Approximate Computing for ML: State-of-the-art, Challenges and Visions

Georgios Zervakis
Karlsruhe Institute of Technology
Karlsruhe, Gergmany
georgios.zervakis@kit.edu

Hassaan Saadat
University of New South Wales
Sydney, Australia
h.saadat@unsw.edu.au

Hussam Amrouch
University of Stuttgart
Stuttgart, Gergmany
amrouch@iti.uni-stuttgart.de

Andreas Gerstlauer
University of Texas
Austin, U.S.A
gerstl@ece.utexas.edu

Sri Parameswaran
University of New South Wales
Sydney, Australia
sri.parameswaran@unsw.edu.au

Jörg Henkel
Karlsruhe Institute of Technology
Karlsruhe, Gergmany
henkel@kit.edu

ABSTRACT

We present a survey of approximate techniques that cover the main pillars of approximate computing research. Our analysis considers both static and reconfigurable approximation techniques as well as operation-specific approximate components (e.g., multipliers) and generalized approximate high-level synthesis approaches. As our main application target, we discuss the improvements that such techniques bring on machine learning and neural networks. In addition to the conventionally analyzed performance and energy gains, we also evaluate the improvements that approximate computing brings in the operating temperature.

CCS CONCEPTS

• **Hardware → Logic circuits**; **High-level and register-transfer level synthesis**; **Hardware accelerators**.

KEYWORDS

Approximate Computing, Architecture, Accelerator, High-Level Synthesis, Inference, Logic, Low-power, Multiplier, Neural Network, Renconfigurable Accuracy, Temperature

ACM Reference Format:
Georgios Zervakis, Hassaan Saadat, Hussam Amrouch, Andreas Gerstlauer, Sri Parameswaran, and Jörg Henkel. 2021. Approximate Computing for ML: State-of-the-art, Challenges and Visions. In *ASP-DAC 2021: ACM Asia South Pacific Design Automation Conference, June 18–21, 2021, Tokyo, Japan.* ACM, New York, NY, USA, 8 pages. https://doi.org/10.1145/1122445.1122456

1 INTRODUCTION

Leveraging the intrinsic error resilience of a vast number of application domains, approximate computing has emerged as a promising design alternative to boost the efficiency of computing systems. Approximate computing introduces error or quality as a new design metric to be traded off for energy/power reduction and/or better performance. Typical paradigms of approximate computing applications are image processing and machine learning domains. Specifically, recent breakthroughs in Neural Networks (NNs) have brought significant advancements in machine learning [1]. These rapid advancements, however, came at the cost of an immense demand for computational power. Hence, to bring the inference speed to an acceptable level, custom ASIC Neural Processing Units (NPUs) are becoming ubiquitous in both embedded and general purpose computing. NPUs perform several tera operations per second in a confined area. Therefore, they become subject to elevated on-chip power densities that rapidly result in excessive on-chip temperatures during operation [2]. As a result, radical changes to conventional computing approaches are required in order to sustain and/or improve performance while satisfying mandatory energy and temperature constraints. Hence, approximate computing transforms from a design alternative to an obligation, especially when considering emerging domains such as machine learning and specifically neural networks.

Driven by this high potential for power reduction, designing approximate circuits has attracted significant research interest. At the custom hardware level, approximate computing targets mainly arithmetic units [3–6] (e.g., adders and multipliers) since they form the core components of all computations and a vast number of error-tolerant applications. Specifically, in NN inference the majority of the energy is consumed in the multiplication operations. Recent research showed that employing approximate multipliers in NN inference can deliver significant energy savings for a minimal loss in accuracy [6–8]. However, designing approximate circuits under quality constraints heavily increases the design time cycle since the designer has to verify both functionality and optimality as well as operating within error bounds [9]. This task becomes even more challenging as the circuit's complexity increases. To this end, several research activities, such as approximate high-level synthesis (AHLS) [10], focus on automating the generation of approximate circuits. Approximate HLS estimates error propagations and distributes the available error budget to the different approximate sub-components of a larger accelerator, such as convolution operators and generic matrix multiply units. As a result, AHLS enables generating complex approximate micro-architectures that satisfy given quality requirements.

Moreover, approximate computing is further subdivided into static and dynamically reconfigurable approximation techniques. The latter, leveraging that error-tolerance and the induced errors

ASP-DAC 2021, June 18–21, 2021, Tokyo, Japan

ACM ISBN 978-1-4503-XXXX-X/18/06…$15.00
https://doi.org/10.1145/1122445.1122456

are context- and input-dependent, aim to improve accuracy by providing a fine grain quality control and/or to further boost (power, energy, and/or delay) gains by applying more aggressive approximation on less-sensitive inputs. Finally, reconfigurable approximation was also recently applied to address thermal constraints [2]. Instead of addressing thermal emergencies by reducing performance, by reducing the accuracy and hence dynamic power in the same area, the circuit's power density decreases, resulting in lower temperatures.

In this paper, we study state-of-the art approaches in each of the aforementioned categories and analyze their application in machine learning and neural network domains. In Section 2, we first evaluate approximate multipliers [6] at the component level and in Section 3, we then focus on AHLS [10] approaches targeting approximate design automation at the complete processor or accelerator level. Section 4 further examines neural network specific runtime reconfigurable approximation techniques that target energy and/or temperature optimization. Finally, in Section 5, we discuss the challenges, limitations, and open issues of approximate computing applications in the machine learning domain.

