A RISC-V based Load Value Approximator Accelerator for Efficient Multimedia Processing

Alain Aoun, Mahmoud Masadeh, Sofiène Tahar

***Abstract*—This paper presents the hardware implementation and evaluation of a Machine Learning-based Load Value Approximator (ML-LVA) integrated into the CVA6 RISC-V processor. The ML-LVA is deployed as a lightweight accelerator, invoked through custom RISC-V instructions, and realized using a ROM-based predictor. The design is synthesized using a 45nm CMOS process with Cadence Innovus and evaluated at 3GHz under realistic memory configurations. Image and audio processing workloads were used to assess performance. The proposed ML-LVA achieves up to 1.08× speedup at the application level and up to 1.73× in memory operations, significantly outperforming prior LVA approach. Hardware overhead remains modest, with area and power increases of only 5.09% and 0.7%, respectively. These results confirm that ML-LVA can be effectively and efficiently integrated into modern out-of-order processors to reduce memory latency and enhance throughput with minimal resource cost.**

***Index Terms*—Approximate Computing, Approximate Memory, Approximate Load Value, Machine Learning, RISC-V, Accelerator, Multimedia Processing**

I. Introduction

The widening disparity between processor speed and memory latency has long been recognized as a central bottleneck in computer architecture. This phenomenon, commonly referred to as the *memory wall* [1], stems from the fundamental constraints of the von Neumann architecture [2], where the performance gains in compute units have far outpaced the improvements in memory subsystems. Since the mid-20th century, advancements in transistor scaling, guided by Moore's Law, have yielded exponential growth in processor frequencies and computational throughput. However, memory systems, bounded by the physical limitations of charge storage and interconnect delays, have not kept pace. As a result, modern CPUs frequently encounter performance stalls due to delayed memory accesses, particularly on cache misses that saturate structures such reorder buffers (ROBs) [3].

Efforts to mitigate this latency gap have focused on microarchitectural innovations. Techniques such as out-of-order execution, speculative execution, and advanced prefetching have significantly improved the effective throughput of modern processors. For instance, hardware prefetchers in architectures like AMD's Zen 3 [4], and machine learning-based schemes such as DeepPrefetcher [5], attempt to anticipate memory access patterns. These solutions primarily attempt to mask memory latency rather than eliminate it, and often incur trade-offs such as cache pollution [6] or increased complexity. Another line of work addresses memory latency through Load Value Speculation (LVS) [7], wherein processors predict the value of a pending memory load to continue execution speculatively. LVS introduces minimal architectural disruption, similar to branch prediction, but entails complex validation mechanisms and costly rollbacks on misprediction.

Recently, researchers have investigated relaxing the strict correctness requirement of Load Value Speculation (LVS) within the broader framework of Approximate Computing (AC) [8], particularly for applications that can tolerate minor inaccuracies, such as multimedia processing, machine learning, and gaming. This has given rise to Load Value Approximation (LVA), where predicted values are allowed to be imprecise, thereby avoiding costly rollbacks associated with mispredictions. Despite its promise, existing LVA implementations often incur significant hardware overhead. This is primarily due to the use of predictor lookup tables, complex hashing mechanisms, and dynamic predictors requiring frequent memory accesses to maintain acceptable quality. To address these challenges, this work explores a Machine Learning-based Load Value Approximation (ML-LVA) approach that leverages the principles of AC while reducing prediction overhead. The proposed ML-LVA is static and lightweight, requiring minimal runtime resources. We introduce a low-cost hardware implementation of the ML-LVA predictor. In contrast to prior designs, our approach eliminates the need for complex runtime structures, offering a more efficient and scalable solution.

The remainder of this paper is organized as follows. Section II surveys related work in load value approximation (LVA) and memory latency mitigation. Section III describes the proposed architecture and predictive model. Section IV presents the implementation details of the LVA, followed by an experimental performance analysis in Section V. Section VI compares the proposed implementation with the state-of-the-art, and Section VII discusses the associated overhead. Finally, Section VIII concludes the paper and outlines directions for future work.

