ML-based Load Value Approximator for Efficient Multimedia Processing

ALAIN AOUN, Department of Electrical and Computer Engineer, Concordia University, Canada
MAHMOUD MASADEH, Computer Engineering Department, Yarmouk University, Jordan
SOFIÈNE TAHAR, Department of Electrical and Computer Engineer, Concordia University, Canada

Approximate computing (AC) has gained traction as an alternative computing method for energy-efficient processing. This paper proposes the exploitation of AC to address the memory wall. The proposed model predicts the memory load value using machine learning (ML). Subsequently, the ML model is a load value approximator (LVA) where the generated value is accepted as-is. The proposed LVA was tested under various approximate conditions, where 50% to 95% of the load instructions were approximated using a set of multimedia applications. The memory access operation using the proposed LVA was more than 6× faster in multiple cases. Additionally, the applications tested ran on average 1.83× faster. The peak signal-to-noise ratio (PSNR) exceeded 100 *dB* in several scenarios. The average normalized mean absolute error (NMAE) was 4.54%.

CCS Concepts: • **Hardware** → **Memory and dense storage**; • **Computing methodologies** → **Artificial intelligence**; • **Computer systems organization** → **Special purpose systems**.

Additional Key Words and Phrases: Approximate Computing, Approximate Memory, Approximate Load Value, Machine Learning, Multimedia Processing, Image Processing, Audio Processing

ACM Reference Format:
Alain Aoun, Mahmoud Masadeh, and Sofiène Tahar. 2025. ML-based Load Value Approximator for Efficient Multimedia Processing. 1, 1 (May 2025), 18 pages. https://doi.org/10.1145/nnnnnnn.nnnnnnn

1 INTRODUCTION

Approximate Computing (AC), or *inexact computing*, has recently attracted renewed attention as a promising alternative to exact computation in error-tolerant applications. AC results in loss of quality in favour of savings in area, power, and delay [17]. This approach is particularly useful in fields where errors are tolerable due to reasons such as the lack of a single correct solution, noisy input data, limited human sensitivity to noisy output or algorithms with error-mitigating characteristics, such as the algorithms used in machine learning, multimedia processing, and search engines. Most AC research has focused on arithmetic units. For instance the work in [13] proposed approximate full adders that generate the result faster compared to traditional ones and require less area overhead. A different approach of achieving AC is reducing the voltage below the nominal voltage, i.e., voltage overscaling, such as the work in [28] with the aim of reducing the energy consumption and increasing the device lifetime. This paper emphasizes the approximation of the *memory access* process. This involves substituting conventional memory accesses with

Authors' addresses: Alain Aoun, a_alain@ece.concordia.ca, Department of Electrical and Computer Engineer, Concordia University, Montréal, Québec, Canada; Mahmoud Masadeh, mahmoud.s@yu.edu.jo, Computer Engineering Department, Yarmouk University, Yarmouk, Irbid, Jordan; Sofiène Tahar, tahar@ece.concordia.ca, Department of Electrical and Computer Engineer, Concordia University, Montréal, Québec, Canada.

ACM XXXX-XXXX/2025/5-ART
https://doi.org/10.1145/nnnnnnn.nnnnnnn

machine learning-based predicted values, aiming to overcome the *memory wall* challenges in memory-intensive applications such as multimedia processing.

Various approaches have been proposed to mitigate the *memory wall*, such as the usage of process-in-memory, approximate memory, and load value speculation. Process-in-memory shifts repetitive computations, e.g., multiply and accumulate operations, from the CPU to memory or its vicinity to reduce data movement [16]. Approximate memory is another solution, where for example the content of the DRAM is manipulated for error-tolerant applications by compressing or decompressing the data in different regions of the memory in order to provide varying levels of accuracy [24]. In [20], a modified DRAM is proposed with transposed data storage and variable refresh rates for different rows, where rows with less critical bits are refreshed less often to save energy. On the other hand, *load value speculation* 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 [23]. Similar to branch prediction, if the load value speculation is wrong, the CPU reverts to its state before the prediction and flushes the pipeline. In [19], the authors introduced the principal of *value locality*, noting that adjacent elements in memory are often similar in value, e.g., adjacent pixel of an image stored closely. They proposed a dynamic lookup table to predict values during memory loads. On misprediction, the processor reverts to the previous state and flushes the pipeline, updating the table after each access to improve predictions. The concept of load value speculation was further explored by other researchers such as the work in [6] and [25], however, all load value speculation techniques aim to hide memory access delays while requiring a check to the correctness of the predictions.

Extending the concept of load value speculations, *load value approximation* (LVA) uses the predicted value as-is, even if it is inaccurate. For instance, the authors of [26] proposed an LVA approach that uses a dynamic predictor that is improved after each memory read. Additionally, their work offers a quality-energy knob where for a higher approximation level, the energy is reduced by decreasing the memory bandwidth. Another LVA method described in [34] is optimized for GPU architectures. The authors used an existing predictor, i.e., two-delta predictor [8], in their proposed approach. Similar to the work in [26], the authors of [34] provided a quality knob where the level of approximation could be controlled. Both the approaches in [26] and [34] uses a dynamic predictor that has to be updated periodically using values from the memory to minimize the error. Additionally, the predictor needs an extra hardware, e.g., lookup table, in addition to complex methods to extract and calculate the input arguments, e.g., hash values, to index the predictor. Alternatively, in [1] we performed a preliminary investigation where we introduced a method that entirely removes real-time memory accesses by profiling an error-tolerant application to instrument load values and context. The context includes a history of the program counter (PC) and memory addresses, among multiple hash values that encode the history of the store and load values and addresses. The instrumented data, i.e., load values, is analyzed for the occurrence of the *value locality*. Thereafter, if the instrumented data showed the existence of *value locality* it is used to train a machine learning (ML) model. Subsequently, the ML model is used in real-time as an LVA.

Even though the model we proposed in [1] addresses the main limitations of the LVA approaches in [26] and [34], it has the drawback of resource usage overhead to compute and store the hash values encoding the history of execution required by the ML-based LVA. Another limitation of [1] is its reliance on machine information, e.g., PC and memory address, for prediction. Since these vary across machines and architectures, this approach limits portability and makes the ML-based LVA machine and architecture dependent. Furthermore, the work in [1] failed to deliver a good quality in all the tested scenarios, e.g., the accuracy for the model trained for the *cannel* benchmark [4] did not exceed 37%. In a subsequent effort, in [2] we developed an improved model that does not require the computation of hash values and hence eliminating its overhead, however, it is only suitable for repetitive access to the same set of multimedia. Since the model proposed in [2] requires training for each set of multimedia, the overhead of training the ML model outstrips the savings of deploying the LVA when the multimedia is not used recursively. Subsequently, the usage of the LVA we proposed in [2] is limited in scope. In addition to the aforementioned limitations of our previous investigations, in [1] our preliminary investigation was limited to the testing of the accuracy of the trained ML-based LVA model , while in the subsequent investigation in [2] the analysis was limited to simulating the load process and measuring the quality of the approximation. In this paper, we address the limitations of all previous approaches by proposing a *static ML-based LVA* that does not need to be updated in real-time, i.e., the model is trained offline. Additionally the ML model is trained once and afterwards used for various sets of multimedia data. Furthermore, the model we present in this paper consist of a complete implementation of an ML-based LVA approach.

