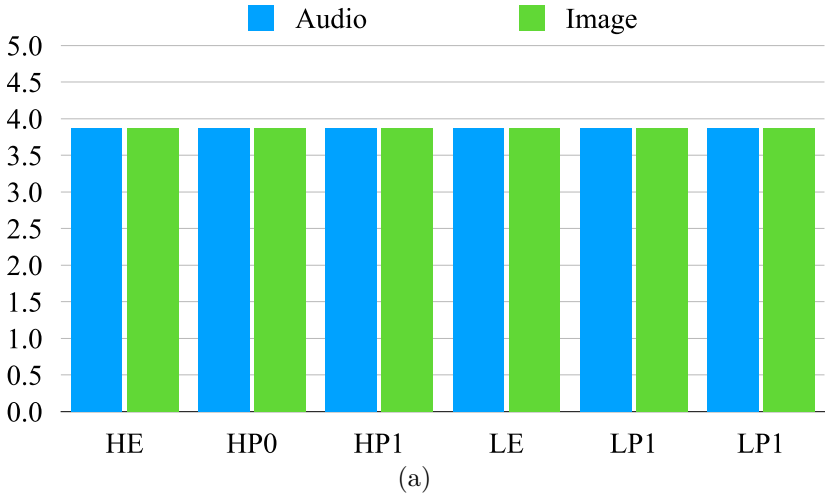
powers of two values, i.e., 32KB and 64KB, the two variations **LP0** and **LP1** are chosen accordingly to avoid biased configuration. Similarly, we created variations based on the **HP** cache configuration named **HP0** and **HP1**. In addition to the variations in cache configurations, we tested the proposed LVA while varying the frequency of the CPU from 1 GHz to 4 GHz. These values are chosen based on the base frequencies of the latest generation of Intel processors. Finally, we add a layer of variations in our CPU configuration in GEM5 where we select two types of DRAM, namely, "DDR3_1600_8x8" and "DDR4_2400_8x8". Subsequently, with six cache configurations, four frequencies and two DRAM types, we were able to generate 48 hardware configurations that cover all possible combinations. Furthermore, the configurations used in GEM5 use the "x86 Timing Simple CPU"[26], where we opt for an architecture that has L1 (separate) and L2 (unified) caches only.

4.5 Performance Analysis

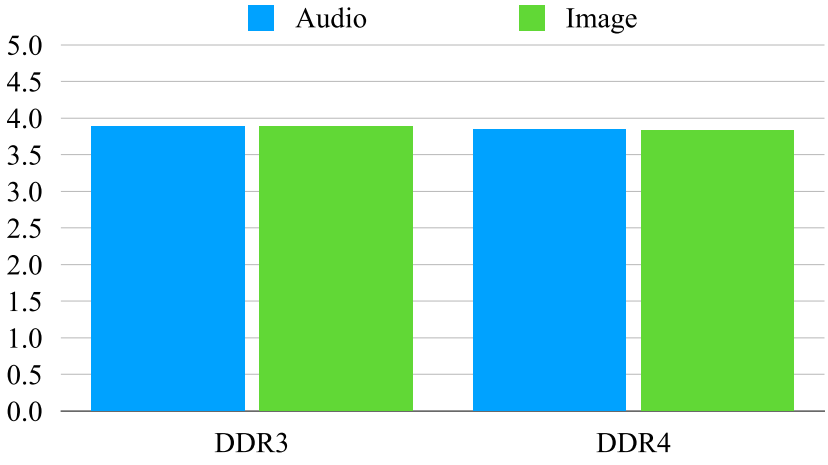
For the various multimedia application, the 48 hardware configurations are tested at two levels, the application level, where we measure the overall speedup as well as at the level of the memory load operations only. For the performance analysis, we tested the six multimedia applications using 16 and 129 images from the *Places* and *Cars* dataset, respectively. Additionally, we selected 65 audio files to analyze the performance of the audio processing applications when implementing the proposed ML-LVA. With 19 different levels of approximation, 48 hardware configurations and the image and audio combinations, we base our analysis in the sequel on 4,381,440 experiments that were conducted on a machine with two 32-cores AMD EPYC 7001 series CPUs and 200GB of RAM.

The simulation process in GEM5 is inherently time-consuming due to its detailed, cycle-accurate modeling of hardware components. This level of precision significantly increases the runtime for each individual simulation. Given the extensive parameter space in our study, the total number of potential simulation combinations is enormous. Even with High-Performance Computing (HPC) resources capable of large-scale parallelization, a full sweep of all combinations would be computationally excessive. Consequently, the need to balance comprehensive performance analysis with available computational resources motivated the decision to limit our experiments to a representative subset of 4,381,440 simulation instances. This subset was carefully selected to provide meaningful insights into system behavior while keeping the total runtime feasible.

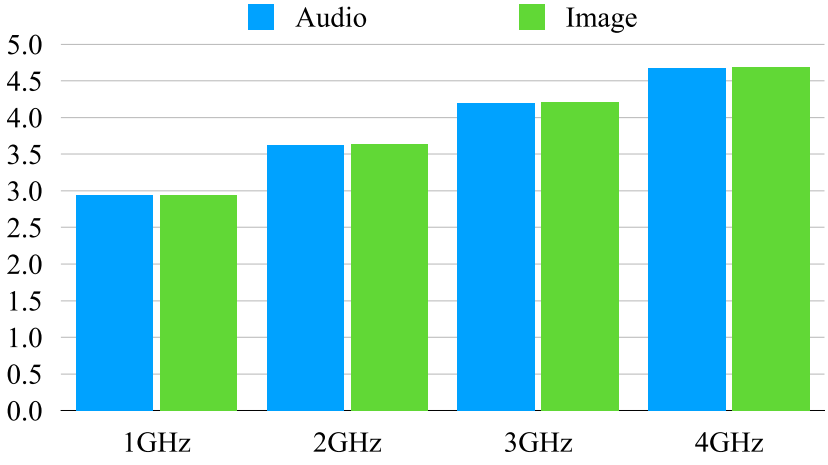
Figure 4.2 depicts the average speedup in the memory operation achieved when varying one hardware configuration at a time. From Figures 4.2(a) and 4.2(b), we can notice that the variations in the cache and DRAM configurations result in a



(a)



(b)



(c)

Figure 4.2: Average Speedup in Memory Access when varying (a) Cache, (b) DRAM, and (c) CPU Frequency Settings

minimal impact on the speedup achieved. On the other hand, we can notice in Figure 4.2(c) that for a higher frequency, the proposed LVA achieves a higher speedup. The various cache and DRAM configurations achieved a similar trend in speedup as these variations affect the execution of the exact and approximate models in the same ratio. Alternatively, a higher frequency achieved a higher speedup since the proposed LVA runs at CPU speed while conventional loads are limited by the memory wall. Subsequently, the proposed LVA achieves a higher performance gain at higher CPU frequencies.

4.6 Experimental Results

In this section, we will present a detailed analysis of the performance characteristics of the proposed LVA technique when applied to six representative multimedia applications, namely, multiplication-based image blending [32], multiplication-based audio blending known as Ring Modulation (RM) [33], audio binarization known as infinite clipping [35], image binarization known as image thresholding [34], polarity inversion of audio [37], and image inversion known as image negatives [36] when varying the frequency and the approximate level (n). The aim of this section is to evaluate how ML-LVA influences computational performance, specifically execution time across both image and audio processing domains. To this end, we measure and compare the execution speedup on two levels, namely, the application and the memory. Performance is analyzed by computing the speedup factor achieved when using the proposed ML-LVA as opposed to the baseline execution without the ML-LVA, across varying system frequencies and approximate levels (n). Due to the large number of simulation instances, we will limit the presentation to the average speedups achieved when varying the frequency.

The remainder of this section is organized into two main categories: image

processing, and audio processing, which are further broken down into three core operations: blending, inversion, and binarization. For each application, we examine the average, the best- and worst-case scenarios, i.e., minimum and maximum, speedup and how performance is impacted by changes in processor frequency and the approximate level (n).

4.6.1 Image Processing

We analyze the performance of image processing tasks when executed on the memory processor using the proposed ML-LVA, compared to their execution on a conventional execution. The evaluated tasks include image blending, image inversion, and image binarization, each representing a distinct class of operations with varying computational complexity and memory access patterns.

4.6.1.1 Image Blending

The performance results for image blending are shown in Figure 4.3 and reveal clear trends in both application overall and memory load speedups as the operating frequency increases from 1 GHz to 4 GHz. A detailed examination shows that while both speedup types generally benefit from increased frequency, the gains are more pronounced in the memory speedup domain, particularly under higher approximate levels.

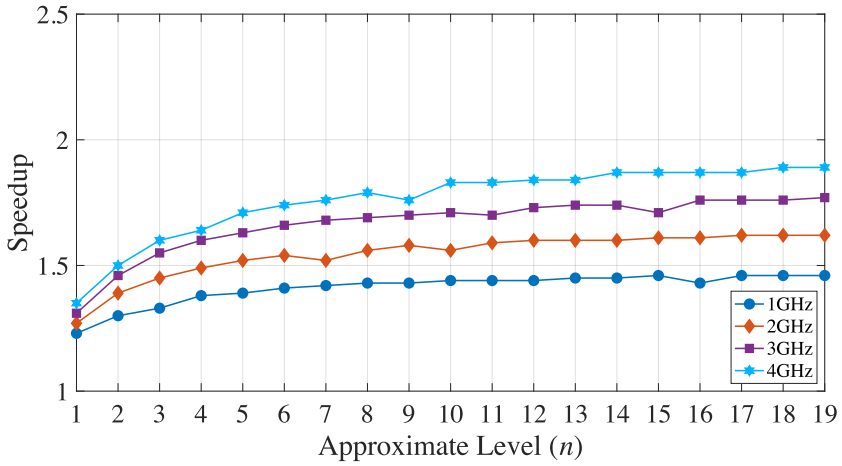
At 1 GHz, as shown in Figure 4.3(a), we notice that the application overall speedups for image blending range from a minimum of 1.23× to a maximum of 1.46×, with an average speedup across all approximate levels of 1.41×. In contrast, we notice from Figure 4.3(b) that the speedup in memory load operations at this frequency demonstrate a slightly wider variation, with minimum values around 1.45×, peaking

at 3.58×, and averaging close to 2.95×. These initial results highlight that even at the lowest frequency tested, the memory speedup is substantially higher compared to the one achieved at the application level.

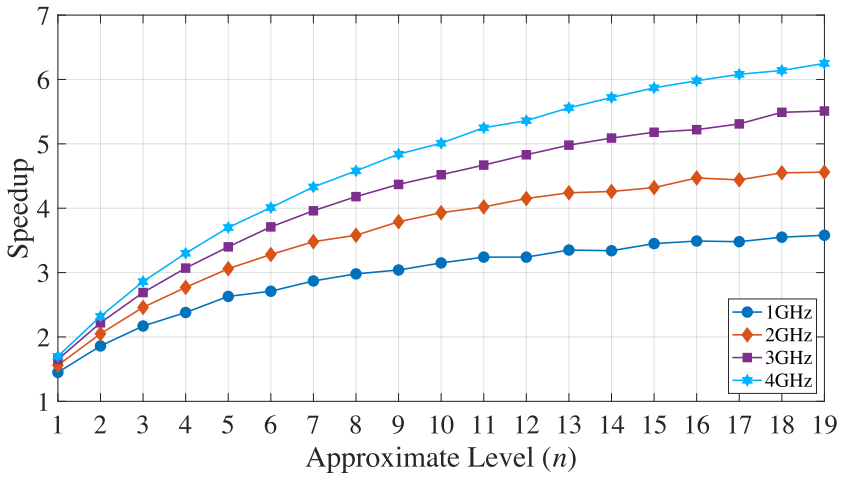
As the frequency increases to 2 GHz, the overall speedups show a modest improvement, with values ranging from 1.27× to 1.62×, and an average speedup of 1.54× as shown in Figure 4.3(a). Memory speedups, however, scale more aggressively, reaching up to 4.56×, with a minimum near 1.56×, and averaging 3.63×. The increasing advantage of memory load speedup suggests that, as the system clock rate increases, the memory access patterns and cache behavior become increasingly a significant performance factors in image blending workloads.

At 3 GHz, the trend continues with overall speedups fluctuating between 1.31× and 1.77×, producing an average of 1.67×. The memory speedups at this frequency are even more notable, ranging from 1.65× to 5.51×, and averaging 4.21×. Finally, at the highest tested frequency, i.e., 4 GHz, the overall speedups reach their peak range for image blending, from 1.35× to 1.89×, yielding an average of 1.76×. Memory speedups also peak at this level, with values spanning from 1.70× to 6.25×, and an average speedup of 4.68×. These figures confirm the cumulative benefit of frequency scaling when deploying the proposed ML-LVA in software.

Finally, as illustrated in Figure 4.3(a), we observe that the speedup curve begins to exhibit a flattening trend once the approximation level exceeds $n = 10$. This behavior can be attributed to the mathematical nature of the approximation scheme employed in the ML-LVA, where the fraction of load instructions subject to approximation is given by $\frac{n}{n+1}$. As n increases, the incremental change in this fraction diminishes, asymptotically approaching an upper bound of 1, i.e., 100% approximation. Consequently, the proportion of approximated loads increases rapidly at lower values



(a)



(b)

Figure 4.3: Average Speedups for Image Blending: (a) Overall, and (b) Memory Loads

of n, but the rate of increase slows significantly as n continues to grow. This saturation effect limits the additional performance gains achievable at higher approximation levels, resulting in the observed plateau in the speedup curve. Thus, while smaller values of n yield noticeable improvements, the marginal benefits become progressively smaller beyond $n = 10$, highlighting a point of diminishing returns in terms of speedup.

In summary, the results for image blending indicate a consistent, frequency-dependent improvement both overall and memory load speedup However, the rate of improvement is greater on the memory side, with the gap widening as frequency

increases. This suggests that for high-performance image blending operations, especially on systems operating at or above 3 GHz, memory load optimization techniques offer more substantial performance gains compared to application level techniques alone.

4.6.1.2 Image Inversion

The performance results shown in Figure 4.4 for image inversion exhibit clear trends in both application overall and memory load speedups as the operating frequency increases from 1 GHz to 4 GHz. Similar to the other image blending tasks, memory load speedups outperform overall speedups, particularly as the frequency rises. This suggests that image inversion, while computationally intensive, benefits greatly from improvements in memory access and cache management.

At 1 GHz, we notice from Figure 4.4(a) that the overall speedups for image inversion range from a minimum of 1.25× to a maximum of 1.81×, with an average speedup of around 1.68×. In comparison, memory load speedups show a slightly wider variation, ranging from 1.54× to 4.17×, and averaging 3.36× as shown in Figure 4.4(b). These results suggest that at the lowest frequency tested, speedups in the memory operation are significantly higher than the overall speedups when tested in the image inversion tasks.

When the frequency is increased to 2 GHz, the overall speedups improve slightly, ranging from 1.33× to 2.06×, with an average speedup of 1.88×. Memory load speedups also improve, ranging from 1.64× to 5.17×, with an average of 4.02×. The widening gap between memory loads and overall speedups at this frequency emphasizes the growing influence of memory access patterns and cache behaviors on the overall performance of image inversion, especially as the clock rate increases.

At 3 GHz, the overall speedups range from 1.37× to 2.25×, yielding an average of 2.03×. Memory load speedups continue to show a more pronounced improvement, ranging from 1.70× to 6.70×, with an average of 4.57×. This further confirms the trend that memory load optimizations are the key driver of performance gains as frequency increases, especially for operations that involve intensive pixel-level manipulation, such as image inversion.

At the highest frequency of 4 GHz, the overall speedups reach their peak range, from 1.41× to 2.43×, with an average speedup of 2.16×. Memory load speedups also peak at this frequency, ranging from 1.75× to 6.77×, with an average of 5.01×. These results clearly demonstrate the cumulative benefits of frequency scaling when deploying the proposed ML-LVA.