II. Related Work

A range of research efforts has been directed toward mitigating memory bottlenecks in modern computing systems. While early studies focused on exploiting memory locality, more recent approaches have explored approximate computing, value prediction, and intelligent prefetching to address this challenge. In this section, we restrict our discussion to methods that are most relevant to the proposed solution in this paper. These include approximate memory systems, machine learning-based memory prefetchers, traditional load value speculation (LVS), and load value approximation (LVA) techniques.

A. Approximate Memory

Approximate memory systems offer energy and bandwidth savings by allowing imprecise data storage when applications can tolerate minor inaccuracies. The work in [9] in-

vestigates a transposed DRAM architecture with variable refresh rates—refreshing rows containing Most Significant Bits (MSBs) more frequently than those with Least Significant Bits (LSBs). This mechanism enables selective loading of MSBs to reduce memory access time and improve throughput. Similarly, [10] proposes a DRAM design that compresses error-tolerant data, allowing different memory regions to operate under varying precision constraints through software-hardware coordination.

While both approaches offer innovative mechanisms for trading off precision for energy and bandwidth efficiency, they also face significant limitations. In [9], the transposed storage and region-specific refresh scheduling increase hardware complexity and pose management challenges, especially in systems with mixed-precision requirements. Moreover, exposing approximate regions can introduce security vulnerabilities, such as susceptibility to side-channel attacks or silent data corruption from malicious bit flipping. The method in [10], although more flexible due to its software-defined approximation levels, incurs latency overhead due to on-the-fly compression and decompression. Additionally, it relies heavily on accurate workload profiling, which limits its adaptability and scalability in general-purpose systems.

B. ML-based Prefetching

ML-based prefetchers aim to anticipate future memory accesses and load data into the cache preemptively. DeepPrefetcher [5] exemplifies this class by employing deep learning models trained on memory access traces to detect complex, non-linear patterns that conventional prefetchers fail to capture. Its ability to adapt in real-time makes it particularly effective in workloads with irregular or data-dependent memory access patterns, significantly improving cache hit rates and reducing latency.

However, ML-based prefetchers are not without drawbacks. One common issue is *cache pollution*, where inaccurate predictions result in unnecessary data being fetched and evict useful cache lines, potentially worsening performance [6]. Additionally, DeepPrefetcher's reliance on continuous online training incurs substantial computational and energy overheads. The model must be frequently updated to remain accurate across dynamic workloads, necessitating specialized hardware support and raising concerns about scalability, especially in energy-constrained environments.

C. Load Value Speculation

Load Value Speculation (LVS) seeks to reduce effective memory latency by predicting the value to be loaded before it arrives. Foundational work in [7] introduced value locality and proposed lookup tables that speculate load values based on previous memory behavior. Later studies, such as [11] and [12], refined this approach with techniques including last-value, stride-based, and context-based prediction. Hybrid schemes and confidence mechanisms were also developed to enhance prediction accuracy and reduce misspeculation penalties.

Despite these advancements, LVS inherently requires validation through actual memory access, meaning incorrect predictions necessitate pipeline rollbacks. These rollbacks not only incur performance penalties but also complicate the processor design due to the need for speculative state tracking and recovery mechanisms. Furthermore, as workloads become increasingly irregular, the effectiveness of deterministic predictors declines, making LVS less suitable for modern, dynamic applications.

D. Load Value Approximation

To eliminate the cost of rollbacks in LVS, Load Value Approximation (LVA) allows approximate predictions to proceed without validation, thus trading precision for performance. The approach in [13] implements an LVA system using dynamic predictors that uses Global History Buffer (GHB) and Local History Buffer (LHB). These structures help estimate load values based on recent patterns, and the system uses relaxed confidence windows to tolerate small errors. The technique also defines an approximation degree to control the reuse of predicted values before retraining, enabling a performance-quality trade-off.

Nevertheless, the proposed design in [13] is burdened by significant hardware complexity. The need to maintain hash complex history buffers, along with continuous updates to the dynamic predictor, demands both memory bandwidth and computational resources. Moreover, the effectiveness of this approach hinges on the quality of recent history, making it less reliable in irregular or noisy workloads.