The rest of the paper is organized as follows. In Section 2 we review the most significant assessment methods of approximate computing. We describe the proposed model of ML-based load value approximation in Section 3. In Section 4, we present the implementation of the trained ML model in software, followed by the performance and quality analyses of the proposed LVA method in Sections 5 and 6, respectively. We compare the proposed approach with the state-of-the-art in Section 7 and conclude the paper in Section 8.

2 EVALUATING APPROXIMATE COMPUTING

An Approximate Computing (AC) design can be deemed usable if the delivered quality is within an acceptable range of tolerable error. Furthermore, the AC design must ensure savings in physical metrics, i.e., area, delay and/or power, otherwise the purpose of approximation is defeated. For this reason, in this section, we review important evaluation metrics of AC and quality assessment.

The generated quality of an AC design can be measured using different metrics. Some of these metrics include:

- **Error Distance (ED)** is the arithmetic distance between the exact value (E_v) and the approximate value (A_v) for a given set of inputs. Hence the ED can be written as: $ED = E_v - A_v$.
- **Relative Error Distance (RED)** is the ratio of the relative ED with respect to the exact value (E_v), i.e., $RED = \frac{ED}{E_v}$.
- **Mean Absolute Error (MAE)** is the average of the absolute values of all ED in space, i.e., $MAE = \left(\frac{\sum |ED|}{n}\right)$, where n is the number of instances.
- **Normalized Mean Absolute Error (NMAE)** is measured to have a better analysis for the worst-case scenario error. $NMAE$ is computed as $NMAE = \frac{MAE}{ED_{max}}$, where ED_{max} is the maximum ED in space, e.g., $ED_{max} = 255$ for 8-bit applications.
- **Mean Squared Error (MSE)** is the average of the ED squared, i.e., $MSE = \frac{\sum ED^2}{n}$.
- **Root Mean Squared Error (RMSE)** is the square root of MSE, i.e., $RMSE = \sqrt{MSE}$.
- **Normalized Root Mean Squared Error (NRMSE)** is computed in a similar fashion to NMAE, where $NRMSE = \frac{RMSE}{ED_{max}}$.
- **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 $PSNR = 20\ \log_{10}\left(\frac{255}{RMSE}\right)$.
- **Bit-Error Rate (BER)** is the percentage of faulty bits in the output. The BER can be expressed in terms of False Negative (FN), False Positive (FP), True Negative (TN) and True Positive (TP) as $BER = \frac{FN+FP}{FN+FP+TN+TP}$. The BER is different from all previously discussed error metrics since it disregards the arithmetic error.

- **Accuracy** is the overall proportion of correct predictions made out of all predictions. Accuracy can be computed as $Accuracy = 1 - BER = \frac{TN+TP}{FN+FP+TN+TP}$. Similar to BER, the accuracy also does not measure arithmetic error.
- **Precision** is the fraction of predicted positive instances that are indeed correct, emphasizing the accuracy in making positive predictions. Precision is computed as $Precision = \frac{TP}{TP+FP}$. Similar to BER and the accuracy metrics, precision does not measure arithmetic error.

The aforementioned error metrics are either magnitude-based or error-rate-based. 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 error-rate-based metrics are the suitable ones. On the other hand, physical metrics are also relevant when assessing an AC design which are measured quantitatively. General metrics such as area, delay and power, are usually multiplied to form composed metrics like power area delay product (PADP) and energy-delay product (EDP). These metrics are usually selected subjectively depending on the targeted field. For instance, some researchers opt for EDP which presents a dominance of the delay since the EDP can be written as $EDP = P \times D^2$, where P and D are the power and delay, respectively. Furthermore, execution time can be a more significant metric for software-based approximation since if the application is executed in a shorter period, the hardware consumes less energy. Thus, speedup in the execution of a software is also a relevant metric in approximate computing when assessing a software-based approximate implementation.

3 PROPOSED METHODOLOGY

To alleviate the challenges of existing LVA methods, in this paper, we propose an LVA that combines the advantages of previously proposed approaches. The key benefits of this LVA are: (*i*) static predictor, (*ii*) quality/performance trade-off in software, and (*iii*) simple predictor requiring minimal overhead.

The proposed LVA requires six steps as shown in Fig. 1. It has an offline phase where we perform the training of the ML-based LVA, the generation of the load predictor and synthesis of an approximated version of the error tolerant application while in the online phase the application is

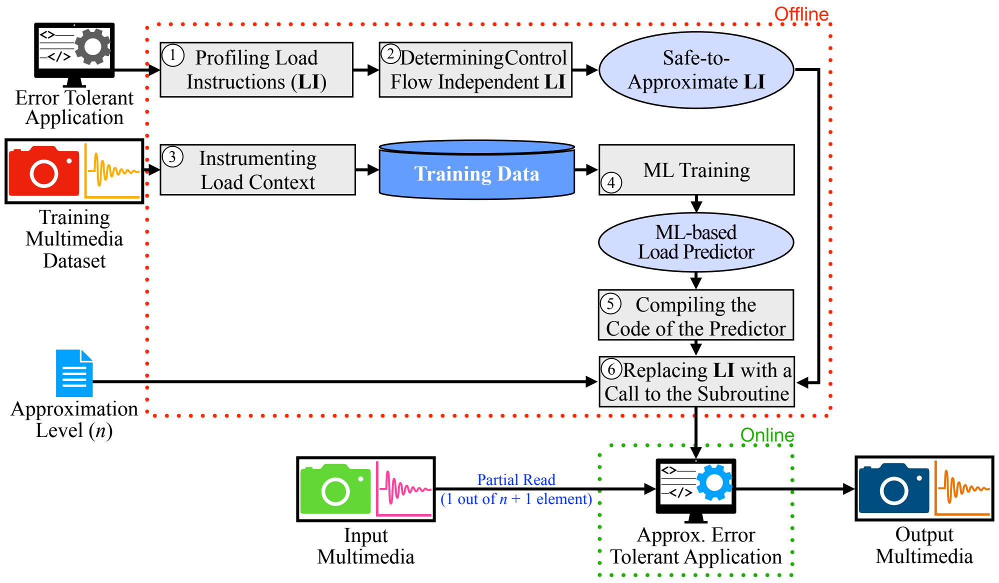

Offline
Error Tolerant Application
① Profiling Load Instructions (LI)
② Determining Control Flow Independent LI
Safe-to-Approximate LI
Training Multimedia Dataset
③ Instrumenting Load Context
Training Data
④ ML Training
ML-based Load Predictor
⑤ Compiling the Code of the Predictor
⑥ Replacing LI with a Call to the Subroutine
Approximation Level (n)
Online
Input Multimedia
Partial Read (1 out of n + 1 element)
Approx. Error Tolerant Application
Output Multimedia

Fig. 1. Proposed Load Value Approximation

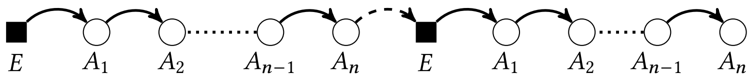


Fig. 2. Prediction Sequence

executed on multimedia inputs and generates approximated data. As shown in the figure, the offline phase accepts three inputs: (*i*) the training multimedia dataset, (*ii*) the error tolerant application and (*iii*) the approximation level. The training multimedia consists of images or audio files, the error tolerant application is an assembly program implementing a multimedia application such as image blending [31]. The approximation level defines the ratio of approximated load instructions, which is given by the user. It specifies the quality-performance trade-off where a higher approximation level results in a higher speed-up for potential decrease in quality. Based on these three inputs, the LVA produces an approximate executable code of the error tolerant application that is used in the online phase. The proposed LVA exploits the principal of *value locality* introduced in [19] to accurately approximate the value where the predicted load value is based on the most recent load value which can be exact or approximate. The example shown in Fig. 2 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.