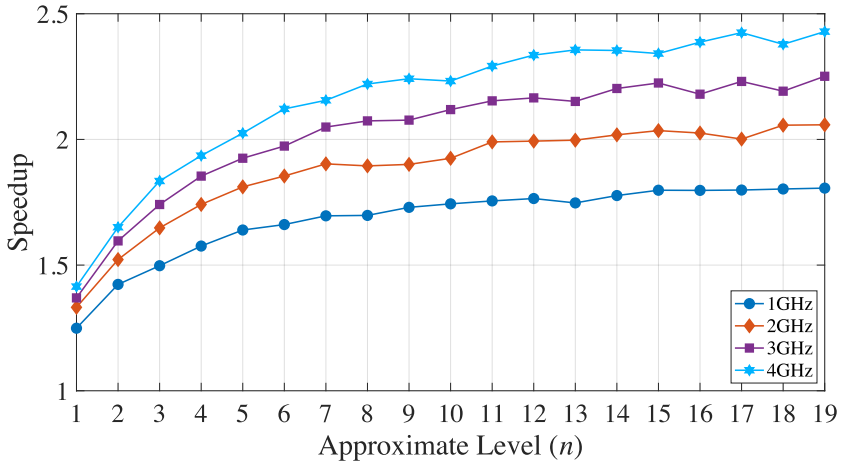
Similar to the behavior observed in the image blending task, Figure 4.4(a) shows that for all four frequencies the overall speedup in image inversion begins to exhibit a flattening trend beyond approximation level $n = 10$. This pattern arises from the approximation ratio which approaches 1 as n increases, leading to reduced performance gains at higher as n increases. However, unlike image blending, image inversion continues to show a modest increase in speedup even beyond this point, indicating that the performance benefits of further approximation, are still present.

4.6.1.3 Image Binarization

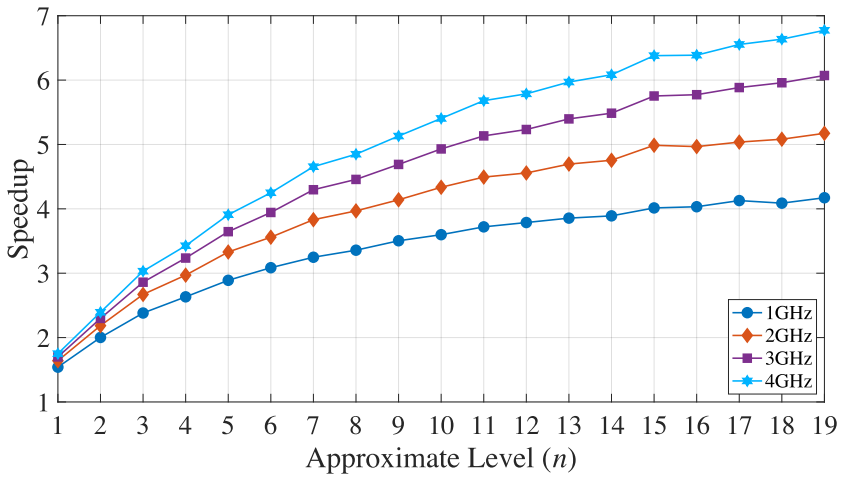
Figure 4.5 depicts the performance results of the image binarization when deploying the proposed ML-LVA and varying the frequency from 1 to 4 GHz. The performance results for image binarization also illustrate a clear trend in both application overall and memory load speedups as the operating frequency increases from 1 GHz to 4 GHz. Notably, similar to the other image processing tasks, the memory load speedup show

more substantial improvements than application overall speedups, particularly at higher frequencies. This indicates that binarization, being a memory-bound operation, benefits significantly from improvements in memory access patterns and caching efficiency.

From Figure 4.5(a), we notice that at 1 GHz, the overall speedups for image binarization range from a minimum of 1.26× to a maximum of 1.85×, with an average of 1.71× across all approximate levels. Furthermore, from Figure 4.5(b), we notice the memory load speedups at this frequency show a wider range, with a minimum



(a)

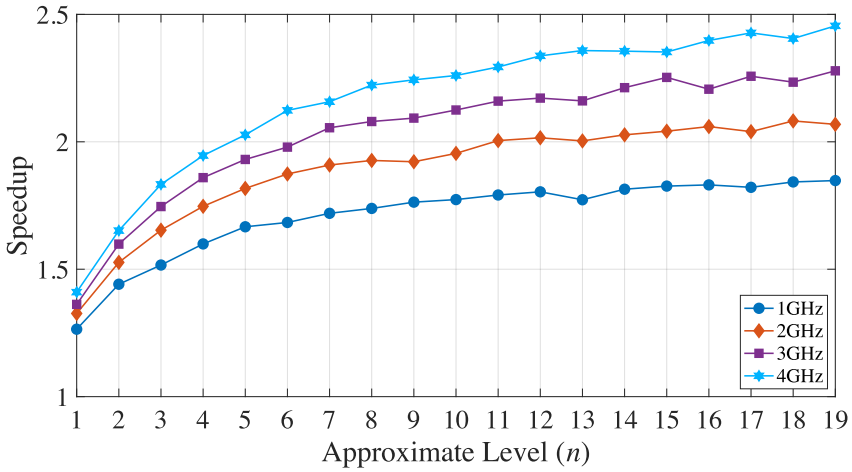


(b)

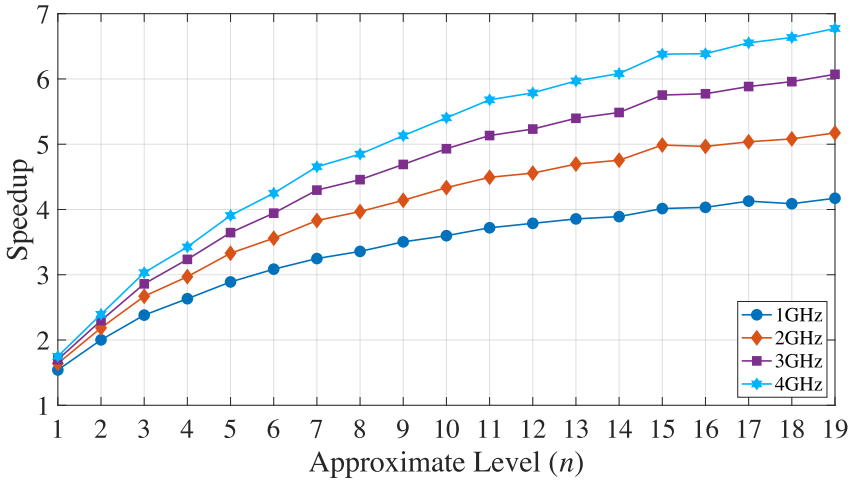
Figure 4.4: Average Speedups for Image Inversion: (a) Overall, and (b) Memory Loads

value of 1.54×, peaking at 4.17×, and an average of 3.36×. When the frequency is raised to 2 GHz, the overall speedups show a moderate increase, ranging from 1.32× to 2.08×, with an average of 1.89×. In contrast, the memory load speedups expand further, with values ranging from 1.64× to 5.17× and averaging 4.02×.

At 3 GHz, the overall speedups range from 1.36× to 2.29×, yielding an average of 2.04×. Meanwhile, the memory load speedups continue to scale more aggressively, ranging from 1.70× to 6.07×, with an average of 4.56×. At 4 GHz, the overall speedups



(a)



(b)

Figure 4.5: Average Speedups for Image Binarization: (a) Overall, and (b) Memory Loads

reach a peak range of 1.41× to 2.45×, with an average of 2.17×. Furthermore, the memory load speedups achieve the highest values, ranging from 1.75× to 6.77×, with a mean of 5.00×. The clear separation between the overall and the memory speedups reinforces the view that image binarization, much like other pixel-based operations, is increasingly bottlenecked by memory access as the frequency increases, making it highly sensitive to memory load optimization techniques.

The flattening trend observed in image binarization closely resembles that of image inversion across the four tested frequencies, as shown in Figure 4.5(a). Although higher approximation levels continue to yield performance improvements, the rate of increase becomes less significant. This behavior is attributed to the approximation ratio $\frac{n}{n+1}$ approaching 1, which has a reduced impact on the percentage of load instructions approximation as n increases.

In summary, the results for image binarization reinforce the benefit of deploying the proposed ML-LVA to reduce the execution time of image binarization. The performance gains from the application overall speedups are steady but limited, while memory load improvements lead to more significant and consistent performance improvements. As frequency increases, the gap between application and memory speedups widens.

4.6.2 Audio Processing

We now present the performance evaluation of audio processing tasks executed on the processor, where the input data is fetched from memory through load operations. The objective is to assess the performance benefits of using the proposed ML-LVA. Unlike image processing, audio processing involves continuous-time data that is typically processed in frames or windows, making latency and throughput critical metrics. These tasks also vary in their degree of arithmetic and branching complexity, which directly impacts how well they benefit from near-memory execution. As with the

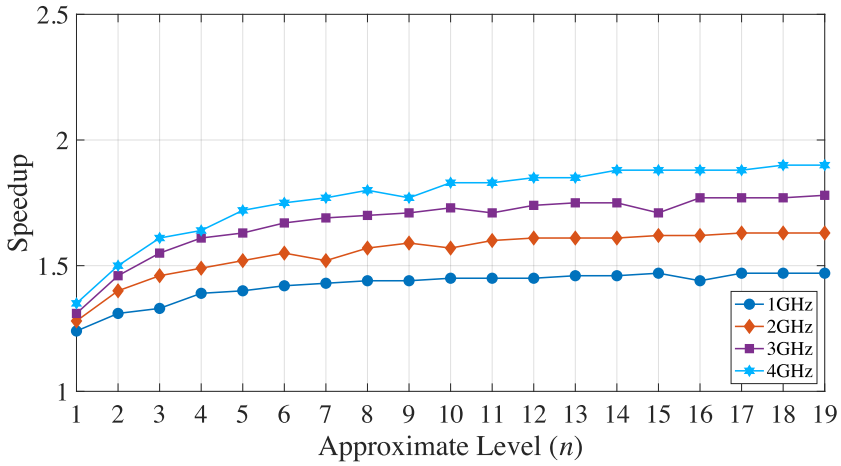
image processing tasks, we evaluate performance at multiple operating frequencies, namely, 1 GHz, 2 GHz, 3 GHz and 4 GHz.

The performance results are analyzed in terms of average speedup, as well as minimum and maximum observed values, to capture both the overall effectiveness and the consistency of the proposed architecture. Additionally, we discuss how the processor frequency influences performance scaling in each task and examine which types of operations yield the most substantial gains under LVA-enabled memory execution. This analysis helps to identify trends in audio processing workloads that are best suited when deploying the proposed ML-LVA in software.

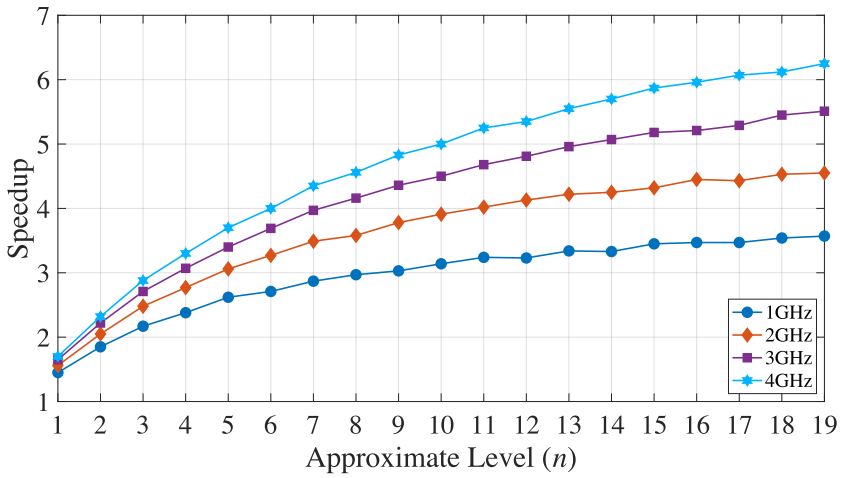
4.6.2.1 Audio Blending

The performance results for audio blending, shown in Figure 4.6, depict a similar trend to those observed in image processing tasks, with clear improvements in both application overall and memory load speedups as the operating frequency increases from 1 GHz to 4 GHz. However, audio blending, being an audio-specific operation, exhibits more variability in performance based on the frequency and optimization level.

Figure 4.6(a) depicts the application overall speedups. We notice from this figure a flattening pattern as n surpasses 10. This trend closely mirrors the one observed in the image blending task. Additionally, we can notice from Figure 4.6(a) that at a frequency of 1 GHz, the overall speedups for audio blending range from a minimum of 1.24× to a maximum of 1.47×, with an average of 1.42×. From Figure 4.6(b), we notice that the memory load speedups show a broader range, from 1.45× to 3.57×, averaging 2.94×. As the frequency increases to 2 GHz, the overall speedups show a slight improvement, ranging from 1.28× to 1.63×, with an average of 1.55×. The



(a)



(b)

Figure 4.6: Average Speedups for Audio Blending: (a) Overall, and (b) Memory Loads

memory load speedups also scale up, ranging from 1.56× to 4.55×, with an average of 3.62×.

At 3 GHz, the overall speedups range from 1.31× to 1.78×, with an average of 1.67×. The memory load speedups show further scaling, with values ranging from 1.65× to 5.51×, and an average of 4.21×. The consistent growth in memory load speedups with higher frequencies suggests that optimized memory access plays a larger role in performance. This trend reflects the intensive memory and processing needs of audio blending.

At the highest tested frequency of 4 GHz, the overall speedups range from 1.35× to 1.90×, with an average speedup of 1.77×. The memory load speedups continue to increase at a higher pace, reaching values from 1.70× to 6.25×, with an average of 4.67×. Overall, the results for audio blending show a consistent trend, where increasing the frequency leads to a higher performance when deploying the proposed ML-LVA in software.

Similar to the previously presented applications, increasing the approximation level leads to a saturation in speedup, shown by a flattening curve. At the memory level, this flattening occurs in fewer patterns and affects lower CPU frequencies more, while higher frequencies are less impacted.

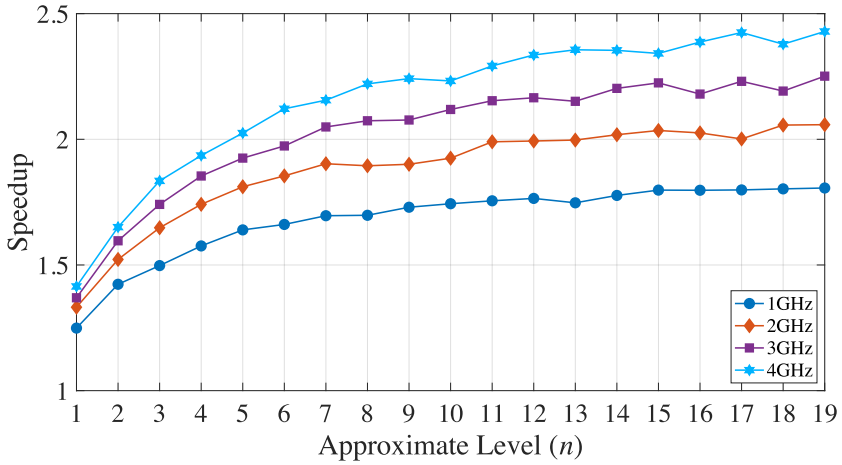
4.6.2.2 Audio Inversion

Figure 4.7 depicts the speedup in the memory load operation and the application for the audio inversion task. The performance results for audio inversion show clear improvements in both application overall and memory load speedups as the operating frequency increases from 1 GHz to 4 GHz. However, similar to other computationally intensive tasks, the speedup at the level of memory loads is substantially higher compared to the one at the application level, particularly as the frequency rises. This suggests that audio inversion, a process involving transformation of signal amplitudes, is also highly sensitive to memory access speeds and cache efficiency.