2 APPROXIMATE COMPONENTS

Multiply-and-accumulate (MAC) operations are the fundamental and dominant computations in neural network algorithms. It has been reported that the MAC operations consume nearly 99% energy in DNNs [11]. Within a MAC unit, a multiplier is more complex and resource-hungry than the adder. Therefore, approximating the hardware multiplier units is one of the promising avenues for achieving overall system efficiency. In approximation of hardware arithmetic units, the logic circuit of the arithmetic unit is simplified such that it becomes faster, smaller, and/or less power/energy-hungry while producing erroneous outputs. In other words, the accuracy of the arithmetic unit is considered as another dimension of the design space, which is compromised for gains in all other design dimensions. This type of approximation is specifically referred to as the functional approximation of hardware units. Since the neural network algorithms are error-resilient, these erroneous outputs at the component-level have negligible effect on the application level accuracy [12].

2.1 Design Goals for Approximate Multipliers

When designing an approximate arithmetic unit, the primary goal is to achieve most resource-efficiency with least error (inaccuracy)[1]. In addition to this primary goal, it is desirable to follow a few other design considerations, as explained below.

- *Error-configurability:* Different applications and their use-cases may have differing degrees of error resilience and acceptable levels of output degradation [14]. These two factors demand that the approximate multiplier design should be *error-configurable*, which allows the system designer to tradeoff the degree of approximation with resource-efficiency gains for the desired output quality in a given application. The error-configurability can be design-time or run-time. This section focuses on the approximate multiplier designs with design-time reconfigurability. The run-time reconfigurability is discussed in Section 4.
- *Low Error Bias:* An approximate arithmetic unit produces outputs which may not be correct for several input combinations. The outputs can be greater (positive error), or smaller (negative error) than the correct value. Preferably, the approximate arithmetic unit should produce both negative and positive errors in a balanced manner for the various input combinations . This leads to cancellation of errors in successive computations instead of accumulation. This capability of an arithmetic unit is referred to as low error bias.
- *Systematic Approaches for the Introduced Approximations:* In recent years there is a demand for systematic approaches in approximate computing [15]. Specifically, the approximation in arithmetic units should be based on mathematical formulation instead of ad-hoc based modification/simplification of logic design [16].

[1]The error is typically characterized using various error metrics [6, 13].

2.2 Design-time Functionally Approximate Multipliers

As discussed earlier, the error-configurability in the approximate multipliers can be design-time or run-time. In general, one key advantage of the functional approximation (both design-time and run-time) is that it can be incorporated in the system at RTL level while utilizing the traditional synthesis and HLS tools, standard-cell CMOS libraries, and/or semiconductor fabrication technologies. In other words, it can be deployed in computing industry straightaway to meet the growing computing demands without significant investment in technology development.

While the design-time approximation implies that the approximate level is fixed once the system is synthesized/fabricated, its most prominent advantage is that it can offer improvements in area, latency, and energy/power consumption, i.e., improvement in all design parameters at the same time. This means that, for a given area and power budget, more compute power can be packed resulting in greater throughput when compared to using accurate multipliers. Alternately, to improve the total power consumption as well as the power density, the approximate multipliers (with lower area and delay) can be operated at lower frequency and voltage for iso-performance with slightly reduced accuracy. Another possible approach to improve the power density is to use slightly larger technology nodes, such as 45nm. Because the approximate components require fewer logic real-estate, the overall silicon footprint can be same with lower power density than when compared to using accurate multipliers on a smaller technology node.

A neural network generally has two phases: training, and inference. Recent research has shown that the inference phase can be implemented using low-precision fixed-point data format, which requires integer arithmetic units at the hardware level. On the other hand, the training phase demands high dynamic range to represent the intermediate data (e.g., the gradients), and thus floating-point format is needed. Therefore, design of resource-efficient approximate integer as well as approximate floating-point multipliers is required to boost the resource-efficiency of ML systems.

2.3 Approximate Integer Multipliers

Over the last decade, several designs for functionally approximate multipliers have been proposed. While a majority of them are error configurable (design-time), most of them are either based on ad-hoc mechanisms or do not have low error bias. A summary of some popular works depicting their features are shown in Table 1. The table depicts that the Mitchell's multiplier, MBM, ImpLM, and

Table 1: Approximate integer multipliers and their features.

Approximate Multiplier Design	Error Configurable	Low Error Bias	Design involves Mathematical Formulation
ETM [17]	✓	✗	✗
AWTM [18]	✓	✗	✗
UDM [19]	✓	✗	✗
DRUM [20]	✓	✓	✗
SSM [21]	✓	✗	✗
Mitchell [22]	✗	✗	✓
MBM [6]	✓	✓	✓
ImpLM [23]	✓	✓	✓
REALM [16]	✓	✓	✓

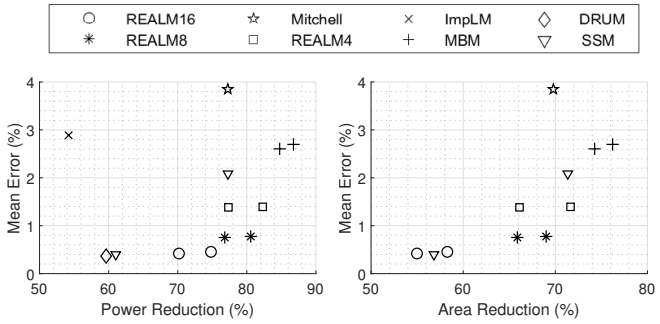
REALM are based on mathematical formulation. These multipliers are the approximate log-based multipliers, which are based on the log multiplication property.