The offline phase of the approximation process in the proposed LVA shown in Fig. 1 performs a profiling of the load instructions of the error tolerant application in step ①. In step ②, the load instructions that are not part of the control flow are identified as “safe-to-approximate” instructions. An example of a “safe-to-approximate" load instruction is a load that retrieves a pixel value. Contrarily, a load that retrieves a loop boundary is not, as it may cause a control flow hazard. In parallel, the training multimedia is fed to step ③ where the load context is instrumented to generate a training database. The load context instrumented is the value locality which is the adjacent element in the memory. Consequently, the database consists of two columns that contain a value and its adjacent value in the memory. The database is thereafter used in step ④ to generate an ML-based *load value predictor*. In the ML training, the first column is used as prediction argument, i.e., feature, while the second column, i.e., the adjacent element in memory, is the prediction value, i.e., target. In step ⑤ the ML-based *load value predictor* is compiled to become a subroutine that can be called by the assembly program to predict the load value. Subsequently, in step ⑥ the approximation level (n) is used to replace a given ratio of load instructions with a call to the ML-based *load value predictor* subroutine and hence producing approximate load instructions, i.e., LVA. For example, for $n = 19$, the LVA will consist of 1 exact load followed by 19 approximate load instructions, i.e., 1 out of 20 load instructions are exact. Thus for $n = 19$, we perform a 95% approximation. As shown in Fig. 1, step ⑥ generates a software-based implementation of the approximate application which is a modified assembly code of the error tolerant application where some of the load instructions are approximated. The code generated in step ⑥ is used in the online phase where the application can accept multimedia as input to generate an approximate output, e.g., the blending of two images.

The advantage of the proposed model is the usage of a static predictor that does not require new load values to update its prediction. Furthermore, the approximation level is user-specified and adjustable where with more approximation, i.e., higher value of n, a higher speed-up is achieved for a potential reduction in quality. Finally, the ML-based predictor used in this paper exploits the principle of *value locality* [19], where the subsequent value is predicted by knowing the previous value, i.e., previously used value either fetched from memory or generated by the ML-based predictor. To measure the quality and performance of the proposed methodology, we tested it using a variety of multimedia applications as described in the next section.

4 ML PREDICTOR IMPLEMENTATION

Optimizing the 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 paper 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. The implementation of the `predictor` subroutine is shown in Listing 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`.

Listing 1. Assembly Subroutine of the ML-based Predictor for x86 Architecture

```
 1  ; ECX contains the history value, i.e., the preceding value.
 2  ; Multiply by 6 the history value in ECX 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         . . .
15         # < skipped portion of the code>
16         . . .
17         mov  ecx , 240     ; if history value = 254
18         ret
19         mov  ecx , 220     ; if history value = 255
20         ret
```

5 QUALITY ANALYSIS

In this section, we apply the proposed LVA on six multimedia applications, namely, multiplication-based image blending [31], multiplication-based audio blending known as Ring Modulation (RM) [14], audio binarization known as infinite clipping [29], image binarization known as image thresholding [11], polarity inversion of audio [30] and image inversion known as image negatives [12]. We chose these applications to represent a practical usage of the proposed methodology and its implication on quality and performance. For instance, the blending is used in image editing tools [31] and in audio mixing tools [3]. Additionally, image binarization is used in optical character recognition (OCR) as it eliminates noise in a scanned image [33]. Furthermore, audio polarity inversion is widely used in active noise cancellation (ANC) systems where the polarity of the noise is inverted so that when combined with the original noise, the two cancel each other out through the *destructive interference* principle [7]. We used 4,094 images out of 8,189 images from the *Flowers* dataset [21] as “training multimedia” (Fig. 1) while the remaining 4,095 images from this dataset were used as multimedia input. Additionally, to diversify the objects in the images used in the multimedia input, we used the 8,041 images from the *Cars* dataset [18] and 16 images from the *Places* dataset [35]. Furthermore, we used 1,255 files of the audio dataset of Babylon 5 [32] as “training multimedia” while using the remaining 1,181 as multimedia input. 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. Among the chosen applications, binarization requires a parameter, i.e., threshold, while the others have fixed behavior. Subsequently, for the binarization, we determined a global threshold per dataset using *Otsu’s Method.* [22]. In a previous work [2], we have shown the superiority of the Extra Decision Trees (EDTs) [27] among other ML training algorithms which we adopt in this paper to generate the ML-based LVA. Finally, using the aforementioned multimedia we built two ML-based load value predictors one based on images and the other on audio files. Subsequently, each of the built ML-based load value predictors was tested on the same type of multimedia used for the training. In the remainder of this section, we will perform the quality analysis of the trained ML-based LVA when tested in various image and audio processing applications.

5.1 Image Processing

In the first experiment, we perform a blending of two *Flowers* together. In our second experiment, we test the training ML-based load value predictor in the blending of two *Cars* together. Finally, we perform a blending of 16 *Places* and 129 *Cars*. Additionally, using the same dataset we perform image inversion and image binarization. Throughout the various experiments, we tested the proposed LVA in more than 40 Million instances. All experimenting instances were tested for $n = 1$ to $n = 19$, i.e., 50% to 95% approximation. The quality of the resulting images of blending and inversion is assessed using PSNR, NMAE, and NRMSE. The quality of the binarized images is measured using the accuracy and precision metrics. The average for each of the metrics is depicted in Figs. 3 and 4. We can notice that the PSNR, NMAE and NRMSE range from 74.11 to 101.30, 0.50% to 10.42% and 1.07% to 15.67%, respectively. In multimedia applications, a PSNR greater than 40 dB is normally considered as very good [5]. Thus, we can conclude that the PSNR exceeds what is considered a very good quality in multimedia. We also notice from Fig. 3 that for a higher approximate level (n) the rate of change in the error is smaller, i.e, the curves flatten. This is because we approximated n out of $(n + 1)$ load instructions, i.e., approximation = $\frac{n}{n+1}$, which flattens for a large value of n. Furthermore, we can notice that the quality of image inversion outperformed the one of image blending. We attribute this difference to the nature of each application where image inversion involves a single input, whereas blending requires two images. Consequently, in image blending, errors from both