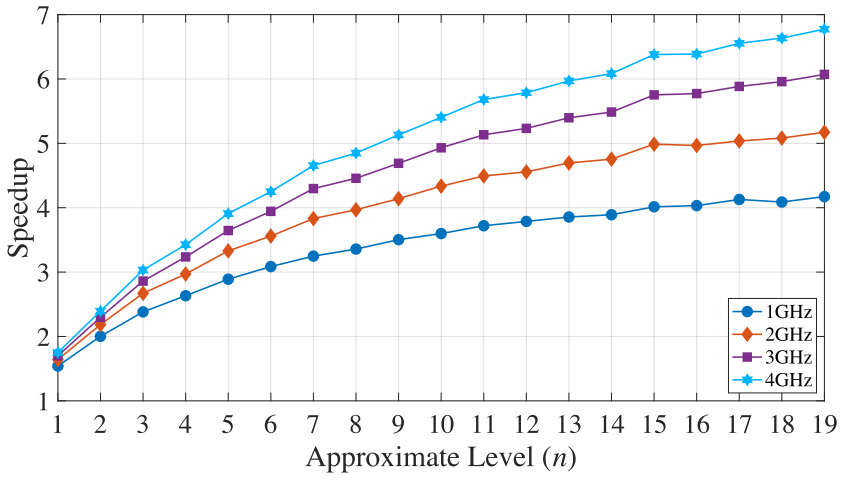
At 1 GHz, the overall speedups for audio inversion range from a minimum of 1.25× to a maximum of 1.81×, with an average speedup of 1.68× as shown in Figure 4.7(a). In comparison, from Figure 4.7(b), we notice a wider range of speedups in the memory load operation, with minimum values 1.54×, peaking at 4.17×, and averaging 3.36×. When the CPU operates at 2 GHz, the overall speedups show a modest improvement,

ranging from 1.33× to 2.06×, with an average speedup of 1.88×. Memory load speedups also scale up, ranging from 1.64× to 5.17×, with an average of 4.02×.

At 3 GHz, the overall speedups range from 1.37× to 2.25×, with an average of 2.03×. Memory load speedups continue to improve significantly, ranging from 1.70× to 6.07×, with an average of 4.56×. When increasing the CPU frequency to the highest tested setting, i.e., 4 GHz, the overall speedups reach a range of 1.41× to 2.43×, with an average speedup of 2.16×. The memory load speedups peak at this frequency, ranging from 1.75× to 6.66×, with an average of 5.00×. These results emphasize the



(a)



(b)

Figure 4.7: Average Speedups for Audio Inversion: (a) Overall, and (b) Memory Loads

importance of memory subsystem performance in audio inversion tasks, especially at higher frequencies, where memory load optimizations provide significant improvements over application level optimizations. In summary, similar to the other applications, the results for audio inversion demonstrate a consistent trend where memory load speedups are more significant than application level improvements, particularly at higher frequencies.

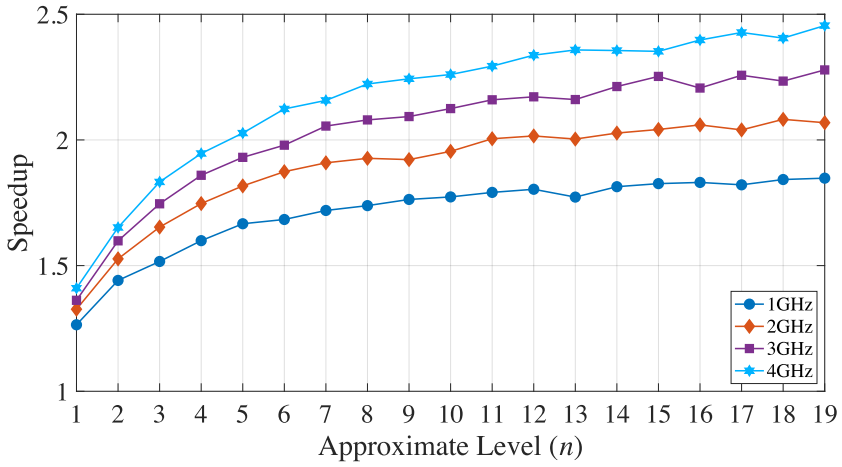
The speedup trend at the application level for audio inversion depicted in Figure 4.7(a), is consistent across all four tested frequencies and closely resembles the patterns observed in image inversion and binarization. In all cases, the speedup curve shows a gradual flattening as the approximation level increases, reflecting the reduced marginal gains at higher levels of approximation.

4.6.2.3 Audio Binarization

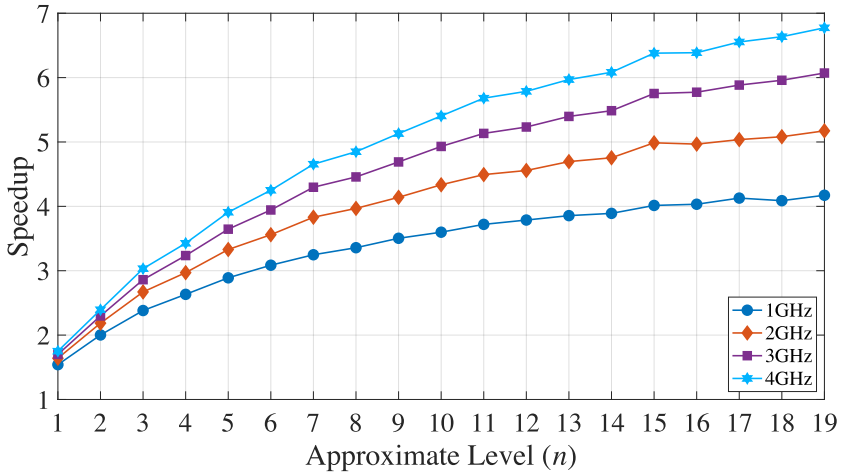
The performance results for audio binarization reveal notable trends in both application overall and at the level of memory loads speedups as the operating frequency increases from 1 GHz to 4 GHz. The speedup results are shown in Figure 4.8. As with other audio and image processing tasks, speedup in the memory load operation continue to provide larger speedup compared to the overall application, particularly. From the overall speedups shown in Figure 4.8(a), we observe a strong resemblance to the trends seen in image inversion, image binarization, and audio inversion. Across all four frequencies, the speedup curves gradually flatten as the approximation level increases. Nonetheless a slight and steady increase in speedup is still maintained.

When testing with the slowest frequency, i.e., 1 GHz, we notice from Figure 4.8(a) that the overall speedups for audio binarization range from a minimum of 1.26× to a maximum of 1.85×, with an average speedup of 1.71×. The memory load speedups at

this frequency show a broader range, from 1.54× to 4.17×, with an average of 3.36× as shown in Figure 4.8(b). At 2 GHz, the overall speedups improve slightly, ranging from 1.33× to 2.08×, with an average of 1.89×. The memory load speedups also show a more substantial increase, ranging from 1.64× to 5.17×, with an average of 4.02×. At 3 GHz, the overall speedups range from 1.36× to 2.27×, with an average of 2.04×. The memory load speedups continue to scale, ranging from 1.70× to 6.07×, with an average of 4.56×. When the frequency is increased to 4 GHz, the overall



(a)



(b)

Figure 4.8: Average Speedups for Audio Binarization: (a) Overall, and (b) Memory Loads

speedups ranges from 1.41× to 2.45×, with an average of 2.17×. The memory load speedups peak at this frequency, ranging from 1.75× to 6.77×, with an average of 5.01×. Similar to the earlier applications, a rise in approximation level results in speedup saturation, which is reflected by the flattening of the curve in Figure 4.8.

In conclusion, the results for audio binarization reflect a consistent pattern of greater performance gains from memory load optimizations, particularly as frequency increases.

4.6.3 Comparison with Related Work

Evaluating new techniques against existing solutions is essential to understand their relative strengths and weaknesses. In this section, we present a detailed comparison between the software-based implementation of the ML-LVA and the LVA proposed in [19], focusing on performance improvements and practical applicability within memory-centric architectures. By benchmarking against prior work, we aim to highlight the advantages offered by ML-LVA in accelerating multimedia processing workloads on CPU-based systems, thus providing a clear perspective on its potential impact and feasibility for real-world deployment.

We limit the comparison to this work since the authors of [24] targeted GPU architectures while the proposed ML-LVA targets CPU architectures. From Table 4.3, we can notice that the proposed LVA provides a much higher speedup for each of the approximate levels when compared to the one proposed in [19]. For instance, for $n = 17$, the LVA proposed in [19] achieved an average speedup of 1.08 over all the experiments conducted. In contrast, for the same approximate level, our proposed LVA achieved a speedup of 1.98 over all the experiments, i.e., average of all hardware configurations and all audio and image applications. This trend can be noticed

throughout all approximate levels where the proposed ML-LVA outperforms the LVA proposed in [19]. Since the proposed LVA delivers a better quality and a higher speedup than the LVA proposed in [19] for the various approximate levels, we can conclude that the LVA we propose is superior.

Table 4.3: Speedup Comparison of the Proposed Software-based ML-LVA with [19]

Approximate Level (n)	**LVA** [19]	**Proposed ML-LVA**
1	1.08	1.29
3	1.07	1.49
5	1.08	1.57
9	1.08	1.62
17	1.08	1.68

4.7 Summary

This chapter detailed the software-based implementation of the ML-LVA, transitioning the conceptual framework into a practical system suitable for deployment on off-the-shelf existing hardware hardware. The implementation was tested using the x86 architecture, leveraging its widespread adoption, compatibility, and stable ISA. All experiments are conducted through cycle-accurate simulation in GEM5. We presented the methodology which involves a careful profiling of load instructions with the aim of identifying those that are "safe-to-approximation, ensuring no disruption to control flow. We presented how the trained machine learning model is translated into highly optimized x86 assembly code, designed to minimize branching overhead. This is achieved through a branchless, jump-table-based predictor subroutine, which enables fast and efficient predictions during execution.

The presented testing environment is comprehensive as we evaluated the implementation across 48 different hardware configurations varying cache sizes, DRAM

bandwidth, and CPU frequencies. GEM5 was chosen for its balanced combination of simulation accuracy and flexibility, offering advantages over alternatives such as QEMU or FPGA emulation. Performance analysis includes a vast experimental set of over 4.3 million runs covering six multimedia applications involving image and audio processing tasks such as blending, inversion, and binarization. The results revealed that increasing CPU frequency significantly improves speedup, as ML-LVA benefits primarily from CPU speed, whereas conventional load instructions remain bound by memory access latency. In contrast, variations in cache and DRAM configurations showed minimal influence on speedup, which scales predominantly with frequency. Furthermore, the analysis identified reduced gain in speedup at high approximation level of ten, reflecting an asymptotic trend in the approximation ratio.

Speedup measurements demonstrated substantial gains, with application overall speedups reaching up to 2.45 times in audio binarization at 4 GHz, and memory load speedups peaking at 6.77 times in image inversion at the same frequency. When compared to prior state-of-the-art LVA, the proposed ML-LVA implementation outperforms significantly, achieving an average speedup of 1.98× at an approximation level of 17, compared to 1.08×. These findings confirm that the software implementation of ML-LVA is both feasible and efficient, delivering superior performance particularly in memory-intensive multimedia workloads. The results underscore the potential for real-world adoption in both general-purpose processors and resource-constrained environments such as ASICs, notably without requiring any hardware modifications.

Following the comprehensive discussion of the software-based implementation of the proposed ML-LVA in this chapter, it is important to acknowledge the inherent limitations of the software solution. While effective in demonstrating feasibility and

based on existing off-the-shelf hardware, the software implementation suffers from higher latency and limited throughput, which may constrain performance improvements in data-intensive and latency-sensitive applications. These constraints highlight the necessity to explore a hardware-based implementation that can better meet the demands of modern computing systems and provide more accurate evaluation of the proposed ML-LVA.

The primary objective of transitioning to a hardware solution is to translate the predictive capabilities of the ML-LVA into a more computationally efficient form that can be tightly integrated within newly developed computer architectures. This shift is motivated by the growing need for low-latency, high-throughput systems capable of mitigating the memory access bottlenecks that often limit performance in contemporary workloads. A hardware-centric approach offers several advantages, including the potential to exploit parallelism, ensure pipeline compatibility, and leverage architectural specialization to accelerate load value prediction. Furthermore, hardware implementation can more effectively support aggressive speculative execution strategies widely adopted in current microarchitectural designs. In the next chapter, we thoroughly evaluate the hardware-based ML-LVA to assess its feasibility, performance gains, cost-effectiveness, and practical viability.

Chapter 5

Hardware Implementation of the ML-LVA

5.1 Introduction

A hardware-centric approach offers the potential to deliver significantly higher performance through parallelism, pipeline compatibility, and architectural specialization. Moreover, such an implementation is better suited to support aggressive speculative execution strategies commonly employed in contemporary microarchitectural designs. Therefore, evaluating the ML-LVA in hardware is essential for thoroughly assessing its feasibility, performance, cost-effectiveness, and overall viability when deployed in real-world hardware platforms. The deployment of the ML-LVA in a hardware implementation arises a variety of technical challenges that are distinct from those encountered in the software development. A critical consideration is the need to maintain a delicate balance between the hardware resource overhead incurred by the predictor and the efficiency gains it is expected to provide. The design has to conform to constraints such as limited logic gates, finite memory block

availability and stringent timing requirements imposed by clock frequency ceilings. These constraints necessitate a deliberate process of simplification, optimization, and architectural tailoring to ensure that the implementation remains both lightweight and effective. Furthermore, the hardware-based ML-LVA is required to integrate smoothly with existing processor infrastructure, particularly with the processor pipeline where timing and latency considerations are prominent. In this context, it is imperative that the predictor produces outputs within a constrained number of clock cycles to avoid introducing pipeline stalls, which would undermine the intended performance improvements. These requirements, in conjunction with the need for generalizability across various computer architectures, play a pivotal role in shaping the hardware architecture described in the subsequent sections of this chapter.

We propose to implement the hardware-based ML-LVA as an accelerator of the CVA6 architecture, which is a RISC-V processor. This choice is motivated by several factors relevant to achieving an efficient hardware implementation. Primarily, the CVA6 is an open-source, industry-grade RISC-V core that offers complete architectural transparency, enabling extensive customization and seamless integration of novel hardware modules such as the ML-LVA. Its open-source nature permits thorough examination and modification of the processor pipeline, which is essential for incorporating the ML-LVA as a custom accelerator with minimal integration overhead. Furthermore, the widespread adoption of the RISC-V instruction set and the modular design principles of the CVA6 make it a suitable platform to demonstrate the generalizability and scalability of the proposed hardware-based implementation. The CVA6 includes separate Level 1 instruction and data caches connected via an AXI4 bus to a shared Level 2 cache and external DDR4 memory, providing a realistic environment to evaluate the impact of the hardware implementation of the proposed

ML-LVA. Using this setup, we observed application overall speedups above 1.08× and memory operation speedups up to 1.73×, confirming the effectiveness of the hardware-based ML-LVA.

In the sequel, we will first introduce the methodology to realize a hardware implementation of the proposed ML-LVA, outlining the design process and rationale behind key architectural decisions. We then describe how the machine learning model, originally trained in a software environment, was systematically translated into a modular, synthesizable hardware block and integrated into the CVA6 general-purpose processor as a custom accelerator. Following the integration, we detail the construction of the complete testbench environment used to enable both functional validation and performance evaluation. This includes a comprehensive explanation of the experimental setup, as well as the tools and testing infrastructure employed throughout the assessment process. Thereafter, we present a comprehensive performance analysis, evaluating the effectiveness of the hardware-deployed ML-LVA across a range of multimedia workloads. These include four representative applications that vary in computational characteristics and memory behavior, thereby offering a robust basis for assessment.

5.2 Proposed Methodology

This section outlines the approach taken to integrate the ML-LVA model, introduced in Chapter 3, into a hardware implementation. Rather than focusing on software translation, the discussion here centers on adapting the trained model for synthesis and deployment within a hardware design flow. Particular attention is given to how the ML-LVA is integrated in a processor and how the applications will make use of the new hardware. Accordingly, this section elaborates on Step ④ of the methodology

illustrated in Figure 1.3 when integrated in a hardware environment.