The Mitchell's multiplier is the most basic approximate log based multiplier. It approximates log of the input operands by linear approximation between each power-of-two-interval of the operands, and then adds and scales them to get the approximate product. However, it is non-configurable and has high error bias. The ImpLM design improves the error in Mitchell's multipliers by modifying the approximation of the log curve. The MBM and REALM multipliers were designed by incorporating error reduction mechanisms in the Mitchell's approximate log-based multiplier while addressing the issue of non- configurability and high error bias.

These approximate-log multipliers along with a few other popular approximate multipliers are compared in Figure 1, where mean error (also referred to as MRED [13]) is plotted against area and power reduction percentages (calculated with respect to an accurate integer multiplier). The Figure shows that the REALM and MBM multipliers, while supporting the desirable features discussed above, also provide Pareto Optimal points in the design space.

2.4 Approximate Floating-point Multipliers

A floating-point (FP) multiplier is typically more complex and hence more resource-hungry than its integer counterpart. The FP multiplier consists of various sub-operations: mantissa multiplication, exponent addition, rounding, normalization, and other logic such as exception handling. Among these, the mantissa multiplier (which is an unsigned integer multiplier) is the most resource-hungry component, as it contributes to nearly 90% of the total area and power consumption in an IEEE single-precision FP multiplier [6, 24].



(a) Power-reduction vs. Mean Error (b) Area-reduction vs. Mean Error

Figure 1: Comparison of several approx. integer multipliers.

Table 2: Selected approx. FP multipliers compared with IEEE Single Precision multiplier and AlexNet errors.

FP Multiplier	Improvement per Multiplication		Error Rate	
	Power	Area	Top-1	Top-5
IEEE Sing. Prec.	*ref.*	*ref.*	42.8%	19.6%
AFDM-0	1.3×	1.4×	42.8%	19.6%
AFDM-10	4.1×	4.1×	42.8%	19.6%
AFMB-10	33.4×	18.6×	42.2%	19.8%
AFMB-20	123.3×	32.1×	41.8%	20.2%

Hence, several researchers have targeted approximation of mantissa multiplier for improving the overall resource-consumption of the FP multiplier. The traditional method of reducing the complexity of mantissa multiplication is to truncate the LSBs of the mantissa (also referred to as precision scaling). In recent year, some research works have used approximate integer multipliers as mantissa multipliers and demonstrated that it can yield better trade-offs than traditional precision scaling method [6, 25].

To this end, the approximate log-based integer multipliers are preferable than the other approximate multiplier schemes [6]. Since the mantissa in FP multiplication is normalized, the leading-one is always fixed at the MSB position. Therefore the leading-one detection and barrel shifting logic in the approximate log-based multipliers can be removed without any accuracy degradation resulting in significant efficiency gains. An example of such approximate multipliers are the AFMB proposed in [6]. A few of the error configurations of these approximate FP designs synthesized at 1GHz using TSMC 45nm library are compared with the IEEE single-precision FP multiplier in Table 2.

2.5 Example: CNN Inference

Consider a software implementation of the inference phase of AlexNet CNN, pre-trained using the ILSVRC-2012 dataset. First, we performed classification using IEEE single-precision FP arithmetic. Next, we replaced each FP multiplication in the implemented CNN with the software-based functional models of a few approximate FP multipliers (AFMB-t and AFDR-t [6]) and performed classification over the same set of images.

The Top-1 and Top-5 error rate results using each of the selected FP multipliers are depicted in Table 2, along with the power and area improvements per FP multiplier. The table shows that the classification errors, when using the approximate FP multipliers, are very close to the classification errors obtained when using the IEEE single-precision FP arithmetic. Thus, the approximate FP multipliers can reduce the power and area contributed by the FP multipliers to the whole system up to 123× and 32×, respectively, with negligible degradation in the classification accuracy of AlexNet.

3 APPROXIMATE HIGH-LEVEL SYNTHESIS

Approximate adders and multipliers as described in the previous section provide the combinational building blocks for approximate datapath, processor and accelerator designs. Approximations can in general be achieved by simply reducing the bit precision, by more intelligent approximating, or by even completely eliminating individual datapath operations at the expense of accuracy in final outputs. As described previously, in case of machine learning and neural network applications specifically, such approaches have been widely applied to trade off prediction accuracy for energy and

complexity savings. Traditionally, however, corresponding accelerators have been hand-crafted where approximations are applied uniformly across all operations and functional units in an application or design, e.g. as shown in the example in Section 2.5.

In a sequential design, the impact of approximations applied to individual operations depends on their interaction with other approximations in a larger sequence of computations. Errors can cancel and average each other out, e.g. when using approximations with a zero-centered and unbiased error distributions in case of larger accumulations or dot-product operations that are common in ML applications [3]. Similarly, depending on the application and its data flow pattern, errors in earlier operations can be either amplified or dampened by downstream computations. Error-configurable functional units as described in Section 2 allow designers to select the amount of approximations applied to each operation. The challenge is, however, in determining the approximation configuration of each operation in a larger data and control flow graph (CDFG) in order to achieve the best overall application-level accuracy-energy tradeoff while accounting for such interactions. With large design spaces and non-obvious tradeoffs, systematic approaches and effective tools are required to automatically derive such quality-energy optimal implementations.