Similarly, RFVP [14] introduces an LVA scheme tailored for GPUs, which predicts load values for cache misses using a history-based predictor indexed by hashed program counters. It accommodates GPU-specific execution patterns, such as warp divergence, by precomputing predictions for inactive threads. Although RFVP improves consistency and prediction stability using stride tracking, it also shares the drawbacks of dynamic prediction. The hashing mechanism, value history tracking, and selective discarding policies all add to the computation overhead, complicating hardware implementation. Moreover, like [13], it relies heavily on precise control logic to maintain acceptable quality across varying workloads.

In [15], we introduced a machine learning-based Load Value Approximation (ML-LVA) technique specifically tailored for multimedia applications, where approximate computations are often acceptable due to the inherent error tolerance of audio and image processing tasks. The ML-LVA model was implemented entirely in software, utilizing a static predictor that leverages recent load history to generate high-quality approximations of memory values with minimal error. This static prediction approach presents an advantage over prior dynamic predictors, which incur additional overhead by continuously updating prediction tables and performing complex memory accesses. By employing a static predictor, the method reduces runtime complexity and achieves competitive approximation quality without the costly overhead associated with dynamic prediction schemes. Experimental results demonstrated that the proposed approach consistently outperformed conventional

load value speculation and approximation techniques in terms of both approximation accuracy and overall system performance.

However, the software-based implementation in [15] relied on subroutine calls within application code, which introduced performance overhead due to the dependence on general-purpose processor execution and frequent memory accesses during model inference. Additionally, lacking hardware acceleration limited the ability to fully exploit architectural features such as speculative execution and memory access pipelining. Consequently, while the software ML-LVA demonstrated clear potential, its capability for system-wide speedup and energy savings remained constrained compared to a dedicated hardware implementation.

In this work, we address these limitations by presenting a hardware implementation of the ML-LVA model. This architecture-aware deployment minimizes inference latency and runtime overhead by integrating the ML-LVA directly into the processor pipeline via a custom instruction extension. By synthesizing the design for ASIC implementation and evaluating its performance on representative multimedia workloads, we demonstrate that the proposed hardware ML-LVA achieves significantly higher speedup and energy efficiency than the prior software-only version, enabling practical deployment in high-performance, error-tolerant systems.

III. Proposed Methodology

This section outlines the approach taken to integrate the ML-LVA model 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.

To this end, we proposed the methodology shown in Fig. 1. 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.

In parallel, in Step ③, 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 ③ is thus a structured dataset mapping historical load values to the subsequent target value. In Step ④, we train the machine learning model using the Extra Trees algorithm [16]. 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 [17] corroborated the effectiveness of Extra Trees for this prediction setting. The output of Step ④ is a trained ML-LVA.

In Step ⑤, we implement the ML-based Load Predictor within the hardware domain. A pragmatic and computationally