For the hardware-based solution of the proposed ML-LVA, we propose the methodology shown in Figure 5.1. Steps ① to ④ shown in this figure remain the same as in the software implementation. For instance, a load instruction that was determined to be "safe-to-approximate" would still be considered as such in a hardware-based implementation. Similarly, an ML predictor that delivers an acceptable quality would uphold the same quality when deployed in a hardware solution. Subsequently, the key differences in delivering the hardware-based implementation are the tasks highlighted in Steps ⑤ and ⑥. Thus, we limit the explanation in this section to the key elements that have not been previously discussed, i.e., Steps ⑤ and ⑥ shown in Figure 5.1.

In Step ⑤, we implement the ML-based Load Predictor within the hardware domain. A pragmatic and computationally efficient solution is achieved through the use of a lookup table. This approach is particularly suitable for the ML-LVA model

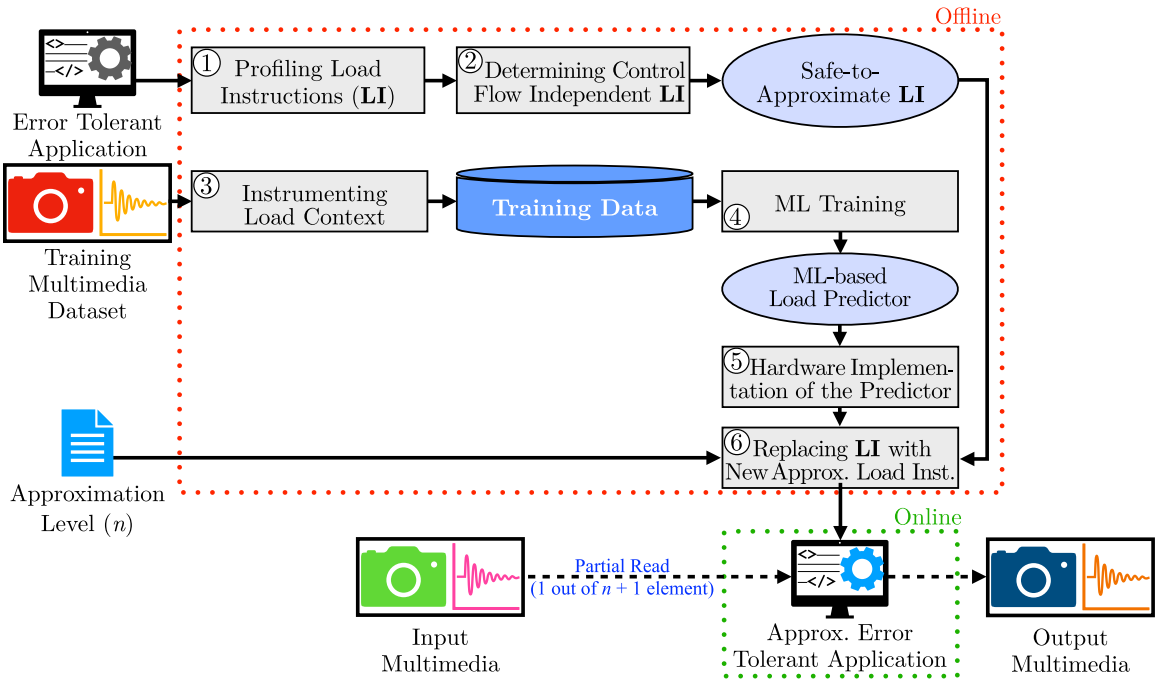


Figure 5.1: Methodology to Implement the Proposed LVA in Hardware

developed in this thesis, as it is designed to predict one byte per load operation. Accordingly, the complete predictor can be encapsulated in a lookup structure comprising 256 entries, with each entry occupying a single byte. Thus, the total memory footprint required for the predictor is merely 256 bytes. To store this table, a Read-Only Memory (ROM) structure is selected due to its minimal area and power overhead relative to more complex alternatives such as Static Random Access Memory (SRAM). Furthermore, the immutable nature of ROM enhances security by safeguarding against unauthorized modifications, such as those introduced by malware aiming to manipulate prediction outcomes for malicious purposes. Nonetheless, a ROM presents a significant limitation: its contents are fixed post-fabrication, thereby precluding updates or reprogramming. To address this constraint, an Electrically Erasable Programmable Read-Only Memory (EEPROM) can be utilized as a more flexible alternative. EEPROM offers a favorable trade-off between hardware simplicity, resilience to tampering, and the ability to update the predictor post-deployment, thereby extending the hardware's applicability to a broader range of workloads and future enhancements.

Following the integration of the predictor into hardware via a lookup table, Step ⑥ involves modifying the processor microarchitecture to enable a seamless communication between the software layer and the newly instantiated hardware predictor. This necessitates augmenting the processor's Instruction Set Architecture (ISA) to include custom instructions capable of invoking the ML-LVA functionality. Such modifications are feasible in many processor designs, particularly those based on extensible ISAs. A notable example is the RISC-V architecture [28], which explicitly reserves certain opcode spaces—specifically the *Custom-0* and *Custom-1* instruction groups—for user-defined extensions [63]. This architectural feature permits the addition of bespoke

instructions without interfering with existing ISA semantics. Once these custom instructions have been integrated into the ISA, the application code is revised such that load instructions previously marked as "safe-to-approximate" are substituted with the newly added load value prediction instructions. During the application runtime, these instructions trigger the predictor to estimate load values using the precomputed lookup table, thereby eliminating the need for conventional memory access and reducing memory latency. This marks a fundamental shift in execution behavior, with the processor relying on predictive computation in place of deterministic memory retrieval for selected operations.

To enable a seamless interaction with the hardware-embedded ML-LVA predictor, two new R-type custom instructions have been introduced: *AxAU* (Approximate Audio Load) and *AxIM* (Approximate Image Load). These instructions are specifically crafted to invoke the ML-LVA predictor directly within the processor pipeline, facilitating efficient prediction of load values without engaging in conventional memory access. Although the R-type instruction format traditionally requires two source operands along with a destination register, the semantics of these new instructions diverge intentionally from this norm to better suit the operational characteristics of the ML-LVA. In both *AxAU* and *AxIM*, only the first source operand is meaningful which represents the "history value" or previously loaded data. This value serves as the index into a dedicated ROM. Notably, *AxAU* and *AxIM* access separate ROMs, one specifically allocated for audio prediction data and the other for image prediction data, in order to deploy the two ML-LVA developed in Chapter 3. Additionally, the second source operand, while still encoded in the instruction to preserve structural compatibility with the R-type format, is effectively ignored during execution and does not influence the instruction's behavior.

When executed, the instruction uses the first operand to access the corresponding entry in the appropriate ROM, retrieving the predicted approximation generated by the ML-LVA model. This predicted value is then written directly into the destination register, effectively substituting the traditional load operation. By leveraging a low-latency ROM lookup instead of accessing the main memory, these instructions substantially reduce load latency and improve execution throughput. By adhering to the RISC-V custom instruction specification, particularly leveraging the reserved opcode spaces for user-defined extensions, these instructions maintain broad portability across RISC-V-based cores that support ISA customization. This strategic alignment enhances the scalability and applicability of the ML-LVA accelerator, allowing future designs to incorporate these instructions with minimal modification while preserving the benefits of load value approximation in latency-critical workloads such as real-time image and audio processing.

5.3 Hardware Implementation

To accurately evaluate the performance impact of the hardware implementation of the ML-LVA, a detailed integration and a realistic hardware simulation environment were required. This section outlines the processor core into which the ML-LVA was integrated, as well as the testing infrastructure used for performance analysis, including the surrounding memory hierarchy and external DRAM model that enabled realistic system-level evaluation. Each component was selected or developed to reflect practical design constraints and to enable cycle-accurate simulation of the full system. In this thesis, the hardware in which the ML-LVA was integrated consists of a RISC-V processor called CVA6 [27] which has L1 cache, an L2 cache and a DRAM as shown in Figure 5.2. All components are connected via the Advanced Microcontroller Bus

Architecture (AMBA) Advanced eXtensible Interface version 4 (AXI4) protocol [64]. In the rest of this section, we will present the details of each of these components and the reasoning for their selection.

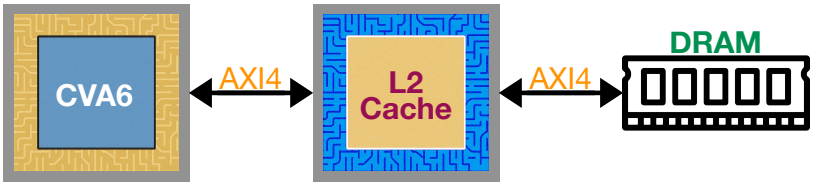


Figure 5.2: Hardware Implementation Environment

5.3.1 CVA6 Processor

At the core of the testing environment is the CVA6 processor, a 64-bit in-order RISC-V core [27]. The CVA6 features a six-stage pipeline architecture, as illustrated in Figure 5.3. The CVA6 supports in-order instruction issue, out-of-order execution and write-back, and an in-order commit stage, and thus preserving the original execution order of the program. The core of the CVA6 implements the Integer (I), Multiplication/Division (M), Atomic (A), and Compressed (C) extensions, as defined in [63], along with [65]. Additionally, the CVA6 supports three privilege levels—Machine (M), Supervisor (S), and User (U)—enabling compatibility with Unix-based operating systems. It incorporates several advanced features, including a configurable microarchitecture, dedicated translation lookaside buffers (TLBs), a hardware page table walker, and branch prediction mechanisms such as a branch target buffer and a branch history table.

A notable architectural feature of the CVA6 is its decoupled frontend pipeline. In this design, the instruction fetch and decode stages operate independently of the backend execution stages. This decoupling allows the frontend to continue fetching and decoding instructions even when the backend is stalled, thereby improving instruction

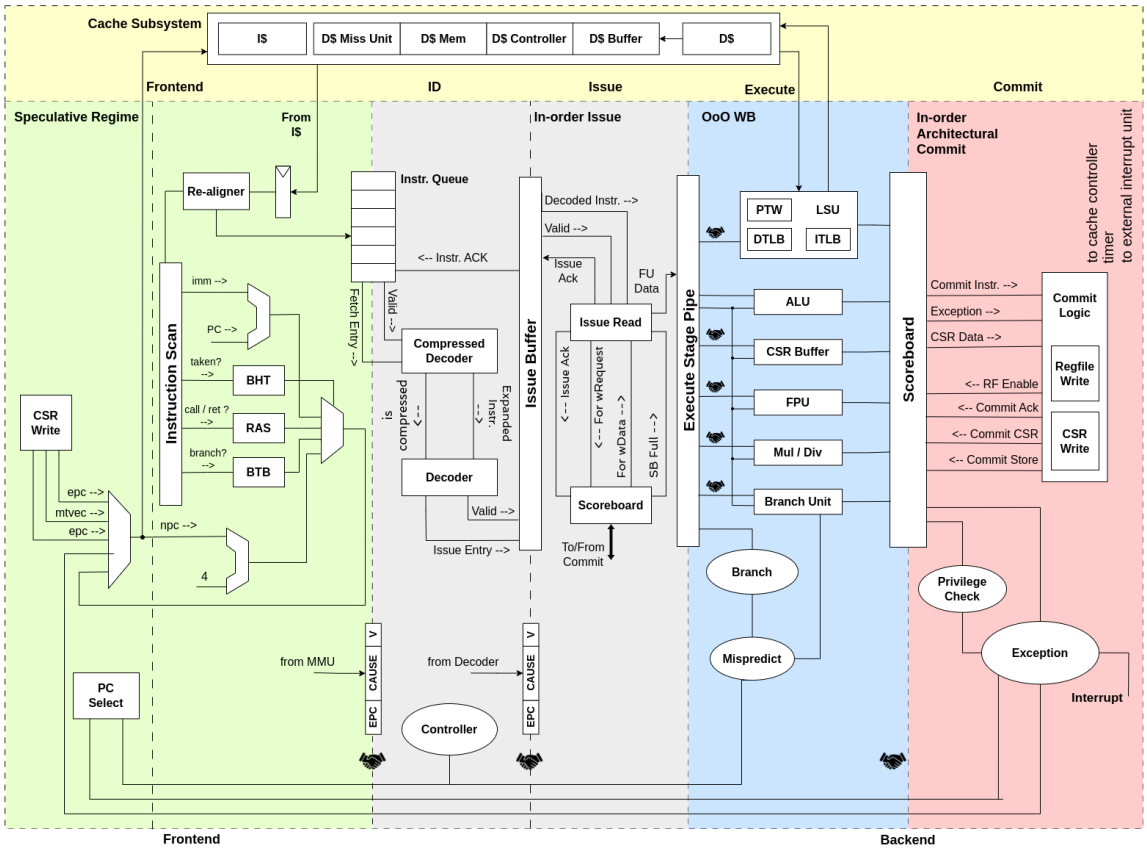


Figure 5.3: Architecture of the CVA6 [27]

throughput and overall pipeline efficiency. By buffering instructions between the frontend and backend, this architecture helps mitigate memory latency and enhances the performance of branch prediction and instruction prefetching.

The CVA6 also includes separate Level-1 (L1) instruction and data caches, both of which offer configurable associativity and replacement policies. Furthermore, it supports the AXI4 protocol [64] for memory and peripheral interfacing, enabling seamless integration with second-level (L2) caches. The processor also provides infrastructure for integrating tightly coupled accelerators via custom instruction support. This feature permits the definition of new opcodes that directly interface with user-defined hardware modules. We chose the CVA6 in this thesis due to its

modular and extensible architecture, open-source availability, and robust support for custom instructions. These characteristics make it an ideal platform for integrating hardware accelerators. Additionally, its support for modern microarchitectural features, such as out-of-order execution and advanced branch prediction, further enhances its suitability for architectural exploration and the implementation of cutting-edge processor techniques. The primary reason for selecting the CVA6 processor is its open-source nature, which ensures that its design and implementation are publicly accessible. This transparency allows unrestricted access to the core's architectural details, enabling comprehensive study and use without licensing constraints. In contrast, widely used architectures such as ARM and x86 are proprietary and closed-source. Their designs are not made publicly available, limiting the ability to examine or modify them freely. Therefore, in the hardware-based implementation we switch from the x86 architecture to the CVA6 RISC-V processor.

To preserve a conventional execution model, the custom instruction implementing the ML-LVA was integrated into the CVA6 using its accelerator extension interface. In this integration, the accelerator does not operate as an independent co-processor with a separate fetch mechanism, private registers, or parallel data-management capabilities. Instead, it is tightly coupled within the Execute stage of the pipeline, reusing existing datapath resources of CVA6. This design results in a simpler datapath with fewer conflicts between parallel paths, while also reducing the verification complexity. By avoiding the need for advanced features such as dual-fetch mechanisms or cache-coherence protocols typically required for external accelerators, the proposed integration maintains consistency with the baseline processor pipeline [66]. Accordingly, we modified the source codes of "*cva6_accel_first_pass_decoder_stub*" [67] and "*acc_dispatcher*" [68] blocks. Subsequently, the modified codes affect the Instruction

Decode (ID) and Execute stages shown in Figure 5.3. Furthermore, the decoder is updated to support two new custom instructions for predicting load values in audio and image applications, respectively. These instructions are encoded as R-type instructions using the "ACCEL" opcode as defined in the *ariane* package [69], with the *funct3* field set to 3'b000 for audio and 3'b001 for image predictions. Subsequently, the code of the "*cva6_accel_first_pass_decoder_stub*" is extended to decode these newly added instructions. In parallel, the logic for the predictor (ML-LVA ROM and its indexing mechanism) was added to the "*acc_dispatcher*" block, where the predicted values are produced in a single clock cycle. Additionally, we set the parameters of L1 caches to: *i*) 4-way set associative organization, *ii*) cache lines of 64 bytes each, and *iii*) total size of 65,536 Bytes. This cache configuration was chosen to simulate the L1 cache of the ARM Cortex-A720 [70]. The data cache was configured as an OpenPiton cache [71] with a write through policy. Finally, the simulation environment for the modified CVA6 was developed in SystemVerilog [72] to run the custom chip at 3 GHz, providing a robust framework for functional validation and waveform inspection during the design and integration of the ML-LVA accelerator.