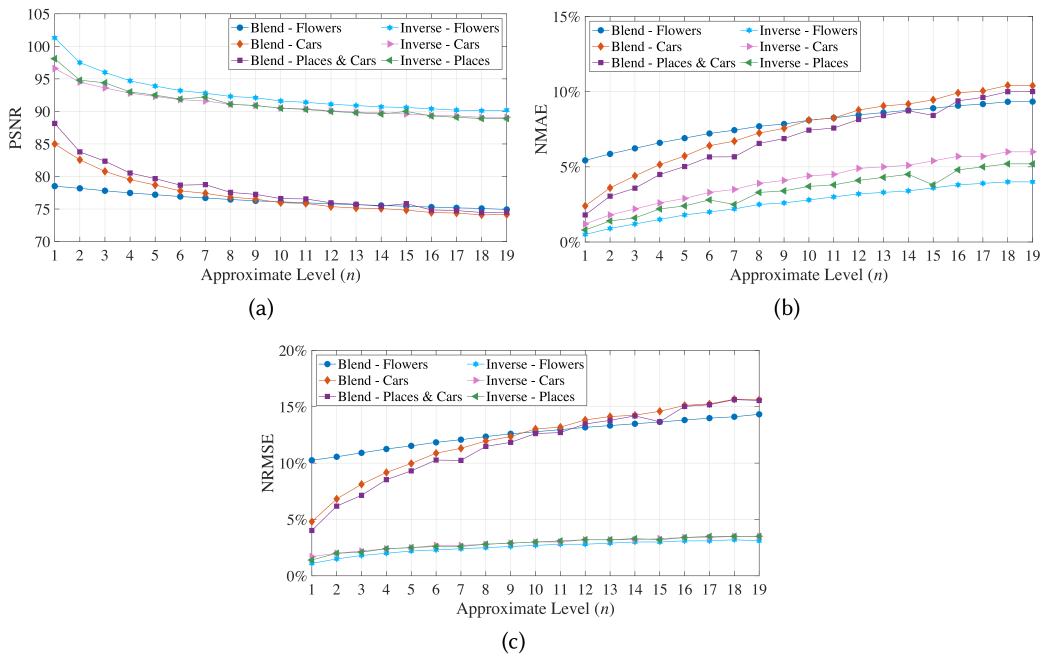


Fig. 3. (a) PSNR, (b) NMAE and (c) NRMSE for Various Image Blending and Image Inversion

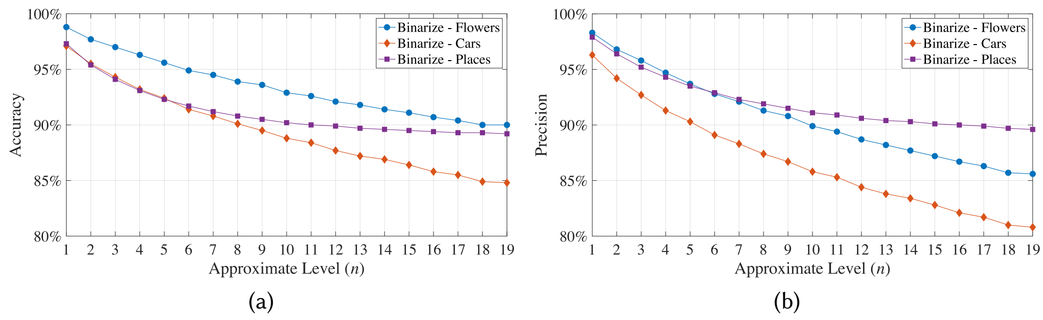


Fig. 4. (a) Accuracy and (b) Precision for Various Image Binarization

images accumulate, leading to a larger overall error in the combined result.Finally, as shown in Fig. 3 we can observe that the quality trend is very similar in a given application regardless of the dataset used in the experiment. For the image binarization experiment, we can notice from Fig. 4 that the maximum loss of accuracy and precision was 15.20% and 19.20%, respectively, which occurred for a 95% approximation, i.e., $n = 19$. On the other hand, for a 50% approximation the average accuracy and average precision were 97.72% and 97.49%, respectively. Additionally, we notice that for a 90% approximation the accuracy was on average 91.21%.

Analyzing the quality objectively, we can notice from Figs. 5 – 8 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 and at 80% approximation the quality is considered acceptable. However, at 90% approximation the quality loss becomes more visible yet still consumable if the performance gain is the ultimate goal.

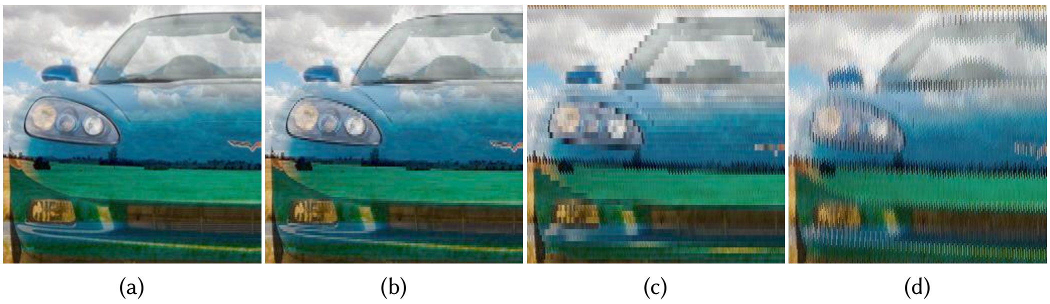


Fig. 5. Image Blending Example 1: (a) Exact, (b) 50% Approximation (n=1), (c) 80% Approximation (n=4) and (d) 90% Approximation (n=9)

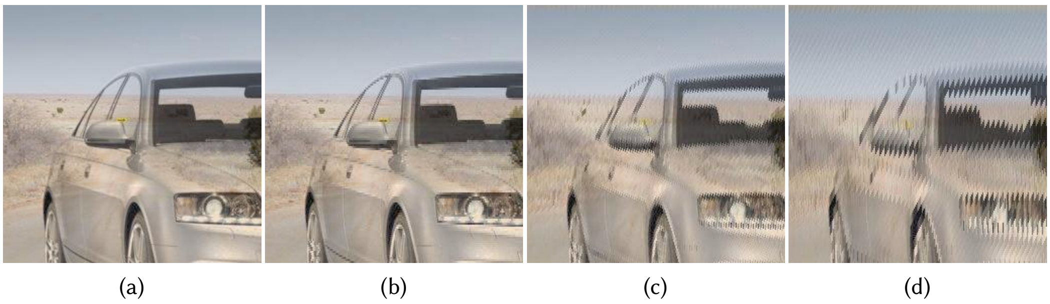


Fig. 6. Image Blending Example 2: (a) Exact, (b) 50% Approximation (n=1), (c) 80% Approximation (n=4) and (d) 90% Approximation (n=9)

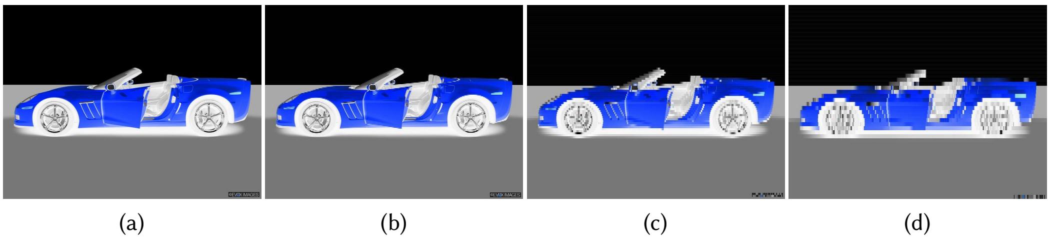


Fig. 7. Image Inversion Example: (a) Exact, (b) 50% Approximation (n=1), (c) 80% Approximation (n=4) and (d) 90% Approximation (n=9)