Selection of optimal precision and approximations for each operator in a larger accelerator design can be naturally folded into existing high-level synthesis (HLS) tools. Such tools are concerned with automatically generating a register-transfer level (RTL) design from a given high-level description written in, e.g. C or SystemC. Scheduling and binding tasks in HLS when and where, i.e. in which clock cycle and on which resource to execute each operation in a CDFG. Within this context, *approximate* HLS (AHLS) tools additionally select approximate units to determine what type and the amount of approximations to apply in each operation while accounting for strong interactions with traditional scheduling and binding tasks [10, 26]. In general, approximations and associated logic simplifications affect not only switching activity and energy but also critical path delay of operators. This can be considered during scheduling and binding, e.g. to pack more operations into one clock cycle and reduce total latency or to balance slack and thus maximize clock frequency. Associated performance gains can in turn be translated into additional energy savings through voltage scaling under iso-performance assumptions. Compared to switching activity reductions alone, voltage scaling is particularly attractive and impactful due to the quadratic relationship between voltage and power or power density. Such opportunities can only be considered to a very limited extend by approaches that apply approximations to an already pre-designed, fixed RTL.

3.1 AHLS Overview

Figure 2 shows an overview of a typical AHLS flow [10, 26]. The input to the flow is a C description of the accurate design behavior including testbench and optimization parameters, as well as an accuracy constraint on the primary design outputs. The output is an approximate RTL description that maximized energy savings while meeting accuracy constraints. Approximations are performed as additional quality-energy optimization passes that tightly integrate with a traditional HLS flow (on the left).

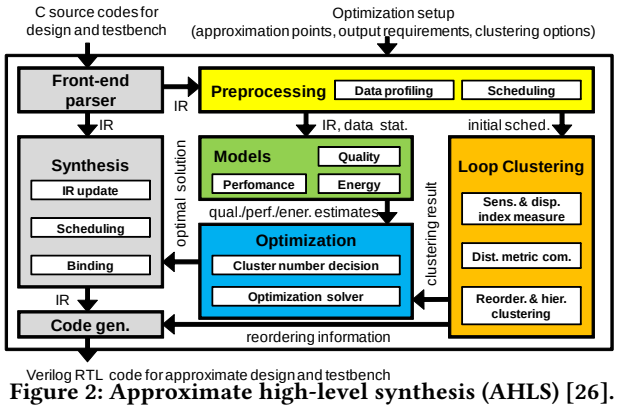


Figure 2: Approximate high-level synthesis (AHLS) [26].

Quality effects of approximations strongly depend on input data applied to operators. In a first pre-processing step, one-time profiling and scheduling of the design is performed to collect operation-level data statistic and mobility information. At the core of the optimization flow are then error/quality, energy and performance models that can predict the impact of different approximations applied to each operation on the final design. These models are in turn used by an optimization solver to perform a heuristics-based search in the design space and find Pareto-optimal solutions. The approximation-optimized solution together with updated delay information is then fed to the existing HLS flow to perform traditional scheduling and binding optimizations in the area-performance space.

Supported approximation techniques can be general in terms of approximated integer or floating-point arithmetic operators, including traditional fixed-point truncation or rounding. To this end, AHLS tools utilize an approximate component library with pre-characterized error, delay and energy models for different units. In addition, automatic pruning of operations as is widely employed in neural network literature can be folded into the AHLS flow as an extreme form of rounding that removes all data bits and completely eliminates operations [10]. In contrast to existing approaches that only consider the accuracy impact or that employ overly simplified cost models, incorporating pruning optimizations into AHLS tools again allows for consideration of their complex interactions with scheduling and binding during synthesis.

Finally, as a special form of signal processing, ML applications are often dominated by loop structures in the source code. Loops have traditionally been a key optimization target in HLS, but without consideration of approximations or quality tradeoffs. Data statistics and thus the impact of approximations can vary across loop iterations. Synthesizing loop bodies to apply the same approximations to all iterations ignores such variations. By contrast, loops can be unrolled such that each iteration can be approximated individually, but this can result in large resource overhead. Instead, dedicated approximation-aware loop optimizations have been proposed applied within an AHLS context [27]. In such approaches, loops are clustered and reordered according to their data statistics and hence approximation impact such that different approximations can be applied for each cluster.

3.2 Error and Quality Models

A crucial component to automated approximation optimizations are fast and accurate quality models that can predict the overall

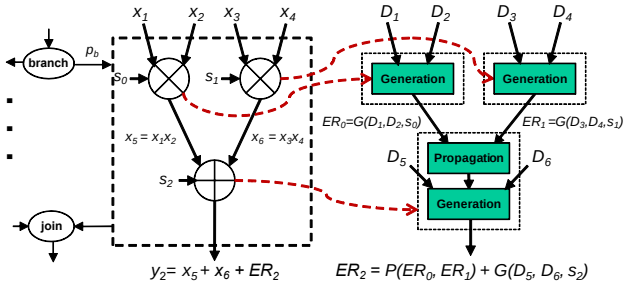


Figure 3: Quality/error modeling [26].