5.3.2 AXI Last Level Cache

To complement the memory hierarchy of the CVA6-based testing platform, the AXI Last Level Cache (LLC) [73] is employed as a unified Level-2 (L2) cache. The AXI LLC is a configurable, open-source developed within the Parallel Ultra-Low Power (PULP) platform [74]. It is designed to interface seamlessly with AXI4-compliant masters and slaves, making it highly suitable for integration into RISC-V-based architectures, specifically, the CVA6 adopted in this thesis.

The AXI LLC functions as a shared L2 cache that sits between the private L1 data

and instruction caches of the CVA6 and the off-chip memory system. By providing a high-bandwidth, low-latency intermediary, the LLC significantly reduces the frequency of expensive memory accesses to external DRAM, thereby improving both application level performance and energy efficiency. The cache supports multiple configurable parameters, including cache size and associativity, allowing for tailored optimization depending on system-level requirements and workload characteristics.

Architecturally, the AXI LLC is designed to operate under a write-back cache policy, meaning that modified cache lines are only written back to main memory when they are evicted from the cache. This approach reduces the frequency of write operations to memory, thereby minimizing memory traffic and enhancing overall system bandwidth. Such a policy is well-suited to systems where reducing latency and conserving memory bandwidth are critical performance goals.

In our hardware implementation, the LLC is integrated into a CVA6-based and configured with specific parameters tailored to balance capacity, associativity, and access granularity. The LLC is configured to replicate the L2 of a recent ARM Cortex A720 processor [70]. We chose to adopt this approach as ARM is a popular RISC architecture that is widely used in the industry and thus serves as a representative reference point for modern RISC-based designs. In alignment with the Cortex-A720, the LLC is implemented as an 8-way set-associative cache. The data width of the LLC, denoted as $DataWidthFull$, is determined by the AXI interface and set to 64 bits. Since the Cortex-A720 uses a 64-byte cache line, the *number of data blocks* (N_{DB}) per line is calculated as $N_{DB} = \frac{64 \text{ bytes}}{64 \text{ bits}} = 8$. To maximize the *number of cache lines* (N_{CL}), we adhere to a constraint imposed by the AXI LLC design [73], which requires that $\log_2(N_{DB})$ does not exceed the width of the AXI len_t signal. Given that len_t is 8 bits wide, this constraint leads us to select $N_{CL} = 256$. Based on this configuration,

the total size of the LLC can be calculated using the following formula:

$$Size\_LLC = SetAssociativity \times N_{CL} \times N_{DB} \times \frac{DataWidthFull}{8} \tag{20}$$

Substituting the appropriate values, we compute the cache size as:

$$Size\_LLC = 8 \times 256 \times 8 \times \frac{64}{8} = 131,072 \tag{21}$$

This results in a total cache size of 131,072 bytes, or 128 KB. The size of this L2 aligns with one of the possible L2 configuration of the ARM Cortex-A720. Such a configuration yields a relatively large and highly associative L2 cache, well-suited for workloads with strong spatial and temporal locality. The high associativity reduces the likelihood of conflict misses, while the presence of multiple blocks per line enhances data reuse across sequential accesses, improving overall cache efficiency. The L2 cache further employs an aggressive prefetch mechanism that anticipates sequential memory accesses, proactively fetching data before it is explicitly requested.On a single miss, the L2 is able to perform a sequential prefetch in burst mode, fetching up to 256 consecutive cache lines, each consisting of 64 bytes, for a total of 16 KB of data. By exploiting spatial locality, this mechanism reduces the number of individual miss penalties and sustains high bandwidth utilization. With the 4-way set associativity of the cache, up to 64 KB of memory can be mapped without conflicts, thereby mitigating contention and improving throughput. Such a configuration is particularly advantageous for multimedia and signal-processing workloads, where spatial locality is of paramount importance and large contiguous memory regions must be accessed efficiently.

From a system integration perspective, the AXI LLC is instantiated as a standalone,

modular component that connects directly to the AXI interconnect without requiring changes to the internal pipeline or memory interface of the CVA6 core. Its parameterizable structure allows for easy adaptation to different performance, area, and power constraints. In the context of PULP-based systems, the LLC offers a scalable and efficient method to extend the memory hierarchy, supporting high-throughput and bandwidth-sensitive applications with minimal design effort.

5.3.3 Micron DDR4 Model

To complete the memory hierarchy of the testing platform, a DRAM memory based on the publicly available DDR4 Verilog simulation model [75], provided by Micron Inc, is connected to the AXI LLC. This memory model serves as the off-chip main memory and provides a realistic behavioral representation of DDR4 memory timing and operation. It supports a configurable range of data rates, spanning from 1066 to 4000 mega transfers per second (MT/s), where each transfer represents the movement of data on both the rising and falling edges of the clock signal, as is characteristic of double data rate (DDR) memory. For the purpose of this evaluation, a data rate of 3200 MT/s—corresponding to a clock cycle time (tCK) of 0.625 nanoseconds—was selected to reflect a high-performance memory configuration. In order to facilitate seamless communication with the rest of the AXI-based system, an AXI-compatible memory interface and DRAM controller were developed and integrated into the testing platform.

The DRAM model itself, obtained from Micron, offers cycle-accurate behavioral modeling of DDR4 memory, including command timing, burst access behavior, bank management, and timing constraint such as Row to Column Delay (tRCD), Row Precharge Time (tRP), and Column Address Strobe (CAS) latency, as defined by

the DDR4 standard [61]. However, the original model is not directly compatible with the AXI protocol, which is used throughout the testing platform developed to test the proposed ML-LVA. To bridge this gap, we design a custom AXI-to-DRAM controller to translate AXI4 memory transactions into appropriate DDR4 command sequences, while adhering to the timing and ordering constraints imposed by the DRAM specification.

The AXI interface operates as a slave connected to the AXI interconnect fabric, accepting read and write transactions from the AXI LLC. Upon receiving a request, the controller schedules and issues the corresponding DDR4 commands, such as activate, read, write, and precharge to the DRAM model. It also handles address translation and manages multiple open row policies across banks to improve memory throughput. This setup ensures accurate timing emulation and provides a representative evaluation of how real DRAM would behave under the memory access patterns generated by the CVA6 processor and its attached accelerators.

From a system-level perspective, the inclusion of the DRAM model allows the testing platform to approximate the latency and bandwidth characteristics of real hardware deployments more closely. It introduces realistic memory delays and access contention scenarios that would not be captured by idealized memory models. This is particularly valuable when evaluating the performance impact of the ML-LVA accelerator, as it provides insights into how memory traffic interacts with cache behavior and custom instruction execution under realistic memory access patterns.

The integration of the DRAM controller and AXI interface was carried out without modifying the existing AXI LLC or CVA6 processor design, thus preserving the modular architecture of the system. This decoupled approach facilitates independent development, testing, and reuse of each subsystem. Overall, the external DRAM

model serves as a critical component to validate the end-to-end memory hierarchy of the platform and supports comprehensive functional and performance evaluation under realistic memory access conditions.

5.4 Experimental Results

This section presents a detailed performance evaluation of the proposed ML-LVA technique using a full-system simulation environment composed of the CVA6 processor, the AXI LLC acting as a L2 cache, and an external DDR4 DRAM model based on Micron's Verilog simulation package. The objective of this analysis is to assess the impact of the proposed ML-LVA on execution performance when integrated into a realistic hardware memory hierarchy.

To explore the applicability of ML-LVA in multimedia processing, four representative applications were selected from both the image and audio domains, namely, image blending [32], image inversion [36], audio blending [33], and audio inversion [37]. These applications were chosen to cover a spectrum of memory access behaviors and computational patterns, providing a meaningful evaluation of ML-LVA's effectiveness across different workloads.

Performance measurements are carried out by quantifying the speedup achieved when employing the ML-LVA compared to a baseline execution without approximation. The evaluation considers two distinct performance levels: overall application speedup, which captures the end-to-end impact on total execution time, and memory load speedup, which focuses specifically on the duration of memory load operations. The analysis spans multiple operating frequencies and approximation levels (n), where each level controls the aggressiveness of value approximation in load instructions. The performance analysis was performed using Siemens Questasim 2024.1 [76].

The structure of this section follows a task-specific organization, grouped into two categories based on processing modality: image and audio. Each category is further divided into two application subtypes: blending and inversion. For each individual application, we analyze the effect of varying the approximate level (n) on the performance. By employing a cycle-accurate simulation platform with realistic memory modeling, this evaluation provides practical insights into the trade-offs introduced by the ML-LVA mechanism. The results underscore the performance benefits and limitations of using ML-based value approximation in a memory-centric architecture and demonstrate the potential of such techniques in accelerating multimedia workloads under modern processor-memory system designs. Thereafter, we compare the performance of the design proposed in this thesis with the state-of-the-art, followed by an analysis of the synthesis results to assess the overhead of the proposed ML-LVA when implemented in hardware.

5.4.1 Image Processing

This subsection presents a performance analysis of image processing tasks executed on the memory processor enhanced with the proposed ML-LVA, in comparison to their execution using a conventional baseline. The tasks under evaluation, i.e., image blending and image inversion, represent distinct categories of operations, each characterized by different levels of computational complexity and memory access behavior.

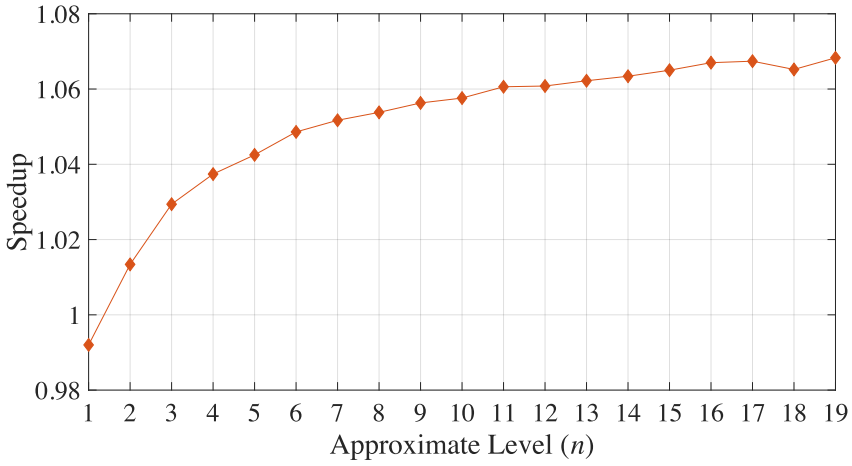
5.4.1.1 Image Blending

The image blending application, which merges two input images by computing a per-pixel weighted average, benefits substantially from ML-LVA-based approximation due

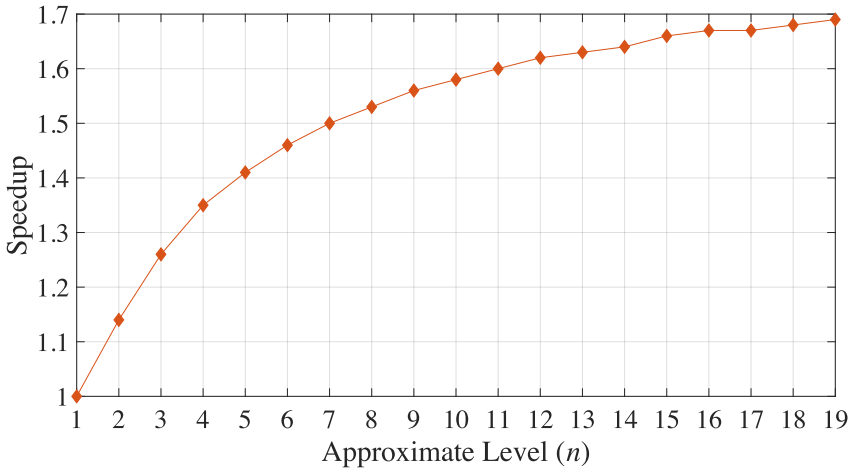
to its consistent memory access patterns. The performance improvements measured at both memory and application levels show a robust upward trend as the approximation aggressiveness increases, i.e., increasing n. The application overall and memory level speedups are shown in Figure 5.4.

From Figure 5.4(a), we notice that for the overall application, the performance trends are more subdued but still meaningful. At the lowest setting of approximate level, i.e., $n = 1$, the speedup is slightly under 1 at 0.992×, indicating a minor overhead, possibly due to hardware instruction routing. When the approximate level increases, the overall speedup rises to 1.013× and reaches 1.029× when $n = 3$. The trend continues with incremental gains, hitting 1.057× and 1.065× for $n = 10$ and $n = 15$, respectively. The highest recorded speedup is 1.068×, i.e., 6.8% speedup, for the highest tested approximate level, i.e., $n = 19$. This relatively slower improvement is expected since memory access acceleration translates into broader application speedup only partially, especially in workloads that are not fully memory-bound. Nonetheless, the positive correlation, across all levels, demonstrates that the ML-LVA leads to a stable and scalable performance advantage even when used as a tightly coupled accelerator within a general-purpose processor.

From Figure 5.4(b), we notice that at the memory level, the speedup begins at a baseline of 1.00× for $n = 1$. Nonetheless, as the approximate level increases to $n = 2$, the average speedup rises to 1.14×. This early gain signals that even limited prediction of load values can significantly reduce memory latency in such structured workloads. As the approximation level increases, this improvement continues almost linearly up to approximate level 10, where the speedup reaches 1.58×. Beyond $n = 10$, the curve begins to flatten, though the gains do not vanish. The speedup continues to rise in a slower pace, reaching 1.66× at approximate level 15 and culminating in



(a)



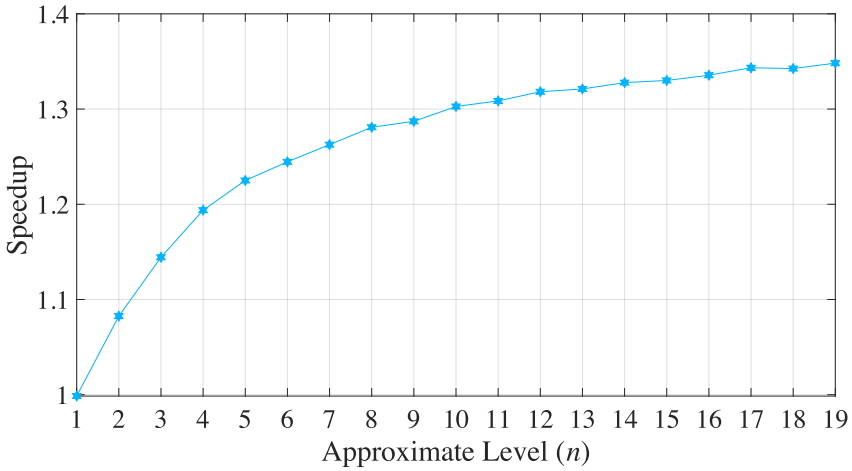
(b)

Figure 5.4: Average Speedups for Image Blending: (a) Overall, and (b) Memory Loads

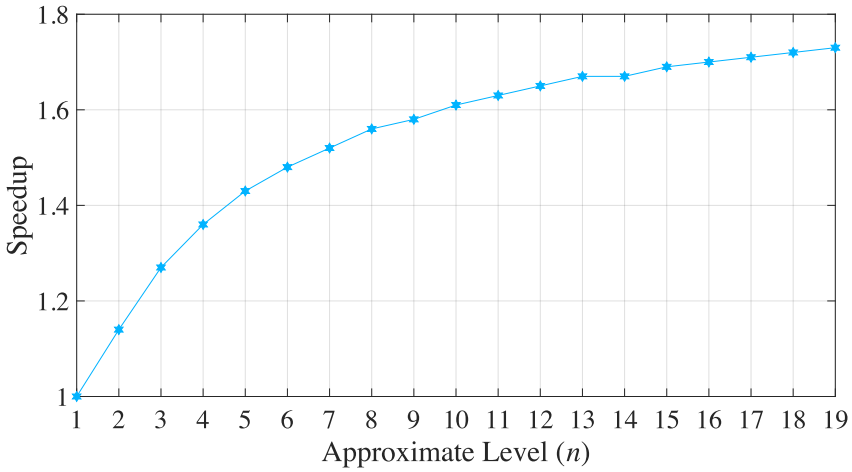
a peak of 1.69× at the highest approximate level, i.e., $n = 19$. The marginal gains beyond $n = 15$, i.e., increase from 66% to 69%, indicate that most of the exploitable redundancy is captured by this point, as the percentage of load that are approximated is calculate as $\frac{n}{n+1}$ which has a flattening pattern as it increases.