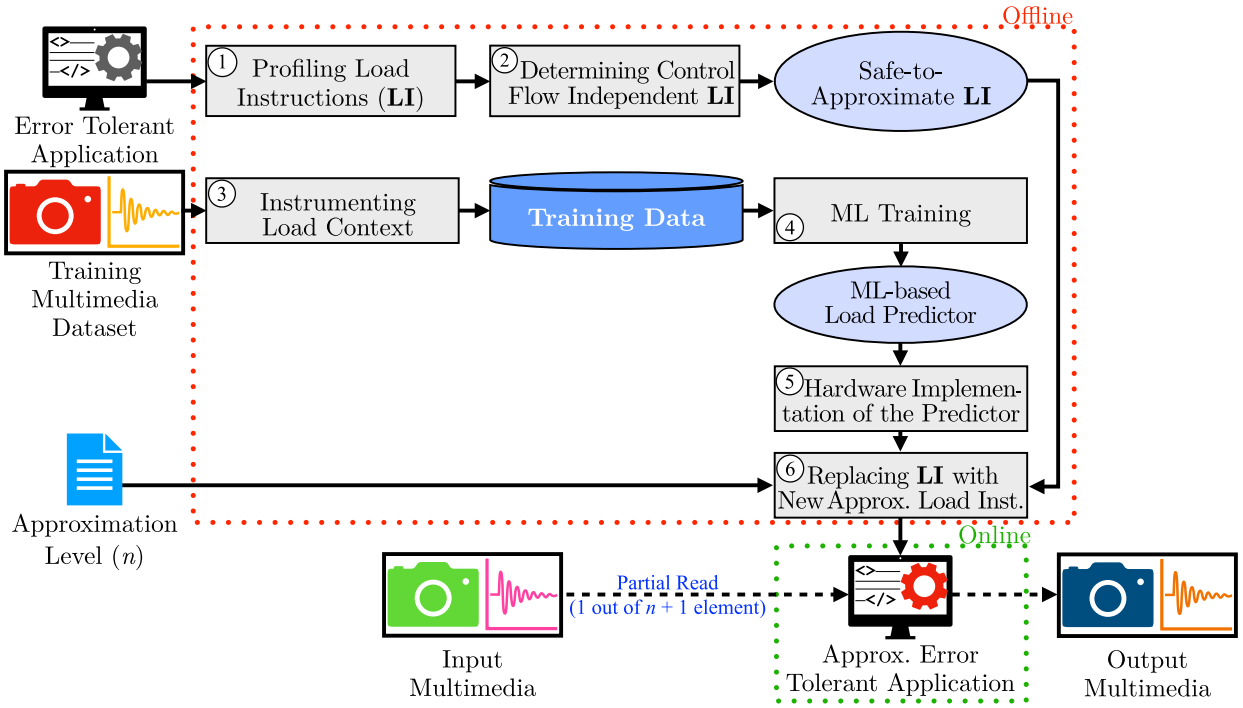

Fig. 1. Methodology to Implement the Proposed LVA in Hardware

efficient solution is achieved using a **lookup table**. This approach is particularly suitable for the ML-LVA model developed in this paper, 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 favourable 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 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 [18], which explicitly reserves certain opcode spaces—specifically the *Custom-0* and *Custom-1* instruction groups—for user-defined extensions [19]. 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 illustrate how the structural changes introduced in Step ⑥ affect the behavior of the error-tolerant application, we present the prediction sequence in Fig. 2. This figure visualizes the operation of the proposed LVA, where squares represent exact load values while circles denote approximate (predicted) values. Each approximation is derived from the most recent value, regardless of whether it was exact or approximate. For example, the first approximate value (A_1) is predicted based on the exact value (E) fetched from memory. Subsequently, the second approximate value (A_2) is predicted

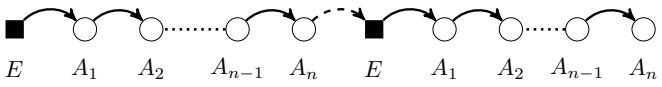

Fig. 2. Prediction Sequence

based on A_1, which is itself a predicted value. This prediction sequence repeats for n times, where n corresponds to the approximation level, thereby producing values A_1 through A_n. After generating n approximations, an exact value (E) is fetched again from memory to resynchronize the prediction stream, and the cycle repeats.

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, to deploy the two ML-LVA. 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.

IV. 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 work, the hardware in which the ML-LVA was integrated consists of a RISC-V processor called CVA6 [20], which has L1 cache, an L2 cache and a DRAM as shown in Fig. 3. All components are connected via the Advanced Microcontroller Bus Architecture (AMBA) Advanced eXtensible Interface version 4 (AXI4) protocol [21]. In the rest of this section, we will present the details of each of these components and the reasoning for their selection.

Fig. 3. Hardware Implementation Environment

A. CVA6 Processor

At the core of the testing environment is the CVA6 processor, a 64-bit in-order RISC-V core [20]. The CVA6 features a six-stage pipeline architecture, as illustrated in Fig. 4. 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 [19], along with [22]. 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 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 [21] 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 work 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. Accordingly, we modified the source codes of "*cva6_accel_first_pass_decoder_stub*" [23] and "*acc_dispatcher*" [24] blocks. Subsequently, the modified codes affect the Instruction Decode (ID) and Execute stages shown in Fig. 4. 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 [25], 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 [26]. The