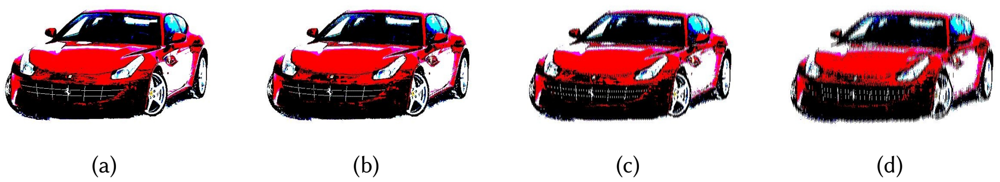
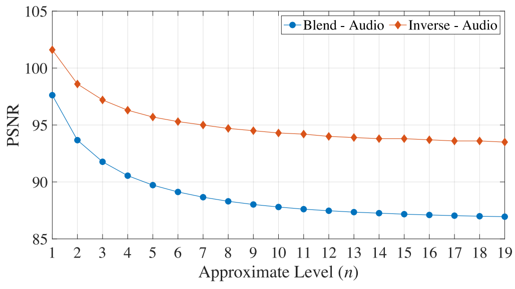


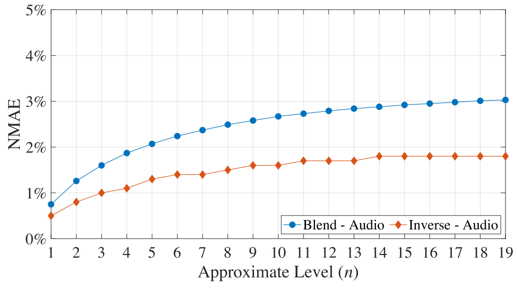
Fig. 8. Image Binarization Example: (a) Exact, (b) 50% Approximation (n=1), (c) 80% Approximation (n=4) and (d) 90% Approximation (n=9)

5.2 Audio Processing

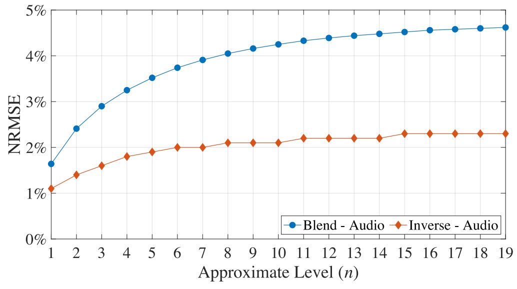
We tested the quality of the audio blending, audio inversion and audio binarization for all possible instances in the "application dataset". The results are summarized in Fig. 9. We can notice that the quality of the audio processing applications exceeds those achieved for image image counterparts where the PSNR, NMAE, and NRMSE range from 86.95 to 101.57, 0.45% to 3.03% and 1.09% to 4.62%. With the lowest PSNR being 86.95 *dB* for an approximate level (n) of 19, i.e., 95% approximation, the achieved quality when testing the proposed ML-based load value predictor is double the minimum value needed for the result to be considered good [5]. Additionally, from Figs. 9(b) and 9(c) we can notice that both metrics did not exceed 5% even for an approximation of 95%. Furthermore, we can notice that in the audio processing, the trend is similar to the one of image processing where inversion performed better than blending. Finally, from Fig. 10, we can notice that the accuracy and precision ranged from 89.30% to 97.30% and 89.78% to 97.93%, respectively. Therefore, the loss in accuracy and precision did not surpass 11% even with a 95% approximation. Subsequently, we note that in the audio binarization application, the quality was also superior to the one of images.



(a)

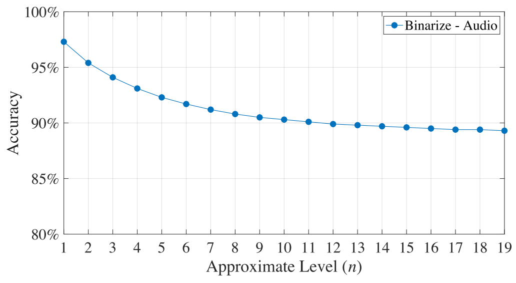


(b)

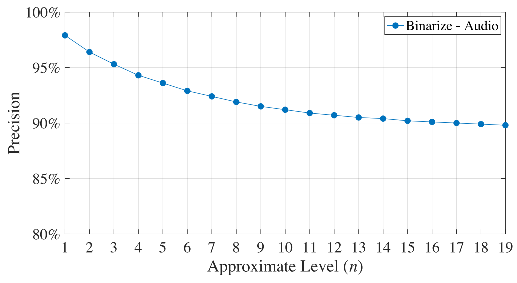


(c)

Fig. 9. (a) PSNR, (b) NMAE and (c) NRMSE for Various Audio Blending and Audio Inversion



(a)



(b)

Fig. 10. (a) Accuracy and (b) Precision for Various Audio Binarization

6 PERFORMANCE ANALYSIS

In order to analyze the speedup resulting from the proposed LVA, we use the GEM5 simulator [10], which allows cycle-accurate measurements of the software execution. 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. Subsequently, using GEM5 provides an ideal environment for analyzing precise execution behavior and performance. The CPU configuration used in GEM5 is based on the most recent trends in commercially available computers. We applied 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 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. 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 1 [15].

In this paper, 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 that are power of two. The various cache settings used in this paper are summarized in Table 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 powers of two values, i.e., 32KB and 64KB, the two variations

Table 1. Cache Settings of the Intel Processor [15]

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 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

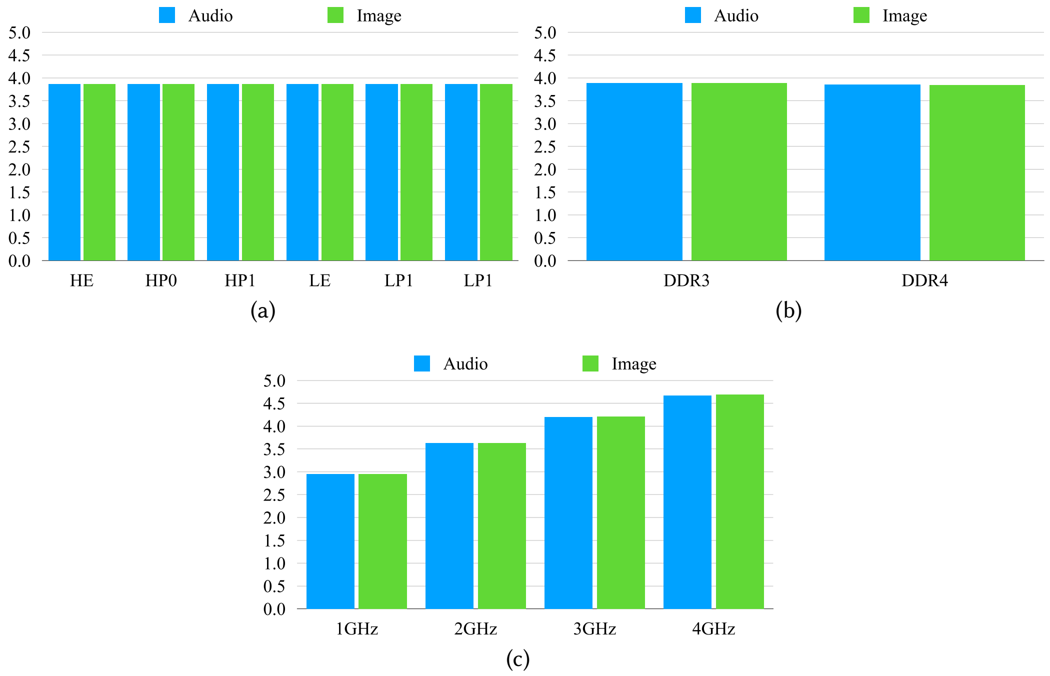


Fig. 11. Speedup in Memory Access when varying (a) Cache, (b) RAM and (c) CPU Frequency Settings

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, i.e., 1 GHz, 2 GHz, 3 GHz and 4 GHz. These values are chosen based on the base frequencies of the latest generation of Intel processors. Finally, we added a layer of variations in our CPU configuration in GEM5 where we selected 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”[9] where we opt for an architecture that has L1 (separate) and L2 (unified) caches only.