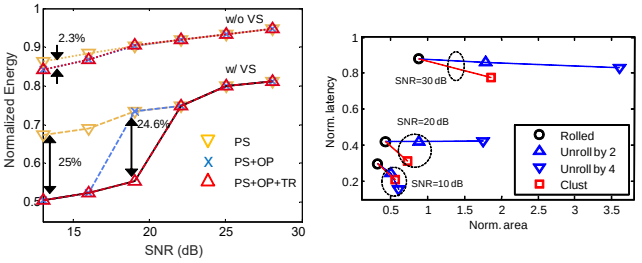


Figure 4: AHLS results for matrix multiplication example: quality-energy tradeoff (left) and loop optimizations (right).

accuracy impact of errors in individual operations of a larger application. Traditionally, simulation-based or analytical error modeling approaches have been used dating all the way back to traditional fixed-point analysis and optimization [28]. However, simulation approaches are flexible but often too time-consuming for design space exploration. By contrast, analytical models are fast but have limitations in supported approximation types. To address this challenge, semi-analytical and statistical quality models have been proposed instead [29]. In general, as Figure 3 shows, errors (ER) can be modeled as independent and additive to data (D), where the error at the output of an operation is a superposition of the error generated by the operation itself (G) and errors at the operand inputs propagated by through the operation (P). Data statistics D can be obtained from one-time profiling of the accurate code. By contrast, errors G_i generated by an operation i as a function of approximation level s_i depend on the specific approximation technique, where corresponding models for different approximate adders and multipliers have been proposed. Finally, depending on the desired quality metric, error propagation P requires traversing the CDFG in a breadth-first manner while accounting for branching and other control flow in a probabilistic manner (p_b). For example, to estimate SNR, a mean and variance propagation method can be employed [10].

3.3 Synthesis

Quality models are further combined with performance and energy models that account for both scheduling impact and voltage scaling. Given such models, quality-energy optimization becomes a generally non-linear and non-convex problem of minimizing energy or power density under quality constraints given integer decision variables for the optimization type and level of each operation. This optimization problem can be solved using standard meta-heuristics or specialized heuristic solvers that greedily evaluate solutions by traversing the graph in a breadth-first, branch-and-bound manner [10]. In the process, the aforementioned loop clustering optimizations can be applied as a pre-processing step [27]. Standard HLS scheduling and binding passes are then executed for the best set of solutions obtained from quality-energy optimization, where a post-synthesis slack balancing optimization selects the final solution for which synthesized RTL code is generated.

Figure 4 shows results of synthesizing a general matrix-matrix multiplication (GEMM) design with image input data as the core operation in image recognition neural network applications using fixed-point rounding as approximation technique. The figure on the left shows the quality-energy tradeoff enabled by AHLS for different SNR constraints, where energy values are normalized against the accurate baseline. Results are shown with and without voltage scaling (VS) using precision-scaling alone (PS) or in combination with operation pruning (OE) and/or scheduling optimizations (TR). Results show that without VS and corresponding opportunities to exploit processing time reductions, savings are limited. By contrast, VS can unlock significant additional gains, where pruning and scheduling optimizations can contribute up to 25% additional savings at more aggressive quality goals. Overall, up to 50% energy savings are achievable while maintaining high accuracy through joint precision scaling, pruning and scheduling optimizations. Finally, the graph on the right shows additional benefits of optimizing the main GEMM loop. Loop clustering primarily targets performance, but latency gains can be translated into energy savings through voltage scaling. Results show that compared to the original rolled design, up to 12% additional latency improvements can be obtained at the expense of larger area, where latency gains are the same or better than unrolling the loop by a factor of 2 or 4, which comes with an even larger area overhead.

4 RECONFIGURABLE APPROXIMATION FOR NEURAL NETWORK INFERENCE

Although fixed approximation circuits, as presented in prior sections, may deliver significant area, power, and/or delay gains, the requirement for numerical accuracy is not constant at runtime and the output quality is highly input dependent [30]. Therefore, fixed approximation circuits are designed under worst case scenarios, limiting the potential benefits [31]. The need for runtime adaptability becomes even more evident in the case of Neural Network (NN) accelerators, where the delivered accuracy highly depends on the NN size and architecture. Given an approximate multiplier, as the NN becomes deeper, the accuracy loss increases significantly [8]. Similar results are obtained for NNs with similar depth but different architectures. This issue can be addressed by applying approximation aware re-training [7]. However, for DNNs, the latter becomes a very time consuming procedure, if not even infeasible.

4.1 Layer-Wise Approximation

A heuristic framework is proposed in [31] that automates the generation of approximate reconfigurable combinational circuits. Wire-by-switch replacement is used as approximation technique and [31] employs high-level error, power, and delay estimation to avoid any hardware related evaluations and speedup the optimization procedure. During accurate operation, each switch outputs the wire's value, while during approximate operation, the switch output is

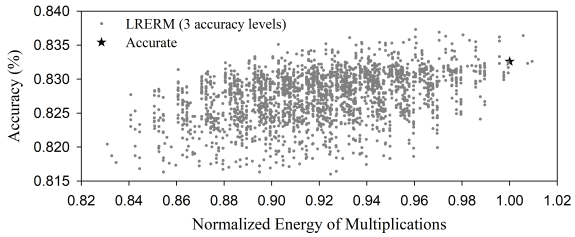


Figure 5: The accuracy-energy tradeoff of ResNet8 when using layer-wise approximation and the approximate reconfigurable LRERM multiplier [31].