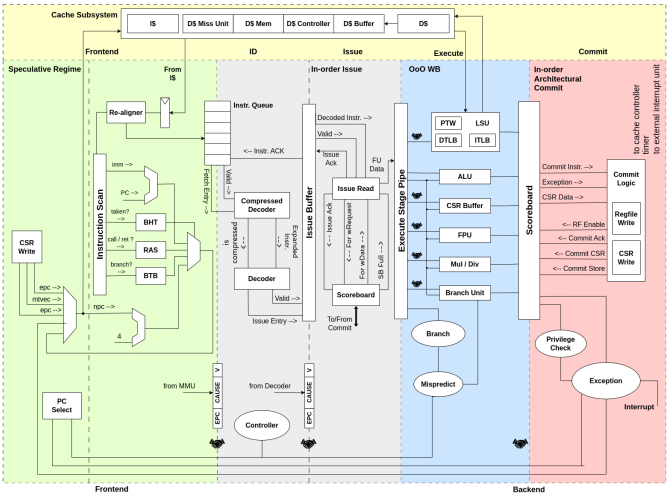

Fig. 4. Architecture of the CVA6 [20]

data cache was configured as an OpenPiton cache [27] with a write through policy. Finally, the simulation environment for the modified CVA6 was developed in SystemVerilog [28] 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.

B. AXI Last Level Cache

To complement the memory hierarchy of the CVA6-based testing platform, the AXI Last Level Cache (LLC) [29] is employed as a unified Level-2 (L2) cache. The AXI LLC is an open-source configurable development within the Parallel Ultra-Low Power (PULP) platform [30]. 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 [26]. 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 [29], 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 $N_{DB} = 256$. Based on this configuration, the total size of the LLC can be calculated using the following formula:

$$Size\_LLC = Assoc. \times N_{CL} \times N_{DB} \times \frac{DataWidth}{8} \quad (1)$$

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

$$Size\_LLC = 8 \times 256 \times 8 \times \frac{64}{8} = 131,072 \quad (2)$$

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.

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 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.

C. Micron DDR4 Model

To complete the memory hierarchy of the testing platform, a DRAM memory based on the publicly available DDR4 Verilog simulation model [31], 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 megatransfers 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 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. 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 [32]. 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.

V. 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. Since the same ML-LVA used in this study was already thoroughly evaluated in terms of prediction quality in our previous work [15], we limit this section to performance analysis. In [15], we showed that the quality of the ML model consistently outperformed existing state-of-the-art LVA techniques [13], [14], achieving a peak signal-to-noise ratio (PSNR) exceeding 100 dB in several scenarios and an average normalized mean absolute error (NMAE) of 4.54%.

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 [33], image inversion [34], audio blending [35], and audio inversion [36]. 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 [37].

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 application, we analyze the effect of varying the approximate level (n) on the performance. Using 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 work 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.

A. 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.

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 Fig. 5.

From Fig. 5a, 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

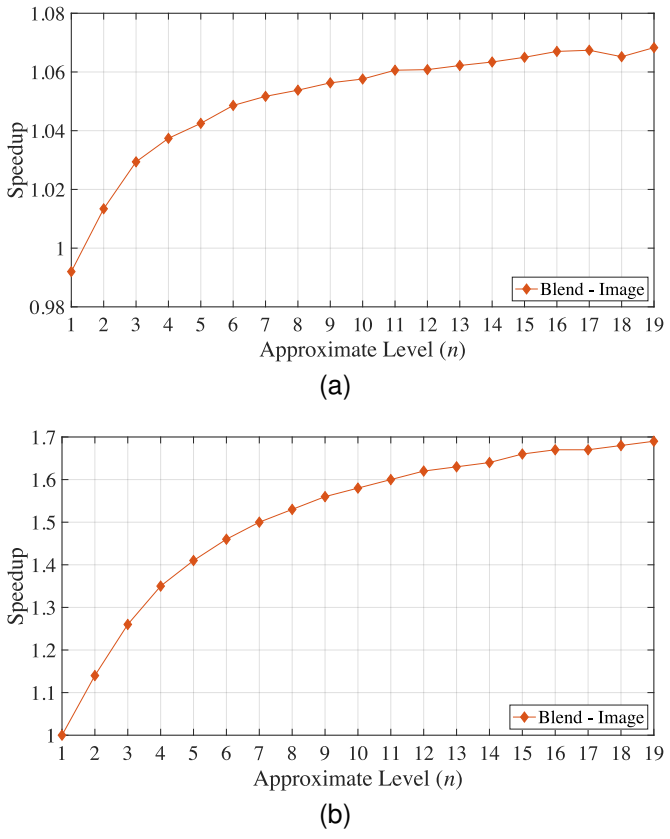

Fig. 5. Average Speedups at the (a) Application, and (b) Memory Levels for Image Blending