The 48 hardware configurations are tested at two levels, namely, the speedup achieved for the memory load operation only in the various applications and at the application level, i.e., overall speedup. For the performance analysis, we tested the six multimedia applications using 16 and 129 images from the *Places* and *Cars* dataset, respectively. Additionally, we select 65 audio files to analyze the performance of the audio processing applications when implementing the proposed ML-based load value approximator.[1] 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.

Fig. 11 depicts the average speedup in the memory operation achieved when varying one hardware configuration at a time. From Figs. 11(a) and 11(b), we can notice that the variations in the cache and DRAM configurations result in a minimal impact on the speedup achieved. On the other hand, we can notice in Fig. 11(c) that for a higher frequency, the proposed LVA achieves a higher

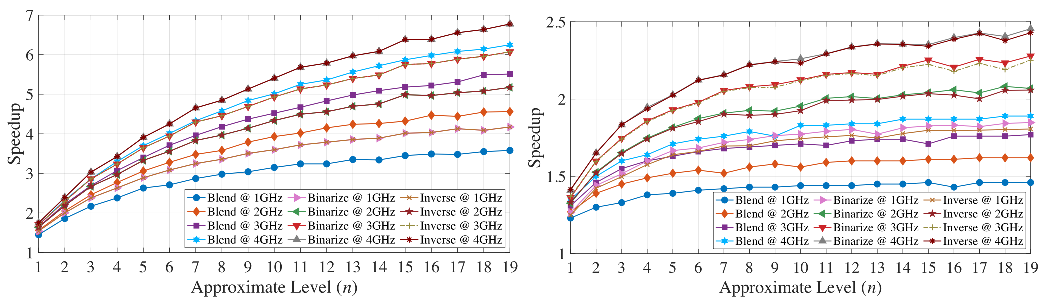
[1]Since the simulation in GEM5 requires an excessive amount of time and given the 48 hardware configurations tested, it is only possible to test a limited set of combinations even when using High-Performance Computing (HPC).

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. In the rest of this section, we will present the achieved speedup for the six applications when varying the frequency and the approximate level (n). However, due to the large number of simulation instances, we will limit the presentation to the average speedups achieved when varying the frequency.

6.1 Image Processing

Fig. 12 shows the speedups achieved for the various CPU frequencies used in the experiment when varying the approximate level (n) for the three selected image processing applications. The results demonstrated in Fig. 12 are the average speedup for each application for the various cache and DRAM configurations. The speedups achieved when n increases follows the same flattening trend observed in in the quality analysis. Furthermore, we can notice a higher speedup when the CPU is running on a faster frequency.

The speedups in memory access operations when using the proposed LVA achieved a range between 1.45 and 6.77 as shown Fig. 12(a). The minimum speedup occurred for $n = 1$, i.e., 50% approximation, and a frequency of 1 GHz while the maximum speedup was achieved for $n = 19$, i.e., 95% approximation, and a frequency of 4 GHz. Furthermore, for a 50% approximation, i.e., $n = 1$, an average speedup among the three applications of 1.51 and 1.73 is obtained for 1 GHz and 4 GHz, respectively. Furthermore, as presented previously, the proposed LVA achieved a higher gain for a higher frequency, i.e., higher speedup. In addition, an average speedup of 3.97 and 6.60 was achieved for $n = 19$ when the CPU runs at 1 GHz and 4 GHz, respectively. Examining the speedups at the application level we can notice as shown in Fig. 12(b) that the speedup in the application is less than the one achieved in the memory read operation. This is because memory access is a portion of the image processing application where the remaining portion, e.g., arithmetic operations, are executed precisely. Nonetheless, a speedup was achieved at the application level. From Fig. 12(b) we can notice an average speedup of 1.25, 1.31, 1.35 and 1.39 at an approximation of 50%, i.e., $n = 1$, for CPU frequencies of 1 GHz, 2 GHz, 3 GHz and 4 GHz, respectively. Additionally, we can notice that the highest speedups achieved in the execution of the application were for $n = 19$, i.e., 95% approximation, with average speedups among the three applications of 1.70, 1.92, 2.10 and 2.26 at 1, 2, 3 and 4 GHz, respectively. Finally, from the results shown in Fig. 12, we can notice that



(a) (b)

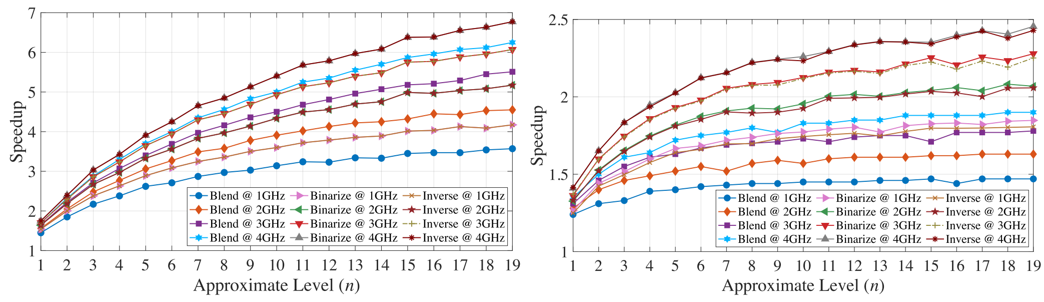
Fig. 12. Average Speedups at the (a) Memory and (b) Application Levels for Various Image Processing Applications

the binarization and inversion applications had a similar performance gain at the memory and application levels. Additionally, we notice that the two aforementioned applications attained a higher speedup compared to the blending application. The difference in memory access speed can be attributed to how the applications interact with memory. The blending application reads from two separate memory regions, while the binarization and inversion applications access only one. When accessing a single region, consecutive memory accesses will target the same memory bank and/or row, causing delays due to row activation and precharge overhead caused by the preceding memory access. In contrast, when accessing two distinct memory regions mapped to different banks, the memory controller can interleave operations, allowing the second access to proceed while the first is completing, reducing overall access latency. Consequently, fetching two elements from separate images stored in different banks is often faster than accessing two consecutive elements that reside in different DRAM rows within the same bank. Hence, the memory access in the binarization and inversion is slower compared to the blending application and applying the proposed LVA led to a better speedup. On the other hand, the speedup at the application level was higher for binarization and inversion, as they require fewer arithmetic operations compared to blending. In image blending, memory access constitutes a smaller fraction of the total executed instructions, resulting in a lower speedup gain. Conversely, in binarization and inversion, where memory access plays a more significant role, the proposed LVA yields a higher speedup.