a constant '0' or '1' limiting the circuit's switching activity. The heuristic optimization identifies the wires that will be replaced by switches so that a given quality function/constraint is satisfied and the dynamic power is minimized. The circuits generated by [31] support several accuracy levels, feature significant power reduction, and a slightly increased area. The accuracy level is dynamically selected at runtime by just setting a control signal. An 8-bit low relative error reconfigurable multiplier, namely LRERM, was generated in [31]. LRERM supports three accuracy levels, i.e., accurate and 0.5%, 1.5% mean relative error. Using LRERM in NN inference enables layer-wise approximation, i.e., dynamically set the desired accuracy level per convolution layer at runtime. The most sensitive layers can select more accurate execution while the less sensitive ones can switch to higher approximation and increase the energy/power gains. Figure 5 presents the energy-accuracy Pareto space for ResNet8 when using LRERM in the inference phase. Each point in Figure 5 corresponds to a different runtime configuration, i.e., LRERM accuracy level per layer. As shown in Figure 5, layer-wise approximation with LRERM delivers 15% energy reduction for only 0.5% accuracy loss. *Therefore, using dynamically reconfigurable approximate multipliers in NN inference, efficiently eliminates the need for retraining and enables achieving significant energy/power savings, while still satisfying tight accuracy loss thresholds.*

4.2 Weight-Oriented Approximation

However, even for the very small ResNet8, the size of the design space is significant (i.e., more that 2000 configurations in Figure 5) and for deeper NNs it will increase exponentially. As a result, although adopting approximate reconfigurable multipliers in NN inference offers high flexibility in controlling the power-accuracy tradeoff, a systematic approach is mandatory to enable efficient mapping of NNs to the supported accuracy levels of the approximate hardware and thus maximize the gains while still satisfying tight accuracy thresholds. Unlike the coarse-grain layer-wise approximation approaches, [8] proposed a fine-grain weight-oriented NN approximation, using approximate reconfigurable multipliers. First, [8] proposed an error correction to mitigate the error induced by the approximate multiplications in the convolution operation. The proposed error correction is implemented by just modifying accordingly the filters' biases. Given the convolution operation:

$$F = B + \sum_{j=1}^{n} W_j \times A_j, \tag{1}$$

the bias of each filter is modified as follows:

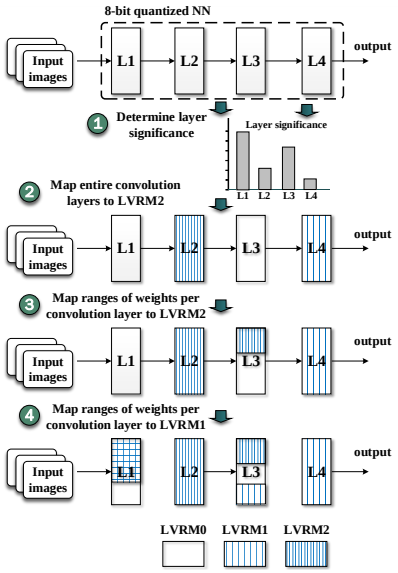


Figure 6: Weight-oriented approximate inference using low-variance approximate reconfigurable multipliers [8].

$$B' = B + \sum_{j=1}^{n} \mu(\epsilon_{W_j}), \tag{2}$$

where $\mu(\epsilon_{W_j})$ is the mean error of the approximate multiplication $W_j \times A$, $\forall A$. As a result, the mean error and variance of the approximate convolution are given by:

$$\begin{aligned} \mu(\epsilon_Y) &= 0 \\ \text{and } Var(\epsilon_Y) &= \sum_{j=1}^{k} Var(\epsilon_{W_j}), \end{aligned} \tag{3}$$

where $Var(\epsilon_{W_j})$ is the variance of the approximate multiplication $W_j \times A$, $\forall A$. Hence, the proposed bias correction effectively nullifies the mean convolution error and the accuracy of the convolution operation is defined only by the error variance of the approximate multiplier. Based on (1)-(3), low-variance is a more important error metric when designing approximate multipliers for NN inference. To this end, a low-variance 8-bit approximate reconfigurable multiplier (namely LVRM) with three accuracy levels (i.e., LVRM0, LVRM1, LVRM2) was generated using [31]. To map the weights of each layer to the accuracy levels of LVRM, a greedy approach was used (Figure 6). For each layer, three ranges are extracted (i.e., one for each accuracy level LVRM0, LVRM1, and LVRM2) and depending on the range that weight belongs to, the respective accuracy level is selected at runtime. In addition, the mapping algorithm leverages the significance of each layer and assigns more aggressive approximation to the less significant layers. This range approach is followed since it requires a very simple control circuitry with a minimal area overhead. Assuming a 64×64 systolic MAC array, only 256 8-bit comparators are required to implement the proposed mapping. The total area overhead is only 3% [8]. Table 3 presents the energy reduction with respect to the multiplication operations and the corresponding accuracy loss, when applying weight oriented approximation. Seven NNs trained on Cifar100 are considered. As

Table 3: Accuracy Loss and Energy Reduction of Multiplications for Varying NNs Trained on Cifar100 [8]

Neural Network	Accuracy loss (%)	Energy Reduction (%)
ResNet32	0.5	17.44
ResNet56	0.2	16.26
VGG11	0.5	18.62
VGG13	0.5	18.56
MobilNetv2	0.4	18.72

shown, the delivered energy reduction is 17.9% on average, while the accuracy loss is up to 0.5%.