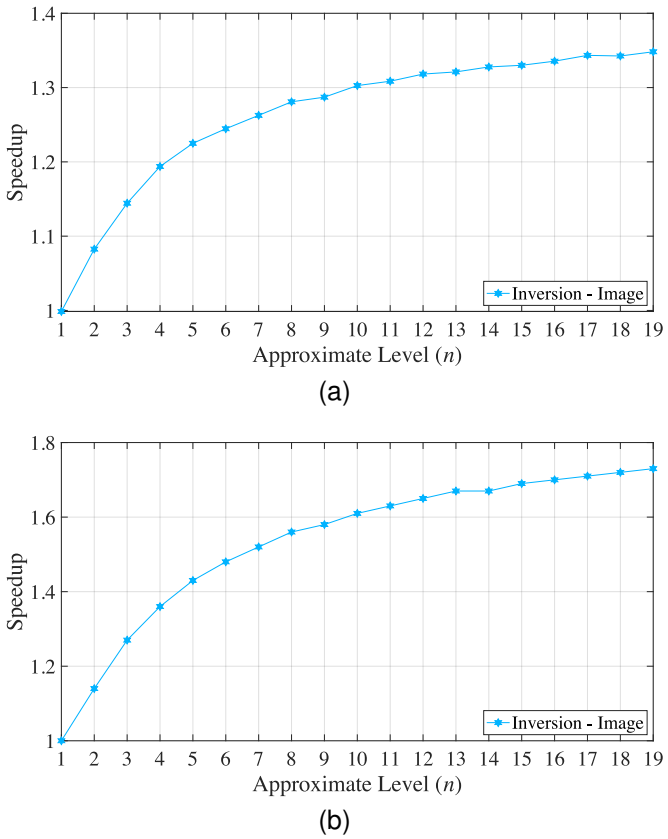

Fig. 6. Average Speedups at the (a) Application, and (b) Memory Levels for Image Inversion

ML-LVA leads to a stable and scalable performance advantage even when used as a tightly coupled accelerator within a general-purpose processor.

From Fig. 5b, 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 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.

2) Image Inversion: The speedups achieved in the image inversion are given in Fig. 6. For the application overall speedup, we notice from Fig. 6*a* that the effect of ML-LVA is 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 Fig. 6b, 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 rising 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 an 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 approximate memory load acceleration.

B. 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 influence 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 LVA 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.

1) Audio Blending: The speedup results of the audio blending, depicted in Fig. 7, show an increased gain as the approximate level n increases. From Fig. 7a, we notice that for the overall application, the improvements are more modest, reflecting the fact that not all 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 Fig. 7b 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.

2) Audio Inversion: The ML-LVA delivers solid gains when tested in audio inversion as shown in Fig. 8. The results of application overall speedup shown in Fig. 8a, reflects 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