6.2 Audio Processing

For the audio applications, we also tested for the various configurations of hardware in GEM5. Fig. 13 depicts the average speedups achieved per audio processing application for the various DRAM and cache configurations when varying the approximate level (n) and the frequency. Figs 13(a) and 13(b) show the speedup achieved in the memory access and the overall application, respectively. From the two measurements shown in Fig. 13, we can notice that for a higher approximate level (n), a higher speedup is achieved. Additionally, similar to the performance of image processing applications we can notice that for a higher approximate level the speedup achieved flattens. From Fig. 13(a) we can notice that the average speedup in memory access operation ranges from 1.45 to 6.77. The minimum speedup was achieved for $n = 1$ and a frequency of 1 GHz while the maximum speedup occurred for the highest approximate level, i.e., $n = 19$, where the CPU is running at a frequency of 4 GHz. We can also notice from Fig. 13(a) that for $n = 19$ the speedup achieved at 4 GHz was almost twice as much as the one achieved when the processor is running at 1 GHz.

The observed trend of increasing n and the frequency resulting in a higher speedup is also observed in the speedup in the execution of the application. However, as shown in Fig. 13(b), we



(a) (b)

Fig. 13. Average Speedups at the (a) Memory and (b) Application Levels for Various Audio Processing Applications

can notice a similar trend to the speedup observed in the image processing applications where the speedup at the application level, i.e., overall speedup, is less than the one achieved when only considering the memory traffic shown in Fig. 13(a). The average speedup in the execution of the audio processing applications ranges from 1.25 to 1.71, 1.31 to 1.92, 1.35 to 2.10 and 1.39 to 2.26 when the CPU is running at frequencies of 1 GHz, 2 GHz, 3 GHz and 4 GHz, respectively. Finally, from Figs. 13(a) and 13(b) we notice that the audio binarization and inversion gained more speedup using the proposed LVA. This trend is similar to the one observed in the image processing applications.

7 COMPARISON WITH RELATED WORK

We compare the proposed ML-based LVA to the state-of-the-art LVA proposed in [26, 34]. Since these related work use different error metrics, we will compare to each of these work separately. For instance, the authors of [26] used the NMAE as an error metric and the comparison with their work is summarized in Table 3. We can notice that the NMAE of the work in [26] is more than double for the various approximate levels. Additionally, the proposed LVA provides a much higher speedup for each of the approximate levels. For instance, for $n = 17$, the LVA proposed in [26] achieved an average speedup of 1.08 over all the experiments conducted. In contrast, for the same approximation 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. Since the proposed LVA delivers a better quality and a higher speedup than the LVA proposed in [26] for the various approximate levels, we can conclude that the LVA we propose is superior.

Table 3. Comparison of NMAE and Speedup of the proposed LVA with [26]

Approximate Level (n)	**LVA [26]**		**Proposed LVA**	
	NMAE	**Speedup**	**NMAE**	**Speedup**
1	5.81%	1.08	1.67%	1.32
3	7.25%	1.07	2.73%	1.62
5	8.97%	1.08	3.52%	1.76
9	11.06%	1.08	4.57%	1.87
17	13.78%	1.08	6.03%	1.98

The authors of [34] proposed the Rollback Free Value Predictor (RFVP) and used the NRMSE as an error metric and the comparison with the proposed LVA is summarized in Table 4. Even though the authors of [34] targeted GPU architectures with different approximation 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 4 shows the normalized root mean squared error (NRMSE) for the various approximate levels. From Table 4 we can notice that for any approximate level, our model provides at least 3.75× better quality and hence outperforms the work in [34].

Table 4. Comparison of NRMSE and Speedup of the proposed LVA with [34]

Approximate Level (n)	**RFVP [34]**		**Proposed LVA**	
	NRMSE	**Speedup**	**NRMSE**	**Speedup**
1	12.21%	1.32	3.25%	1.32
3	18.65%	1.80	4.60%	1.62
4	23.46%	2.01	5.10%	1.79
9	31.25%	3.05	6.43%	1.87

8 CONCLUSION

Approximate computing (AC) has gained traction as an alternative computing technique that offers savings in area and power consumption. Previously, explorations of AC techniques have focused on arithmetic operations which are beneficial for computation-intensive tasks. However, for memory-demanding applications, the *memory wall* would still curb the performance. To address this challenge, we proposed in this paper a load value approximator (LVA) that predicts the value using machine learning (ML) and requires minimal overhead. Unlike previously proposed ML-based LVA techniques, the model we proposed is architecture and machine independent. The proposed model was evaluated using six multimedia applications for various approximation levels (n), i.e., 50% to 95% of load instructions were approximated. Using the proposed LVA, the memory access was 1.45× to 6.77× faster with an average speedup of 4.11×. The speedup in memory access reflected an overall speedup ranging from 1.23 to 2.45 where on average the applications ran at 1.83× faster. The average NMAE, i.e., average relative error, was 4.54% which is deemed as an acceptable error in approximate computing. The lowest value of peak signal-to-noise ratio (PSNR) in all conducted experiments was 74.11 dB. This value of PSNR is also deemed acceptable for multimedia applications. Since the proposed model was able to achieve a decent quality when 95% of the load instructions were approximated, we consider the proposed ML-based LVA as an efficient technique to be used when processing multimedia applications.

As a future work, we plan to experiment with the proposed methodology using a variable approximate level (n) in a dynamic fashion for a fine-tune in quality/energy trade-off. We also plan to test the proposed LVA on different architectures, e.g., RISC-V and ARM, to evaluate the speedup benefits and examine any limitations or differences in behavior compared to x86. Furthermore, we plan to analyze the performance gain and energy saving when implementing the proposed methodology in hardware.

ACKNOWLEDGMENTS

This research was enabled in part by support provided by Calcul Québec (calculquebec.ca) and the Digital Research Alliance of Canada (alliancecan.ca)

REFERENCES

[1] Alain Aoun, Mahmoud Masadeh, and Sofiène Tahar. 2023. A Machine Learning Based Load Value Approximator Guided by the Tightened Value Locality. In *Great Lakes Symposium on VLSI*. ACM, 679–684. https://doi.org/10.1145/3583781.3590207

[2] Alain Aoun, Mahmoud Masadeh, and Sofiène Tahar. 2024. Machine Learning based Memory Load Value Predictor for Multimedia Applications. In *International Conference on Microelectronics*. IEEE, 1–6. https://doi.org/10.1109/ICM63406.2024.10815775

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

[4] Christian Bienia. 2011. *Benchmarking Modern Multiprocessors*. Ph. D. Dissertation. Princeton University, USA.