5.4.1.2 Image Inversion

The speedups achieved in the image inversion are given in Figure 5.5. For the application overall speedup, we notice from Figure 5.5(a) that the effect of ML-LVA is



(a)



(b)

Figure 5.5: Average Speedups for Image Inversion: (a) Overall, and (b) Memory Loads

also strong. Similar to the image blending, the speedup at approximate level 1 is below 1 and was measured to be 0.999×. As the approximation increases, the performance quickly improves, where we measured a speedup of 1.08×, 1.19× and 1.24× for $n = 2$, $n = 4$ and $n = 6$, respectively. When the approximate level is increased to 10, the speedup reaches 1.30×, with further increments taking it to 1.34× at $n = 16$ and peaking at 1.348× for the highest approximate level.

From the results shown in Figure 5.5(b), we notice that at the memory level the gains are both steep and sustained. Starting from a speedup of 1.00× for $n = 1$ and

rises to 1.14× for $n = 2$. The growth in speedup continues with: 1.27×, 1.43× and 1.56× for the approximate levels 3, 5 and 8, respectively. This consistent acceleration shows that prediction remains effective even as more speculative loads are deployed. Unlike image blending, where speedup gains tapered off around approximate level 15, image inversion continues to benefit across all tested levels. At approximation level 14, the speedup reaches 1.67×, and achieves a maximum speedup of 1.73× for $n = 19$.

Interestingly, while the image blending's overall speedup flattens early, image inversion continues to show small but steady gains well into the higher approximation levels. This sustained growth suggests that image inversion is more tightly bound by memory latency, and therefore more responsive to approximative memory load acceleration. This trend of deviation in the flattening of the overall speedup results between blending and inversion tasks was also observed in software-based implementation of the ML-LVA.

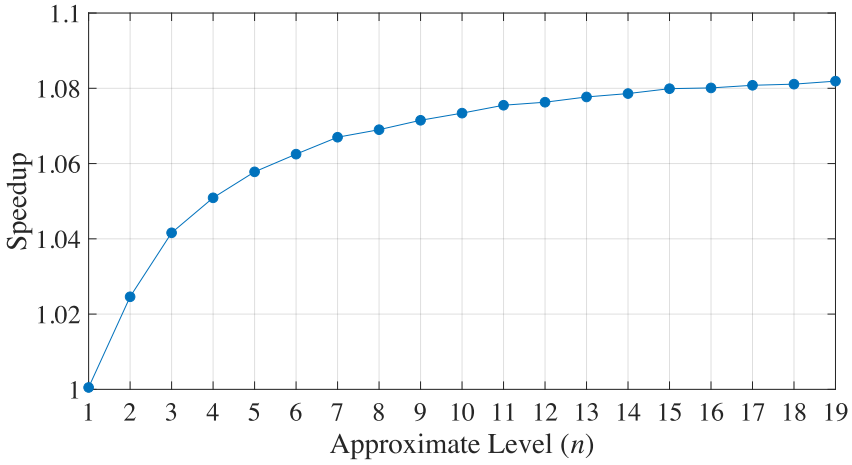
5.4.2 Audio Processing

This subsection presents the performance evaluation of audio processing tasks executed on the hardware platform integrating the proposed ML-LVA accelerator. In contrast to image processing, which operates on spatial data, audio processing deals with time-continuous signals typically handled in discrete frames or windows. This framing makes throughput and latency especially critical for maintaining real-time performance. Furthermore, audio workloads differ in their computational characteristics, with varying levels of arithmetic intensity and control flow complexity, which influences how effectively they benefit from acceleration near memory. The evaluation focuses on the observed speedups measured when executing audio tasks with the ML-LVA. We analyze how different classes of audio operations respond to hardware-level load value

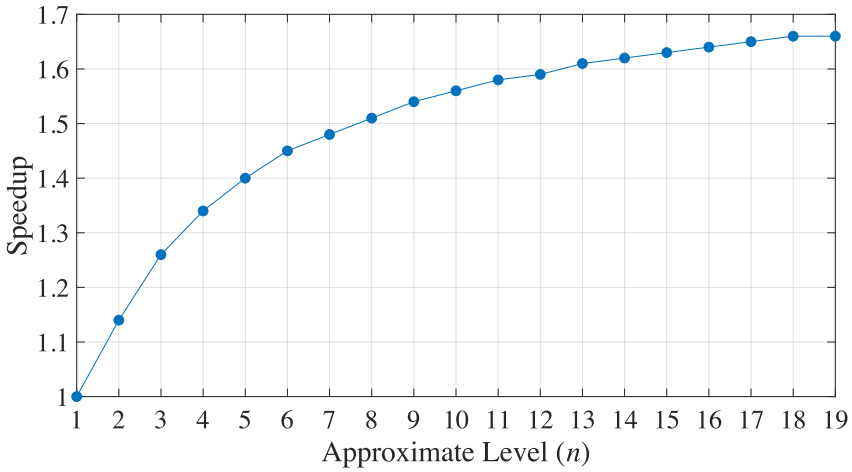
approximation and identify the greatest performance gains. These results provide insight into the suitability of the adapted CVA6 to integrate the ML-LVA in its architecture when accelerating audio tasks.

5.4.2.1 Audio Blending

The speedup results of the audio blending, depicted in Figure 5.6, show an increased gain as the approximate level n increases. From Figure 5.6(a), we notice that for the overall application, the improvements are more modest, reflecting the fact that not all



(a)



(b)

Figure 5.6: Average Speedups for Audio Blending: (a) Overall, and (b) Memory Loads

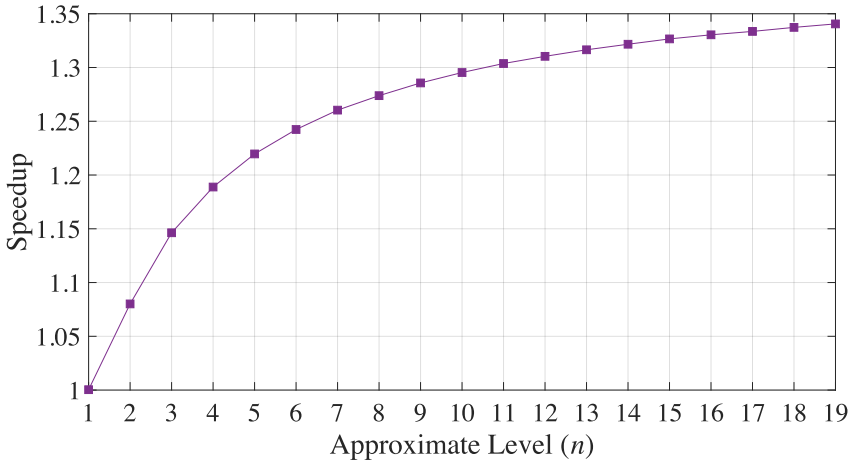
parts of the audio blending process are equally memory-bound. Starting from 1.0005× at approximate level of 1, the speedup reaches 1.050× when $n = 4$ and continues upward in small but consistent steps. By the approximate level 10, the overall speedup is 1.073×, and it gradually climbs to 1.0799× when the approximate level reaches 15. The final speedup at level 19 is 1.0819×. The tight clustering of these values in the later stages demonstrates the onset of saturation, where memory latency is no longer the principal bottleneck. Still, the gain of over 8% in execution time at high approximation levels is significant for embedded or real-time systems, where energy or throughput constraints are tight. The results confirm that audio blending is a strong candidate for approximate memory techniques, delivering high memory load gains and consistent application level improvements.

At the memory level, we notice from Figure 5.6(b) that the benefits of approximation are immediate and substantial. The speedup grows from the baseline of 1.00× for $n = 1$ to 1.14× when $n = 2$ and reaches 1.34× when n is increased to 4. The steepness of this growth continues through approximate levels 5 to 10, reaching a speedup of 1.56×. Beyond level 10, the growth becomes more incremental, yet it remains steady. The maximum speedup achieved is 1.66× at level 18, with level 19 maintaining this value. This consistency suggests that even at aggressive approximation levels, the ML-LVA performs reliably.

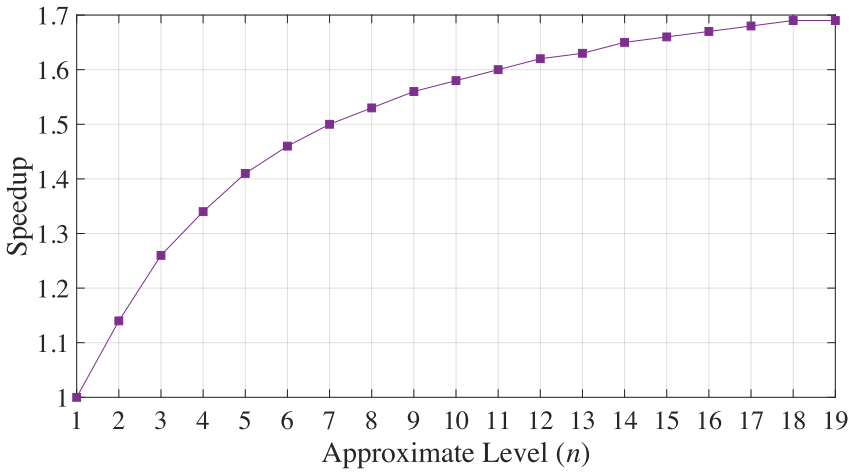
5.4.2.2 Audio Inversion

The ML-LVA delivers solid gains when tested in audio inversion as shown in Figure 5.7. The results of application overall speedup shown in Figure 5.7(a), reflects a more striking improvements. Beginning from a near-unity value of 1.0003× at approximate level 1, the speedup surges to 1.15× and 1.22× for the approximate levels 3 and 5,

respectively. This rapid climb indicates a high dependence on memory performance. The growth continues with 30% and 33%, i.e., 1.30× and 1.33×, at levels 10 and 15, respectively. At level 19, the maximum overall speedup is 1.34×. With a 34% improvement in the overall performance, the hardware-based implementation of the ML-LVA achieved a notable result demonstrating its practicality. These results are among the best observed in the study, suggesting that the ML-LVA not only accelerates memory operations but also significantly reduces the execution time of the entire application. The monotonic increase across all approximation levels indicates that the



(a)



(b)

Figure 5.7: Average Speedups for Audio Inversion: (a) Overall, and (b) Memory Loads

load values in audio inversion, remain within a predictably learnable range for the model.

From Figure 5.7(b), we notice that at the memory level, the speedup begins at 1.00×, rising to 1.26× at level 3 and 1.41× at level 5. The increase continues smoothly with speedups of 1.56× and 1.63× for approximate levels of 9 and 13, respectively. By approximation level 19, the memory speedup peaks at 1.69×. The absence of slowed gain at extreme approximation levels suggests that the ML-LVA manages to maintain improved performance when deployed in image inversion for the various approximate levels tested.

5.4.3 Comparison with Related Work

We compare the hardware implementation of the ML-LVA with the state-of-the-art LVA proposed in [19]. We omit the comparison with the LVA proposed in [24] since their work targets a GPU while the one we propose in this thesis aim to approximate the load value in a CPU. Table 5.1 shows the average speedup achieved for the various approximate levels among the various applications. From Table 5.1, we can notice that the LVA proposed in [19] deliver a higher speedup for a 50% approximation, i.e., $n = 1$. However, at approximate level 3 and higher, i.e., more than 75% approximation, the proposed LVA outperforms the state-of-the-art, where our model delivered an increased

Table 5.1: Speedup Comparison of the Proposed Hardware-based ML-LVA with [19]

Approximate Level (n)	**LVA** [19]	**Proposed ML-LVA**
1	1.08	1.00
3	1.07	1.09
5	1.08	1.14
9	1.08	1.18
17	1.08	1.21

speedup when n increases, while the one proposed in [19] delivered a constant speedup. Subsequently, we conclude that the hardware-based ML-LVA also outperforms the LVA proposed in [19].

5.4.4 Overhead Measures

In order to analyze the resource usage overhead of incorporating the ML-LVA in the CVA6, we synthesized the original CVA6 and the version with the ML-LVA using *Cadence Innovus* [77]. The synthesis is performed using a Cadence Generic Process Design Kit (GPDK) based on the 45nm CMOS technology node. The results of the synthesis are summarized in Table 5.2. From these results, we can notice that the area and power increases in rate of 5.09% and 0.79%, respectively, when the ML-LVA is added to the CVA6. Nonetheless, this is expected since an additional hardware was added to the processor. However, with a speedup in memory load value surpassing 70% in multiple cases, the measured overhead can be deemed acceptable.

Table 5.2: Synthesis Results of the CVA6

Metric	**CVA6**	**CVA6 w/ ML-LVA**	**Increase**
Area (μm^2)	167,986.64	176,529.11	5.09%
Power (mW)	344.15	346.87	0.79%

5.5 Summary

This chapter presented the hardware implementation and evaluation of the ML-LVA within the CVA6, a RISC-V processor. The ML-LVA was integrated into the processor pipeline as an accelerator. The resulting hardware was synthesized using Cadence Innovus with a 45nm GPDK CMOS process. To ensure realism, the implementation

was tested under practical execution conditions using a CPU clocked at 3GHz and paired with a 3200MT/s DDR4 memory subsystem. The focus was on media processing workloads involving image and audio applications.

The implementation tackled several key challenges, including minimizing resource overhead, maintaining timing closure, and integrating seamlessly with the processor's out-of-order pipeline. CVA6 was selected as the host platform due to its modular, open-source architecture, which facilitated straightforward hardware augmentation. The ML-LVA employed a lightweight predictor realized as a lookup table, stored in ROM. To enable efficient invocation of the ML-LVA, two custom RISC-V R-type instructions called *AxAU* and *AxIM* were introduced, specifically tailored to image and audio load prediction. Support for these custom instructions required modifications to the instruction decoder and execution pipeline of the CVA6. Each instruction accesses a dedicated ROM storing prediction tables for its respective data type.

The memory subsystem used to analyze the performance of the proposed implementation featured a configurable hierarchy with L1 and L2 caches which are modeled after an ARM Cortex-A720 design to mirror modern embedded processors. Additionally, the memory subsystem was complemented with a DDR4 DRAM to provide a complete memory hierarchy. An AXI-to-DDR4 to bride the connection between the Micron DDR4 DRAM and the L2 cache. Subsequently, the hardware implementation was tested in a cycle-accurate simulation environment using SystemVerilog and Questasim.

The performance evaluation demonstrated that the ML-LVA delivers notable speedups across a range of media applications. Speedup values reached up to 1.08× at the application level and up to 1.73× in memory operations, confirming the ability of the implemented ML-LVA to alleviate memory bottlenecks. The most significant gains

were observed in the audio inversion workload. Importantly, the ML-LVA maintained and even increased its performance benefits as approximation levels rose. This marks a significant improvement over the existing LVA such proposed in [19], which tend to plateau even at modest levels of approximation.

The synthesis results confirm that these benefits come at an acceptable cost. The ML-LVA introduced just 5.09% area and 0.79% power overhead. Given the substantial improvements in memory latency—exceeding 70% in certain cases—these overheads are a justified trade-off. The hardware prototype thus validates the practicality of integrating machine learning-based speculative approximation into modern processors.

In summary, the ML-LVA offers a robust, efficient, and scalable approach to load value approximation near memory. Its strong performance, especially at high approximation levels, and modest resource footprint position it as a compelling enhancement for future processors aiming to reduce memory access latency and improve throughput in energy-constrained environments.

Chapter 6

Conclusion and Future Work

6.1 Conclusion

The widening disparity between processor speeds and memory latency, commonly referred to as the *memory wall*, continues to constrain the performance of modern computing systems. In this thesis, we have addressed this fundamental bottleneck by proposing a Machine Learning-based Load Value Approximator (ML-LVA), a novel approach that leverages the principles of approximate computing to intelligently speculate memory load values with minimal overhead. By shifting from conventional memory-bound execution models to speculative, error-tolerant alternatives, the ML-LVA presents a compelling solution for accelerating data-intensive applications without sacrificing output quality.