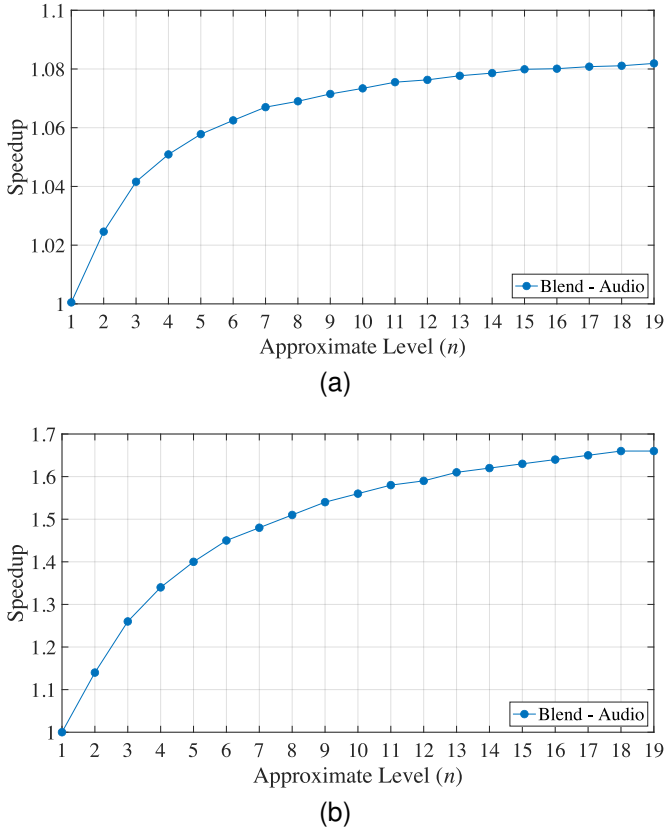

Fig. 7. Average Speedups at the (a) Application, and (b) Memory Levels for Audio Blending

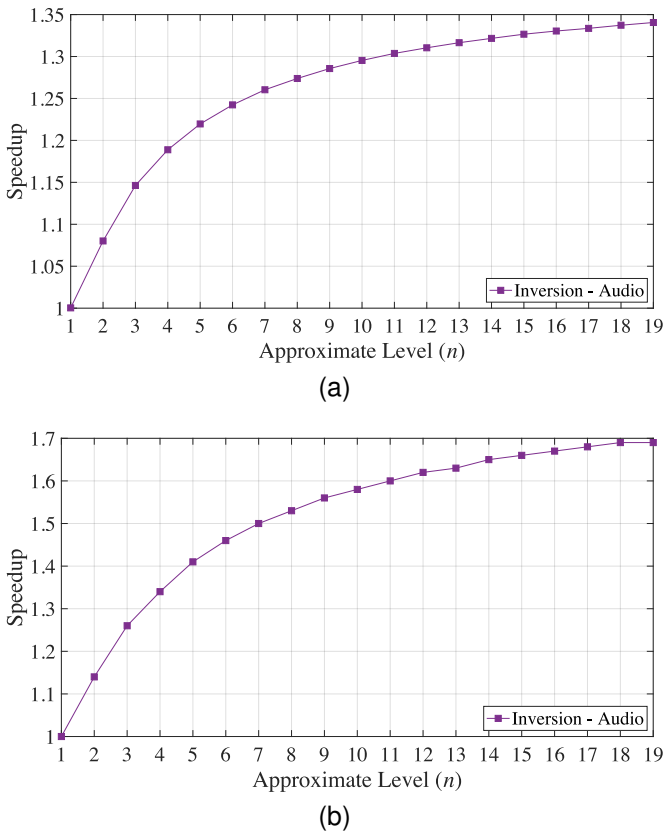

Fig. 8. Average Speedups at the (a) Application, and (b) Memory Levels for Audio Inversion

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

From Fig. 8b, 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.

VI. Comparison with Related Work

To validate the effectiveness of the hardware-based ML-LVA presented in this paper, it is important to compare its speedup against existing state-of-the-art methods. The software-based evaluation reported in [15] has already shown that the ML-LVA surpasses the quality of the LVAs proposed in [13] and [14]. Building on these results, we focus our hardware-level comparison on the LVA introduced in [13]. We omit the method from [14] from this analysis, as it targets GPU architectures, whereas our work is designed for approximating load values in CPU-based systems.

Table I shows the average speedup achieved for the various approximate levels among the various applications. From Table I, we can notice that the LVA proposed in [13] 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 speedup when n increases, while the one proposed in [13] delivered a constant speedup. Subsequently, we conclude that the hardware-based ML-LVA also outperforms the LVA proposed in [13].