Runtime reconfigurable approximation approaches deliver high power reduction without decreasing the area. Hence, a significant reduction in power density is achieved that translates to a temperature decrease. For example, considering a 64×64 systolic MAC array, the approximate architecture of [8] features the same operating frequency, 1.03x higher area, 11% lower power, and thus 14% lower power density. *Therefore, reconfigurable approximation can be employed to efficiently address thermal constraints.*

4.3 Temperature-Aware Approximation

Leveraging the impact of runtime reconfigurable approximation on the temperature, [2] proposed PFS-TE, the first hybrid thermal management for Neural Processing Units (NPUs). PFS-TE uses three knobs to control the temperature at runtime: frequency scaling, advanced on-chip cooling (superlattice thin-film Thermoelectric (TE) [32]), and dynamic approximation through precision scaling. Frequency scaling trades-off throughput to decrease temperature. Decreasing the operating frequency results to lower dynamic power and thus lower power density. Nevertheless, since the throughput of the NPU is linearly correlated to frequency, frequency scaling decreases significantly the NPU performance. Active Cooling trades-off power with temperature. As the input current of superlattice TE increases, the on-chip temperature decreases significantly. However, increasing the TE input current, increases the chip's power consumption. Finally, Precision Scaling trades accuracy for temperature decrease. Dynamically decreasing the precision of weights and activations, results in reduced switching activity and thus lower power and power density. On the other hand, using lower precision leads to an accuracy loss. In [2], precision scaling is applied though post-trainning min/max quantization to compensate some of the accuracy loss. Dynamic approximation through precision scaling is a vital component of PFS-TE since its role is twofold. For the same cooling cost (current in TE), precision scaling can be used to reduce the power and thus the temperature. In addition, for the same total chip power consumption, precision scaling can used to increase both the frequency as well as the input current of TE and boost the performance of the NPU under the same temperature threshold. *As a result, for given total power and temperature constraints, approximation becomes inevitable in order to boost performance.*

Figure 7 presents the efficiency improvement achieved by PFS-TE compared with the baseline. To reduce the temperature, the baseline applies frequency scaling and maximum forced convection air-cooling. Efficiency is defined as performance per Joule (TOPS/Joule). As shown in Figure 7, when no precision scaling is applied, PFE-TE can deliver 1.6x higher efficiency for the same accuracy (i.e., 8bit in Figure 7). However, when precision scaling is applied, PFS-TE can deliver 2.4x higher efficiency than the baseline. Hence, as Figure 7 shows, dynamic approximation significantly

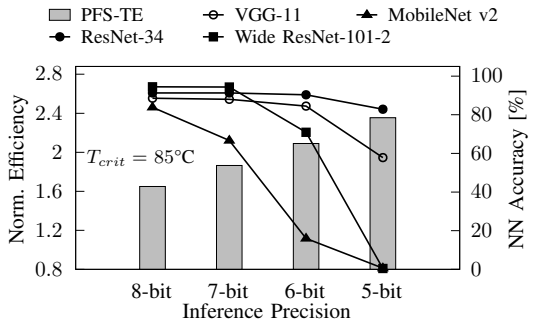


Figure 7: Efficiency (TOPS/Joule) improvement delivered by PFS-TE[2]. An 85°C temperature constraint is considered.

boosts the efficiency (from 1.6x to 2.4x, i.e., 1.5x). In addition, Figure 7 presents the accuracy variation w.r.t. precision scaling for four NNs trained on the ImageNet dataset. *The latter highlights the need for dynamic approximation, since different NNs feature significantly different accuracy loss under the same approximation.*

5 CHALLENGES, VISION, AND PERSPECTIVE

Approximate units such as multipliers form the basic building blocks for approximate hardware processors and accelerators. Approximate design of higher-level building blocks such as MAC and dot-product units, considered as a whole, further promises gains in resource-efficiency. Synergizing the state-of-the-art research in approximate multipliers and adders is important in this regard.

AHLS tools utilize libraries of such components to explore energy-accuracy tradeoffs at the whole RTL design level. Novel types of components can thereby open up whole new regions of the design space. Utilizing such components in AHLS and automated design space exploration tools will, however, require appropriate analytical or semi-analytical error (as well as delay and energy) models to be developed. Components that inherently provide predictable behavior based on well-defined mathematical foundations, e.g., log-based multipliers, may provide specific advantages in that regard.

At the same time, reconfigurability has to be supported by AHLS tools that allow for automatic synthesis of sequential, multi-stage circuits that can provide a set of configurable accuracy levels at their primary outputs while minimizing average energy including reconfiguration overhead. Given that a finer reconfiguration granularity at the individual stage level will come with larger overhead, the challenge is in finding the optimal combination of configurations in each stage to achieve the desired accuracy levels at primary outputs [33]. This further expands the design space and creates additional interactions with scheduling and binding tasks.