The proposed methodology to develop the ML-LVA comprises two main phases: an Offline phase and an Online phase. During the Offline phase, the target application is profiled to identify load instructions that can tolerate approximation without compromising correctness. Concurrently, training data is preprocessed and partitioned to train and validate an ML model, which is subsequently integrated into the

application. The approximation level, defined by a user-specified parameter, determines the proportion of load instructions replaced by approximate predictions. The Online phase corresponds to the runtime execution of the approximated application producing outputs. The ML-LVA framework supports both software- and hardware-based implementations. The software approach realizes load value prediction as a function call to a subroutine, enabling deployment on existing hardware platforms without architectural modifications. Conversely, the hardware implementation incorporates a dedicated accelerator within the processor pipeline, triggered via a custom instruction, to minimize latency and overhead. This dual-mode design ensures flexibility for deployment across a wide range of systems, from commodity hardware to custom processor architectures.

A detailed analysis of quality of the proposed ML-LVA was performed across six representative audio and image processing tasks. These tasks include multiplication-based blending, inversion, and binarization operations. Quality evaluation employed established metrics including Peak Signal-to-Noise Ratio (PSNR), Normalized Mean Absolute Error (NMAE), Normalized Root Mean Squared Error (NRMSE), as well as accuracy and precision for classification-oriented tasks. Across six real-world multimedia applications, namely, image and audio blending, inversion, and binarization, the ML-LVA consistently demonstrated adaptability, robustness, and performance scalability. Notably, applications exhibiting temporal or spatial locality, such as audio inversion, maintained PSNR values above 33.11 dB even at high speculation rates, affirming their resilience to approximation delivered by the trained ML-LVA. The static nature of the predictor, where runtime training is eliminated, makes it attractive for both general-purpose and resource-constrained platforms. When compared to existing techniques such as dynamic value predictors and rollback-free speculation,

the ML-LVA offered compelling advantages, delivering up to 3.75× lower normalized root mean squared error (NRMSE) and 1.98× higher performance, thereby achieving a well-calibrated balance between speed and quality.

The proposed technique was comprehensively validated through both software and hardware implementations. On the software side, the ML-LVA was integrated into conventional CPU-based workflows as a subroutine, yielding substantial performance enhancements across a suite of multimedia benchmarks. In tasks such as image inversion and audio binarization, speedups in memory operations reached up to 6.77×, while the speedup in the application peaked at 2.45× under high approximation levels. Crucially, these gains were achieved with minimal quality degradation, as evidenced by PSNR values consistently exceeding 86 dB at 95% approximation. The performance benefits scaled with processor frequency, highlighting the relevance of the ML-LVA in latency-bound environments.

In the hardware implementation, the ML-LVA was deployed as an accelerator in the form of a static ROM-based predictor within a RISC-V processor, CVA6. Despite its minimal hardware footprint requiring just 256 bytes of storage and completing predictions within a single cycle, the accelerator delivered up to 1.73× reduction in memory access latency and 1.34× increase in application throughput in representative workloads such as audio inversion. This integration not only underscores the practicality of the ML-LVA in modern out-of-order processor pipelines, but also demonstrates its compatibility with emerging microarchitectures.

The proposed Machine Learning-based Load Value Approximator (ML-LVA) demonstrates clear superiority over state-of-the-art LVAs from [19] and [24]. Compared to the LVA in [19], the proposed ML-LVA achieves more than a twofold reduction in error across various approximation levels, while delivering substantially higher speedups

on CPU-based systems—for instance, an average speedup of 1.98× at approximation level $n = 17$ compared to 1.08× reported in [19]. Although [24] targets GPUs while the proposed ML-LVA targets CPU architectures, we are able to fairly compare the quality of the two models since the quality is irrelevant of the underlying architecture. The proposed ML-LVA attained at least 3.75× better prediction quality across all approximation levels, confirming its effectiveness.

Hardware synthesis on a 45nm CMOS node using Cadence Innovus reveals only modest overheads—5.09% area and 0.79% power increase—when integrating ML-LVA into the CVA6 processor, which are justified by memory load value prediction speedups exceeding 70% in several cases. Overall, these results establish ML-LVA as a high-quality, efficient load value predictor with practical applicability for accelerating CPU-centric workloads.

6.2 Future Work

While this thesis has demonstrated the viability and benefits of the ML-LVA in multimedia processing pipelines, its broader implications extend to a wide spectrum of approximate computing use cases. Future research can build upon this foundation in several directions. A promising extension involves applying the ML-LVA framework to domains where value locality hold such as wireless sensor networks and edge AI—contexts where controlled approximation is both acceptable and advantageous. For example, speculative reads in Internet of Things (IoT) environments could significantly reduce memory bottlenecks and power consumption without impairing application correctness.

Another rich area for exploration involves increasing the adaptability of the ML-LVA. Although the current predictor benefits from its static, low-overhead design,

incorporating elements of runtime adaptivity, such as periodic retraining, online fine-tuning, or lightweight reinforcement learning, could make the system more responsive to non-stationary workloads. Hybrid prediction schemes that blend static learning with dynamic calibration may yield better performance-quality trade-offs in real-world deployments. Furthermore, closer integration with the memory hierarchy via coupling with prefetchers, compression algorithms, Approximate Memory or Processing-in-Memory (PIM) subsystems could enable more synergistic optimizations, effectively creating a layered approximation stack that addresses latency, bandwidth, and energy constraints in concert.

Since the ML-LVA loads one value and subsequently predicts the following n values, a prefetcher could be made approximation-aware by recognizing the approximation level (n). This would allow the cache to selectively fetch only the values that are explicitly required, while omitting those that are to be predicted by the ML-LVA. Such an approach would reduce unnecessary cache line fills, lower memory bandwidth consumption and improve energy efficiency. Moreover, a cooperative design between the ML-LVA and the prefetcher could provide a more fine-grained control of memory traffic, ensuring that approximation not only accelerates execution but also optimizes the utilization of the memory hierarchy, while also providing higher speedup when the two techniques are combined.

Integrating ML-LVA with Approximate Memory enables a layered approximation approach, where the predictor and memory-level approximation can work together to optimize system behavior. Among the key advantages, this combination can amplify performance and energy benefits by reducing memory access latency, lowering bandwidth usage, and decreasing energy consumption. Additionally, it allows for fine-tuning of approximation: the ML-LVA can selectively avoid predictions in cases where

the Approximate Memory has already provided sufficiently accurate results, thereby preserving the best quality from either mechanism. At the same time, this integration introduces several challenges. The system becomes more delicate, as errors in one layer can influence the other, requiring careful calibration and monitoring. Approximation errors may also compound across layers, potentially degrading computational accuracy if not controlled. Moreover, integrating the two approaches increases the complexity of the system, as additional hardware or logic may be needed to track quality, limit cumulative errors, and ensure that multiple approximation layers do not lead to excessive degradation of application-level correctness.

Exploring the proposed software-based implementation of the ML-LVA on RISC-V and ARM architectures rather x86 architecture developed in this thesis, presents a compelling opportunity for several reasons. Both RISC-V and ARM architectures are increasingly prominent in embedded, mobile, and edge computing domains, where power efficiency and customized hardware-software co-design are critical. Investigating ML-LVA on these platforms could reveal unique interactions between the LVA and the more streamlined, energy-aware instruction sets. Exploring ML-LVA on RISC-V and ARM architectures could unlock new avenues for performance and efficiency gains that are not as readily accessible on traditional x86 platforms, ultimately broadening the applicability and impact of the proposed ML-LVA.

Extending the ML-LVA to support larger data widths could enable its deployment across a broader range of application domains. From a hardware perspective, the primary challenge lies in the exponential growth of the lookup table, which significantly increases storage requirements and access complexity. To mitigate this overhead, techniques such as pipelined ROM

lookups [78] combined with approximate arithmetic can be employed, reducing implementation complexity while preserving overall performance.

The identification of safe-to-approximate load instructions is performed manually in this thesis. While effective, this approach is time-consuming and does not scale well to complex applications. An artificial intelligence-based method could automate this process by analyzing program behavior and memory access patterns to identify instructions that can be safely approximated. This detection could be further improved if applied in real time, dynamically selecting load instructions that both create memory bottlenecks and tolerate approximation. However, integrating artificial intelligence has drawbacks. The accuracy depends on training data and model quality, and misclassification of critical instructions could cause fatal errors. Safeguards are therefore necessary, such as maintaining fallbacks to exact execution. Additionally the model has to be lightweight so its overhead does not outweigh performance gains achieved by the usage of the proposed ML-LVA.

In addition, the development of runtime control mechanisms to dynamically modulate the approximation level n in response to workload characteristics or user-defined quality thresholds offers another valuable avenue. In this thesis, we specifically explored a scheme in which one exact load is followed by n approximated loads. An alternative direction would be to investigate different ratios of exact to approximated values fetched from memory, e.g., two exact load followed by n approximation. Such systems could, for instance, escalate speculation during high-memory-pressure phases or reduce it when critical computations demand higher fidelity, thereby ensuring optimal trade-offs at runtime. Nevertheless, these techniques must be designed with care, as an imprudent choice of the number of exact loads or approximation level could diminish the bandwidth savings that approximation is meant to achieve.

Lastly, as with all approximation-based methods, security and reliability warrant careful attention. While the static, deterministic nature of ML-LVA provides some resistance to adversarial manipulation, future iterations should incorporate advanced verification techniques to prevent error propagation and ensure robust behavior in safety-critical contexts. Extending the ML-LVA to embedded real-time systems—such as automotive controllers or medical devices—will necessitate guarantees around worst-case latency, bounded error rates, and fault tolerance, all of which represent meaningful future work.

References

[1] J. L. Hennessy and D. A. Patterson, *Computer Architecture: A quantitative approach.* Morgan Kaufmann Publishers, 2019.

[2] W. A. Wulf and S. McKee, "Hitting the memory wall: Implications of the obvious," *ACM Computer Architecture News*, 1995.

[3] A. Gholami, Z. Yao, S. Kim, C. Hooper, M. W. Mahoney, and K. Keutzer, " AI and Memory Wall ," *IEEE Micro*, vol. 44, no. 03, pp. 33–39, 2024.

[4] Intel, *P6 Family of Processors: Hardware Developer's Manual*, 1998.

[5] K. Diefendorff, "K7 Challenges Intel: New AMD Processor Could Beat Intel's Katmai," *Microprocessor Report*, vol. 12, no. 14, pp. 1–7, 1998.

[6] S. Eyerman and L. Eeckhout, "Probabilistic modeling for job symbiosis scheduling on SMT processors," *ACM Transactions on Architecture and Code Optimization*, vol. 9, no. 2, pp. 1–27, 2012.

[7] M. H. Lipasti, C. B. Wilkerson, and J. P. Shen, "Value locality and load value prediction," *SIGPLAN Notices*, vol. 31, no. 9, p. 138–147, September 1996.

[8] AMD, "AMD Launches AMD Ryzen 5000 Series Desktop Processors: The Fastest Gaming CPUs in the World," 2020.

[Online]. Available: https://www.amd.com/en/newsroom/press-releases/2020-10-8-amd-launches-amd-ryzen-5000-series-desktop-process.html

[9] E. Alpaydin, *Introduction to Machine Learning*. MIT Press, 2020.

[10] G. O. Ganfure, C.-F. Wu, Y.-H. Chang, and W.-K. Shih, "Deepprefetcher: A deep learning framework for data prefetching in flash storage devices," *IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems*, vol. 39, no. 11, pp. 3311–3322, 2020.

[11] V. Seshadri, S. Yedkar, H. Xin, O. Mutlu, P. B. Gibbons, M. A. Kozuch, and T. C. Mowry, "Mitigating prefetcher-caused pollution using informed caching policies for prefetched blocks," *ACM Transactions on Architecture and Code Optimization*, vol. 11, no. 4, 2015.

[12] AMD, "AMD 3D V-Cache™ Technology," 2025. [Online]. Available: https://www.amd.com/en/products/processors/technologies/3d-v-cache.html

[13] F. Alted, "Breaking down memory walls," 2018. [Online]. Available: https://www.blosc.org/posts/breaking-memory-walls/

[14] Intel, "Intel® Xeon® Processor E3-1245 v5 8M Cache, 3.50 GHz," 2025. [Online]. Available: https://www.intel.com/content/www/us/en/products/sku/88173/intel-xeon-processor-e31245-v5-8m-cache-3-50-ghz/specifications.html

[15] LZ4, "LZ4 - Extremely fast compression," 2025. [Online]. Available: https://lz4.org

[16] Samsung, "PIM Technologies - Samsung Semiconductor Global," 2025. [Online]. Available: https://semiconductor.samsung.com/technologies/memory/pim/

[17] D.-I. Jeon, K.-B. Park, and K.-S. Chung, "HMC-MAC: processing-in memory architecture for multiply-accumulate operations with hybrid memory cube," *IEEE Computer Architecture Letters*, vol. 17, no. 1, pp. 5–8, May 2017.

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

[19] J. San Miguel, M. Badr, and N. E. Jerger, "Load value approximation," in *International Symposium on Microarchitecture*. IEEE, 2014, pp. 127–139.

[20] D. T. Nguyen, N. H. Hung, H. Kim, and H.-J. Lee, "An approximate memory architecture for energy saving in deep learning applications," *IEEE Transactions on Circuits and Systems I: Regular Papers*, vol. 67, no. 5, pp. 1588–1601, 2020.

[21] A. Ranjan, A. Raha, V. Raghunathan, and A. Raghunathan, "Approximate memory compression for energy-efficiency," in *International Symposium on Low Power Electronics and Design*. IEEE, 2017, pp. 1–6.

[22] L. Ceze, K. Strauss, J. Tuck, J. Torrellas, and J. Renau, "CAVA: Using checkpoint-assisted value prediction to hide l2 misses," *ACM Transactions on Architecture and Code Optimization*, vol. 3, no. 2, pp. 182–208, June 2006.

[23] G. Reinman and B. Calder, "Predictive techniques for aggressive load speculation," in *International Symposium on Microarchitecture*. IEEE, 1998, pp. 127–137.

[24] A. Yazdanbakhsh, G. Pekhimenko, B. Thwaites, H. Esmaeilzadeh, O. Mutlu, and T. C. Mowry, "RFVP: Rollback-free value prediction with safe-to-approximate loads," *ACM Transactions on Architecture and Code Optimization*, vol. 12, no. 4, pp. 1–26, 2016.

[25] GEM5, "The gem5 Simulator System," 2025. [Online]. Available: https://www.gem5.org

[26] GEM5, "gem5::TimingSimpleCPU Class Reference," 2025. [Online]. Available: https://doxygen.gem5.org/release/current/classgem5_1_1TimingSimpleCPU.html

[27] Open Hardware Group, "CVA6 RISC-V CPU," 2025. [Online]. Available: https://github.com/openhwgroup/cva6

[28] RISC-V, "RISC-V international," 2025. [Online]. Available: https://riscv.org

[29] M. Masadeh, O. Hasan, and S. Tahar, *Adaptive Approximate Accelerators with Controlled Quality Using Machine Learning.* Springer Nature, 2024, pp. 501–529.

[30] S. Annadurai, "Image compression," in *Fundamentals of digital image processing.* Pearson Education India, 2007, ch. 5, pp. 131–226.

[31] S. Birla, S. K. Dargar, N. Singh, and P. Sivakumar, *Low Power Designs in Nanodevices and Circuits for Emerging Applications.* CRC Press, 2023.