TABLE I
Speedup Comparison of the Proposed Hardware-based ML-LVA with [13]

Approximate Level (n)	**LVA** [13]	**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

VII. 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* [38]. 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 II. 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 II
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%

VIII. Conclusion

This paper presented the hardware implementation of a Machine Learning-based Load Value Approximator (ML-LVA) integrated into the CVA6 RISC-V processor. The memory hierarchy, including L1 and L2 caches and a DDR4 DRAM, was designed to reflect modern embedded systems for accurate performance analysis. The ML-LVA was implemented as a lightweight ROM-based accelerator, integrated via custom RISC-V instructions. The hardware implementation achieved up to 1.08× speedup at the application level and 1.73× in memory operations, with the most significant gains observed in audio inversion. Compared to the LVA proposed in [13], the ML-LVA demonstrated superior scalability, delivering improved performance as the approximation level increased. Synthesis results using Cadence Innovus at a 3GHz CPU frequency showed minimal resource overhead, with 5.09% in area and 0.79% in power, which is well justified given the achieved speedups.

Future research could extend the ML-LVA to domains such as edge AI, bioinformatics, and wireless sensor networks, where controlled approximation is both practical and beneficial. Broadening support to additional architectures, such as ARM, would further expand deployment opportunities, particularly in power- and resource-constrained environments. Lastly, formal verification of the ML-LVA is a key direction to ensure system robustness and prevent vulnerabilities, including potential side-channel attacks.

References

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

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

[3] 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.

[4] 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

[5] 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.

[6] 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, Jan. 2015.

[7] M. H. Lipasti, C. B. Wilkerson, and J. P. Shen, "Value locality and load value prediction," *SIGPLAN Notices*, vol. , Volume 31, no. Issue 9, p. 138–147, September 1996. [Online]. Available: https://doi.org/10.1145/248209.237173

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

[9] 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.

[10] 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.

[11] 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.

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

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

[14] 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.

[15] A. Aoun, M. Masadeh, and S. Tahar, "Ml-based load value approximator for efficient multimedia processing," *ACM Trans. Multimedia Comput. Commun. Appl.*, May 2025. [Online]. Available: https://doi.org/10.1145/3736582

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

[17] 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.

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

[19] 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, 2016, pp. 53–57.

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

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

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

[23] 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

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

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

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

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

[28] 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.

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

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

[31] 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

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

[33] 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

[34] R. Gonzalez and R. Woods, "Intensity transformations and spatial," in *Digital Image Processing Global Edition*. Pearson, 2017, ch. 10, pp. 119–202. [Online]. Available: https://www.pearson.com/store/p/digital-image-processing-global-edition/GPROG_A101708555905_learnernz-availability/9781292223049

[35] 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

[36] 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

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

[38] 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

Alain Aoun (Gradate Student Member, IEEE) received the B.Eng. degree in Electrical Engineering from the Notre Dame University, Louaize, Lebanon, in 2018, the M.A.Sc. degree in Electrical and Computer Engineering from Concordia University, Montreal, QC, Canada in 2021, under the supervision of Prof. Sofiène Tahar.

Currently, he is pursuing a Ph.D. degree at Concordia University under the supervision of Prof. Sofiène Tahar. His research focuses on approximate computing, approximate memory and approximate computer architecture.

Mahmoud Masadeh (Senior Member, IEEE) received a B.Sc. degree from Yarmouk University, Irbid, Jordan, in 2003, an M.Sc. degree from Delft University, Delft, The Netherlands, in 2011, both in computer engineering, and a Ph.D. degree in electrical and computer engineering from Concordia University, Canada, in 2020.

Currently, he is an Associate Professor of Computer Engineering at Yarmouk University. He is an author/co-author of more than 50 peer-reviewed journal and conference papers. His research focuses on approximate computing, machine learning and its applications, and energy-efficient VLSI circuit design.

Sofiène Tahar (Senior Member, IEEE) received the Engineering Diploma degree from the University of Darmstadt, Darmstadt, Germany, in 1990, and the Ph.D. degree (Hons.) from the University of Karlsruhe, Karlsruhe, Germany, in 1994.

He is currently a Professor and Senior Research Chair in formal verification of systems-on-chip with the Department of Electrical and Computer Engineering, Concordia University, Montreal, QC, Canada. He is the Founder and Director of the Hardware Verification Group with a research interest in formal verification of hardware and physical systems, and safety and reliability analysis.