[5] David R. Bull and Fan Zhang. 2021. Digital picture formats and representations. In *Intelligent Image and Video Compression*. Oxford Academic Press, Chapter 4, 107–142. https://doi.org/10.1016/B978-0-12-820353-8.00013-X

[6] Luis Ceze, Karin Strauss, James Tuck, Josep Torrellas, and Jose Renau. 2006. CAVA: Using checkpoint-assisted value prediction to hide L2 misses. *ACM Transactions on Architecture and Code Optimization* 3, 2 (June 2006), 182–208. https://doi.org/10.1145/1138035.1138038

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

[8] Richard J. Eickemeyer and Stamatis Vassiliadis. 1993. A load-instruction unit for pipelined processors. *IBM Journal of Research and Development* 37, 4 (1993), 547–564. https://doi.org/10.1147/rd.374.0547

[9] GEM5. 2024. gem5::TimingSimpleCPU Class Reference. https://doxygen.gem5.org/release/current/classgem5_1_1TimingSimpleCPU.html

[10] GEM5. 2024. The gem5 Simulator System. https://www.gem5.org

[11] Rafael Gonzalez and Richard Woods. 2017. Image Segmentation. In *Digital Image Processing Global Edition* (4 ed.). Pearson, Chapter 10, 699–810. https://www.pearson.com/store/p/digital-image-processing-global-edition/GPROG_A101708555905_learnernz-availability/9781292223049

[12] Rafael Gonzalez and Richard Woods. 2017. Intensity Transformations and Spatial. In *Digital Image Processing Global Edition* (4 ed.). Pearson, Chapter 3, 119–202. https://www.pearson.com/store/p/digital-image-processing-global-edition/GPROG_A101708555905_learnernz-availability/9781292223049

[13] Vaibhav Gupta, Debabrata Mohapatra, Anand Raghunathan, and Kaushik Roy. 2012. Low-power digital signal processing using approximate adders. *IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems* 32, 1 (January 2012), 124–137. https://doi.org/10.1109/TCAD.2012.2217962

[14] Jeffrey Hass. 2021. Synthesis. In *Introduction to Computer Music*. Indiana University, Chapter 4. https://cmtext.indiana.edu/synthesis/chapter4_am_rm.php

[15] Intel. 2024. 13th Generation Intel® Core™ and Intel® Core™ 14th Generation Processors – Datasheet, Volume 1 of 2. 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/

[16] Dong-Ik Jeon, Kyeong-Bin Park, and Ki-Seok Chung. 2017. HMC-MAC: Processing-in Memory Architecture for Multiply-Accumulate Operations with Hybrid Memory Cube. *IEEE Computer Architecture Letters* 17, 1 (May 2017), 5–8. https://doi.org/10.1109/LCA.2017.2700298

[17] Honglan Jiang, Francisco Javier Hernandez Santiago, Hai Mo, Leibo Liu, and Jie Han. 2020. Approximate arithmetic circuits: A survey, characterization, and recent applications. *Proceedings of the IEEE* 108, 12 (December 2020), 2108–2135. https://doi.org/10.1109/JPROC.2020.3006451

[18] Jonathan Krause, Michael Stark, Jia Deng, and Li Fei-Fei. 2013. 3D Object Representations for Fine-Grained Categorization. In *International Conference on Computer Vision Workshops*. IEEE, 554–561. https://doi.org/10.1109/ICCVW.2013.77

[19] Mikko H. Lipasti, Christopher B. Wilkerson, and John Paul Shen. 1996. Value locality and load value prediction. *SIGPLAN Notices* , Volume 31, Issue 9 (September 1996), 138–147. https://doi.org/10.1145/248209.237173

[20] Duy Thanh Nguyen, Nguyen Huy Hung, Hyun Kim, and Hyuk-Jae Lee. 2020. An approximate memory architecture for energy saving in deep learning applications. *IEEE Transactions on Circuits and Systems I: Regular Papers* 67, 5 (May 2020), 1588–1601. https://doi.org/10.1109/TCSI.2019.2962516

[21] Maria-Elena Nilsback and Andrew Zisserman. 2008. Automated flower classification over a large number of classes. In *Indian Conference on Computer Vision, Graphics & Image Processing*. IEEE, 722–729. https://doi.org/10.1109/ICVGIP.2008.47

[22] Nobuyuki Otsu. 1979. A Threshold Selection Method from Gray-Level Histograms. *IEEE Transactions on Systems, Man, and Cybernetics* 9, 1 (January 1979), 62–66. https://doi.org/10.1109/TSMC.1979.4310076

[23] David A. Patterson and John L. Hennessy. 1994. *Computer Organization and Design: The Hardware/Software Interface*. Morgan Kaufmann. https://www.sciencedirect.com/book/9781483207759/

[24] Ashish Ranjan, Arnab Raha, Vijay Raghunathan, and Anand Raghunathan. 2017. Approximate memory compression for energy-efficiency. In *International Symposium on Low Power Electronics and Design*. IEEE, 1–6. https://doi.org/10.1109/ISLPED.2017.8009173

[25] Glenn Reinman and Brad Calder. 1998. Predictive techniques for aggressive load speculation. In *International Symposium on Microarchitecture*. IEEE, 127–137. https://doi.org/10.1109/MICRO.1998.742775

[26] Joshua San Miguel, Mario Badr, and Natalie Enright Jerger. 2014. Load value approximation. In *International Symposium on Microarchitecture*. IEEE, 127–139. https://doi.org/10.1109/MICRO.2014.22

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

[28] Alireza Senobari, Jafar Vafaei, Omid Akbari, Christian Hochberger, and Muhammad Shafique. 2024. A Quality-Aware Voltage Overscaling Framework to Improve the Energy Efficiency and Lifetime of TPUs Based on Statistical Error Modeling. *IEEE Access* 12 (July 2024), 92181–92197. https://doi.org/10.1109/ACCESS.2024.3422012

[29] Eric Tarr. 2018. 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, Chapter 10, 147–182. https://doi.org/10.4324/9781351018463

[30] Eric Tarr. 2018. Signal Gain and DC Offset. In *Hack Audio: An Introduction to Computer Programming and Digital Signal Processing in MATLAB*. Taylor & Francis, Chapter 6, 57–78. https://doi.org/10.4324/9781351018463

[31] Scott Valentine. 2012. List of Blend Modes. In *Hidden Power of Blend Modes in Adobe Photoshop*. Adobe Press, Chapter 7, 145–179. https://www.peachpit.com/store/hidden-power-of-blend-modes-in-adobe-photoshop-9780321823762

[32] Jim Ward. 2014. Sound Archive for Babylon 5. http://b5.cs.uwyo.edu/bab5/

[33] James M. White and Gene D. Rohrer. 1983. Image Thresholding for Optical Character Recognition and Other Applications Requiring Character Image Extraction. *IBM Journal of Research and Development* 27, 4 (July 1983), 400–411. https://doi.org/10.1147/rd.274.0400

[34] Amir Yazdanbakhsh, Gennady Pekhimenko, Bradley Thwaites, Hadi Esmaeilzadeh, Onur Mutlu, and Todd C Mowry. 2016. RFVP: Rollback-free value prediction with safe-to-approximate loads. *ACM Transactions on Architecture and Code Optimization* 12, 4 (January 2016), 1–26. https://doi.org/10.1145/2836168

[35] Bolei Zhou, Agata Lapedriza, Jianxiong Xiao, Antonio Torralba, and Aude Oliva. 2014. Learning deep features for scene recognition using places database. In *International Conference on Neural Information Processing Systems, Volume 1*. MIT Press, 487–495. https://dl.acm.org/doi/10.5555/2968826.2968881