[32] S. Valentine, "List of blend modes," in *Hidden Power of Blend Modes in Adobe Photoshop.* Adobe Press, 2012, ch. 7, p. 150. [Online]. Available: https://www.peachpit.com/store/hidden-power-of-blend-modes-in-adobe-photoshop-9780321823762

[33] J. Hass, "Synthesis," in *Introduction to Computer Music.* Indiana University, USA, 2021, ch. 4. [Online]. Available: https://cmtext.indiana.edu/synthesis/chapter4_am_rm.php

[34] R. Gonzalez and R. Woods, "Image segmentation," in *Digital Image Processing*. Pearson, 2017, ch. 10, pp. 699–810. [Online]. Available: https://www.pearson.com/subject-catalog/p/digital-image-processing-global-edition/P200000004313/9781292223070

[35] E. Tarr, "Distortion, saturation, and clipping element-wise processing: Nonlinear effects," in *Hack Audio: An Introduction to Computer Programming and Digital Signal Processing in MATLAB*. Taylor & Francis, 2018, ch. 10, pp. 147–182. [Online]. Available: https://doi.org/10.4324/9781351018463

[36] R. Gonzalez and R. Woods, "Intensity transformations and spatial," in *Digital Image Processing*. Pearson, 2017, ch. 3, pp. 119–202. [Online]. Available: https://www.pearson.com/subject-catalog/p/digital-image-processing-global-edition/P200000004313/9781292223070

[37] E. Tarr, "Signal gain and dc offset," in *Hack Audio: An Introduction to Computer Programming and Digital Signal Processing in MATLAB*. Taylor & Francis, 2018, ch. 6, pp. 57–78. [Online]. Available: https://doi.org/10.4324/9781351018463

[38] Apple, "Use ES2 ring modulation in Logic Pro for Mac," 2025. [Online]. Available: https://support.apple.com/en-ca/guide/logicpro/lgsia1273a3/11.1/mac/14.6

[39] N. Otsu, "A threshold selection method from gray-level histograms," *IEEE Transactions on Systems, Man, and Cybernetics*, vol. 9, no. 1, pp. 62–66, 1979.

[40] J. M. White and G. D. Rohrer, "Image thresholding for optical character recognition and other applications requiring character image extraction," *IBM Journal of Research and Development*, vol. 27, no. 4, pp. 400–411, 1983.

[41] Dyson, "The difference between active noise cancellation and passive noise cancellation," 2024. [Online]. Available: https://www.dyson.com/discover/insights/audio/noise-canceling/the-difference-between-active-noise-cancellation-and-passive-noise-cancellation

[42] A. Aoun, M. Masadeh, and S. Tahar, "A machine learning based load value approximator guided by the tightened value locality," in *Great Lakes Symposium on VLSI*. ACM, 2023, pp. 679–684.

[43] C. Bienia, "Benchmarking modern multiprocessors," Ph.D. dissertation, Princeton University, USA, 2011. [Online]. Available: https://www.cs.princeton.edu/techreports/2010/890.pdf

[44] Scikit-Learn, "Sklearn Ensemble Extra Trees Regressor," 2024. [Online]. Available: https://scikit-learn.org/stable/modules/generated/sklearn.ensemble.ExtraTreesRegressor.html

[45] A. Aoun, M. Masadeh, and S. Tahar, "Machine learning based memory load value predictor for multimedia applications," in *International Conference on Microelectronics*, 2024, pp. 1–6.

[46] R. Timofte, V. De Smet, and L. Van Gool, "Anchored neighborhood regression for fast example-based super-resolution," in *International Conference on Computer Vision*. IEEE, 2013, pp. 1920–1927.

[47] M.-E. Nilsback and A. Zisserman, "Automated flower classification over a large number of classes," in *Indian Conference on Computer Vision, Graphics & Image Processing*. IEEE, 2008, pp. 722–729. [Online]. Available: https://paperswithcode.com/dataset/oxford-102-flower

[48] J. Krause, M. Stark, J. Deng, and L. Fei-Fei, "3D Object Representations for Fine-Grained Categorization," in *International Conference on Computer Vision Workshops*. IEEE, 2013, pp. 554–561. [Online]. Available: https://paperswithcode.com/dataset/stanford-cars

[49] B. Zhou, A. Lapedriza, J. Xiao, A. Torralba, and A. Oliva, "Learning deep features for scene recognition using places database," in *International Conference on Neural Information Processing Systems*, vol. 1. MIT Press, 2014, p. 487–495. [Online]. Available: https://dl.acm.org/doi/10.5555/2968826.2968881

[50] J. Ward, "Sound Archive for Babylon 5," 2014. [Online]. Available: http://b5.cs.uwyo.edu/bab5/

[51] J.-R. Ohm, "Bildsignalverarbeitung fuer multimedia-systeme," *Skript*, vol. 1, p. 2, 1999.

[52] J. Klaue, B. Rathke, and A. Wolisz, "EvalVid – a framework for video transmission and quality evaluation," in *Computer Performance Evaluation Modelling Techniques and Tools,* vol. 2794 of Lecture Notes in Computer Science. Springer, 2003, pp. 255–272.

[53] J. Gross, J. Klaue, H. Karl, and A. Wolisz, "Cross-layer optimization of OFDM transmission systems for MPEG-4 video streaming," *Science Direct Computer Communications*, vol. 27, no. 11, pp. 1044–1055, 2004.

[54] S. Che, M. Boyer, J. Meng, D. Tarjan, J. W. Sheaffer, S.-H. Lee, and K. Skadron, "Rodinia: A benchmark suite for heterogeneous computing," in *International Symposium on Workload Characterization*. IEEE, 2009, pp. 44–54. [Online]. Available: https://doi.org/10.1109/iiswc.2009.5306797

[55] B. He, W. Fang, Q. Luo, N. K. Govindaraju, and T. Wang, "Mars: a mapreduce framework on graphics processors," in *International Conference on Parallel Architectures and Compilation Techniques*. ACM, 2008, p. 260–269. [Online]. Available: https://doi.org/10.1145/1454115.1454152

[56] Nvidia, "CUDA Samples," 2018. [Online]. Available: https://docs.nvidia.com/cuda/archive/10.0/cuda-samples/index.html

[57] QEMU, "A generic and open source machine emulator and virtualizer," 2025. [Online]. Available: https://www.qemu.org

[58] Cadence, "Protium enterprise prototyping," 2025. [Online]. Available: https://www.cadence.com/en_US/home/tools/system-design-and-verification/emulation-and-prototyping/protium.html

[59] Cadence, "Palladium emulation," 2025. [Online]. Available: https://www.cadence.com/en_US/home/tools/system-design-and-verification/emulation-and-prototyping/palladium.html

[60] JEDEC, "DDR3 SDRAM Standard," 2012. [Online]. Available: https://www.jedec.org/standards-documents/docs/jesd-79-3d

[61] JEDEC, "DDR4 SDRAM Standard," 2021. [Online]. Available: https://www.jedec.org/standards-documents/docs/jesd79-4a

[62] Intel, "13th Generation Intel® Core™ and Intel® Core™ 14th Generation Processors – Datasheet, Volume 1 of 2," 2024. [Online]. Available: https://edc.intel.com/content/www/us/en/design/products/platforms/details/raptor-lake-s/13th-generation-core-processors-datasheet-volume-1-of-2/ia-cores-level-1-and-level-2-caches/

[63] A. Waterman, Y. Lee, D. Patterson, and K. Asanović, "Chapter 9 - RV32/64G Instruction Set Listings," in *The RISC-V Instruction Set Manual, Volume I: User- Level ISA, Version 2.1.* Electrical Engineering and Computer Sciences, University of California at Berkeley, USA, 2016, pp. 53–57.

[64] ARM, "AMBA AXI protocol specification," 2025. [Online]. Available: https://developer.arm.com/documentation/ihi0022/latest

[65] A. Waterman, K. Asanović, and J. Hauser, *The RISC-V Instruction Set Manual, Volume II: Privileged Architecture*, RISC-V International, Dec. 2021.

[66] I. Sarno, S. Di Matteo, E. Valea, and C. Chavet, "RISC-V-based acceleration strategies for post-quantum cryptography," in *RISC-V Summit Europe*, 2025, pp. 1–2.

[67] N. Wistoff, "cva6_accel_first_pass_decoder_stub.sv," 2023. [Online]. Available: https://github.com/openhwgroup/cva6/blob/master/core/cva6_accel_first_pass_decoder_stub.sv

[68] M. Cavalcante and N. Wistoff, "acc_dispatcher.sv," 2020. [Online]. Available: https://github.com/openhwgroup/cva6/blob/master/core/acc_dispatcher.sv

[69] F. Zaruba, "ariane_pkg.sv," 2017. [Online]. Available: https://github.com/openhwgroup/cva6/blob/master/core/include/ariane_pkg.sv

[70] ARM, "Arm® Cortex-A720 Core Technical Reference Manual - Revision: r0p2," 2023. [Online]. Available: https://developer.arm.com/documentation/102530/0002

[71] Printcton Parallel Group, "OpenPiton open source research processor," 2017. [Online]. Available: https://www.openpiton.org

[72] IEEE, "IEEE Standard for SystemVerilog–Unified Hardware Design, Specification, and Verification Language," *IEEE Std 1800-2023 (Revision of IEEE Std 1800-2017)*, pp. 1–1354, 2024.

[73] PULP Platform, "AXI4-compliant last-level cache (LLC)," 2025. [Online]. Available: https://github.com/pulp-platform/axi_llc

[74] PULP Platform, "PULP Platform: Open hardware, the way it should be!" 2025. [Online]. Available: https://pulp-platform.org

[75] Micron Technology Inc., *Simulation Model: DDR4 SDRAM Verilog Model*, 2018. [Online]. Available: https://www.micron.com/content/dam/micron/global/secure/products/sim-model/dram/ddr4/ddr4-verilog-models.zip

[76] Siemens Digital Industries Software, "Questa advanced simulator," 2025. [Online]. Available: https://eda.sw.siemens.com/en-US/ic/questa/simulation/advanced-simulator/

[77] Cadence, "Innovus Implementation System," 2025. [Online]. Available: https://www.cadence.com/ko_KR/home/tools/digital-design-and-signoff/soc-implementation-and-floorplanning/innovus-implementation-system.html

[78] AMD, "Pipelining the RAM," 2025. [Online]. Available: https://docs.amd.com/r/en-US/ug901-vivado-synthesis/Pipelining-the-RAM

Biography

Education

- **Concordia University**: Montreal, Quebec, Canada.

  Ph.D., Electrical & Computer Engineering – 2021-2025

- **Concordia University**: Montreal, Quebec, Canada.

  M.A.Sc., Electrical & Computer Engineering – 2019-2021

- **Notre Dame University**: Zouk-Mosbeh, Lebanon.

  B.E., Electrical Engineering – 2013-2018

Awards

- Concordia University Conference and Exposition Award (PhD) - 2023
- Concordia University Merit Scholarship (PhD) – 2021
- Concordia University International Tuition Award of Excellence (PhD) – 2021-2024
- Concordia University International Tuition Award of Excellence (MASc) – 2019-2021
- Notre Dame University Academic Honors: Dean's List – 2014-2018

Work History

- **Research Assistant**, Hardware Verification Group, Department of Electrical and Computer Engineering, Concordia University, Montreal, Quebec, Canada – 2019-2025
- **Teaching Assistant**, Department of Electrical and Computer Engineering, Concordia University, Montreal, Quebec, Canada – 2021-2024
- **Webmaster**, Hardware Verification Group, Department of Electrical and Computer Engineering, Concordia University, Montreal, Quebec, Canada – 2019-2025
- **IT Admin**, Hardware Verification Group, Department of Electrical and Computer Engineering, Concordia University, Montreal, Quebec, Canada – 2020-2025

Publications

Journal Papers

[Bio-Jr1] **A. Aoun**, M. Masadeh and S. Tahar: ML-based Load Value Approximator for Efficient Multimedia Processing; ACM Transactions on Multimedia Computing Communications and Applications, Vol. 21, No. 7, July 2025, pp. 210:1-210:18.

[Bio-Jr2] M. Masadeh, **A. Aoun** and S. Tahar: Design Space Exploration of Array-based Approximate Squaring Unit for Error-tolerant Computing; Analog Integrated Circuits and Signal Processing, Vol. 124, No. 53, July 2025, pp. 1-17.

[Bio-Jr3] **A. Aoun**, M. Masadeh and S. Tahar: A RISC-V based Load Value Approximator Accelerator for Efficient Multimedia Processing; IEEE Transactions on Emerging Topics in Computing, July 2025, pp. 1-11. [Under Review]

[Bio-Jr4] **A. Aoun**, M. Masadeh, O. Hasan and S. Tahar: Design Space Exploration for Approximate Circuits Based on Arithmetic Calculus Modeling; Journal of Circuits, Systems and Computers, World Scientific, May 2025, pp. 1-30. [Under Review]

[Bio-Jr5] **A. Aoun**, M. Masadeh and S. Tahar: Design Space Exploration for Energy-Efficient Approximate Sobel Filter; International Journal of Electronics and Communications, Elsevier, Vol. 172, 154887, December 2023, pp. 1-9.

Conference Papers

[Bio-Cf1] **A. Aoun**, M. Masadeh and S. Tahar: Machine Learning Based Memory Load Value Predictor for Multimedia Applications; Proc. IEEE International Conference on Microelectronics (ICM'24), Doha, Qatar, December 2024, pp. 1-6.

[Bio-Cf2] **A. Aoun**, M. Masadeh, O. Hasan and S. Tahar: Arithmetic Calculus Modeling for Approximate Circuits; Proc. IEEE International Conference on Microelectronics (ICM'24), Doha, Qatar, December 2024, pp. 1-6.

[Bio-Cf3] M. Masadeh, **A. Aoun** and S. Tahar: Energy-Efficient Approximate Squaring Unit; Proc. IEEE International Conference on Microelectronics (ICM'24), Doha, Qatar, December 2024, pp. 1-5.

[Bio-Cf4] **A. Aoun**, M. Masadeh and S. Tahar: On the Implementation of Approximate Load Value using Machine Learning; Design Automation Conference (DAC'23), San Francisco, California, USA, July 2023. [Poster Presentation]

[Bio-Cf5] **A. Aoun**, M. Masadeh and S. Tahar: A Machine Learning based Load Value Approximator guided by the Tightened Value Locality; Proc. ACM Great Lakes Symposium on VLSI (GLS-VLSI'23), Knoxville, Tennessee, USA, June 2023, ACM Publications, pp. 679-684.

[Bio-Cf6] **A. Aoun**, M. Masadeh and S. Tahar: On the Design of Approximate Sobel Filter; Proc. IEEE International Conference on Microelectronics (ICM'22), Casablanca, Morocco, December 2022, pp. 1-5. [**BEST PAPER**]

[Bio-Cf7] M. Masadeh, **A. Aoun**, O. Hasan and S. Tahar: Highly-Reliable Approximate Quadruple Modular Redundancy with Approximation-Aware Voting; Proc. IEEE International Conference on Microelectronics (ICM'20), Irbid, Jordan, December 2020, pp. 1-4.

[Bio-Cf8] M. Masadeh, **A. Aoun**, O. Hasan and S. Tahar: Decision Tree-based Adaptive Approximate Accelerators for Enhanced Quality; Proc. IEEE International Systems Conference (SysCon'20), Montreal, Quebec, Canada, August 2020, pp. 1-5.

[Bio-Cf9] **A. Aoun**, A. Iliovits, A. Kassem, P. Sakr and N. Metni: Arthro-Glove a Hybrid Bionic Glove for patients diagnosed with Arthritis, ALS and/or Dysmorphia; Proc. IEEE Cairo International Biomedical Engineering Conference (CIBEC'18), Cairo, Egypt, 2018, pp. 106-109.

[Bio-Cf10] E.J. Maalouf, N. Marina, J. B. Abdo, **A. Aoun**, M. Hamad and A. Kassem: Asthma Irritant Monitoring; Proc. IEEE International Conference on Microelectronics (ICM'18), Sousse, Tunisia, 2018, pp. 120-123.

[Bio-Cf11] **A. Aoun**, A. Kassem and M. Hamad: Sun Stimulator for Daylight System; Proc. IEEE International Arab Conference on Information Technology (ACIT'18), Werdanye, Lebanon, 2018, pp. 1-4.

[Bio-Cf12] A. Kassem, M. Hamad, C. El Moucary, E. Nawfal and **A. Aoun**: MedBed: Smart medical bed; Proc. IEEE International Conference on Advances in Biomedical Engineering (ICABME'17), Beirut, Lebanon, 2017, pp. 1-4.