In addition to input variations, reconfigurability can also be used to compensate for other dynamic effects within the hardware itself. Specifically, several degradation mechanisms such as temperature and/or aging effects manifest themselves as delay increases in the critical paths of circuits that vary over time. Such delay increases result in timing violations and hence timing errors, due to unsustainable clock frequencies. To avoid that, circuits' designers traditionally need to include sufficient timing guardbands to keep degradation-induced timing errors at bay. However, timing guardbands directly lead to large efficiency losses. It has been recently

shown that employing principles from approximate computing enables designers to translate stochastic degradation-induced errors (which typically result in catastrophic failures and unacceptable loss in quality of circuits unless carefully controlled [34]) into deterministic and well-controlled approximations. This enables designers to trade-off efficiency with approximation while still overcoming degradation effects such as temperature [35] and/or aging [36, 37].

As discussed in Sections 2-4, there is a vast number of works that focus on improving the efficiency of neural network inference accelerators through approximate computing. However, research activities on approximate circuits for neural network training are still very limited. Using approximate accelerators in training might raise several issues/challenges that have to be addressed. For example, training might not converge or the trained model could be tightly coupled to this specific accelerator. Hence, running the obtained model on another platform (even an accurate one) might result in high accuracy loss.

In the ML application space, we are also seeing increasing trends towards customization and specialization of ML models and neural network architectures. This has led to the emergence of irregular ML architectures and automated machine learning (auto-ML) tools that explore corresponding design spaces to synthesize application-specific ML models for a given deployment. However, existing auto-ML approaches tend to either focus purely on accuracy optimization or use overly simplified complexity and cost models. In practice, auto-ML tools should be implementation-aware and closely coupled with ML-specific approaches and tools for hardware/software synthesis. This conversely necessitates the development of domain-specific, ML-aware AHLS and other approximate hardware design tools. Existing techniques for approximate computing have in general been largely applied in isolation at different levels of the abstraction stack, and there is a broader need for end-to-end cross-layer and co-design approaches that can jointly consider algorithm, system architecture and circuit optimizations [38].

REFERENCES

[1] N. P. Jouppi et al. 2017. In-datacenter performance analysis of a tensor processing unit. In *International Symposium on Computer Architecture*, 1–12.

[2] H. Amrouch, G. Zervakis, S. Salamin, H. Kattan, I. Anagnostopoulos, and J. Henkel. 2020. Npu thermal management. *IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems*.

[3] J. Miao, K. He, A. Gerstlauer, and M. Orshansky. 2012. Modeling and synthesis of quality-energy optimal approximate adders. In *IEEE/ACM International Conference on Computer-Aided Design (ICCAD)*.

[4] M. Shafique, W. Ahmad, R. Hafiz, and J. Henkel. 2015. A low latency generic accuracy configurable adder. In *Design Automation Conference*.

[5] G. Zervakis, K. Koliogeorgi, D. Anagnostos, N. Zompakis, and K. Siozios. 2019. Vader: voltage-driven netlist pruning for cross-layer approximate arithmetic circuits. *IEEE Trans. on Very Large Scale Integration Systems*, 27, 6, 1460–1464.

[6] H. Saadat, H. Bokhari, and S. Parameswaran. 2018. Minimally biased multipliers for approximate integer and floating-point multiplication. *IEEE Trans. Comput. Aided Des. Integr. Circuits Syst.*, 37, 11, 2623–2635.

[7] S. S. Sarwar, S. Venkataramani, A. Ankit, A. Raghunathan, and K. Roy. 2018. Energy-efficient neural computing with approximate multipliers. *ACM Journal on Emerging Technologies in Computing Systems (JETC)*, 14, 2, 1–23.

[8] Z. Tasoulas, G. Zervakis, I. Anagnostopoulos, H. Amrouch, and J. Henkel. 2020. Weight-oriented approximation for energy-efficient neural network inference accelerators. *IEEE Trans. Circuits Syst. I: Regular Papers*, 1–14.

[9] G. Zervakis, S. Xydis, D. Soudris, and K. Pekmestzi. 2019. Multi-level approximate accelerator synthesis under voltage island constraints. *IEEE Transactions on Circuits and Systems II: Express Briefs*, 66, 4, 607–611.

[10] S. Lee, L. K. John, and A. Gerstlauer. 2017. High-level synthesis of approximate hardware under joint precision and voltage scaling. In *Design, Automation and Test in Europe (DATE) Conference*.

[11] S. Jain, S. Venkataramani, V. Srinivasan, J. Choi, P. Chuang, and L. Chang. 2018. Compensated-dnn: energy efficient low-precision deep neural networks by compensating quantization errors. In *2018 55th ACM/ESDA/IEEE Design Automation Conference (DAC)*.

[12] E. Wang, J. J. Davis, R. Zhao, H.-C. Ng, X. Niu, W. Luk, P. Y. Cheung, and G. A. Constantinides. 2019. Deep neural network approximation for custom hardware: where we've been, where we're going. *arXiv preprint arXiv:1901.06955*.

[13] H. Jiang, C. Liu, L. Liu, F. Lombardi, and J. Han. 2017. A review, classification, and comparative evaluation of approximate arithmetic circuits. 13, 4.

[14] S. Venkataramani, S. T. Chakradhar, K. Roy, and A. Raghunathan. 2015. Approximate computing and the quest for computing efficiency. In *Design Automation Conference (DAC)*, 1–6.

[15] S. Rehman, W. El-Harouni, M. Shafique, A. Kumar, J. Henkel, and J. Henkel. 2016. Architectural-space exploration of approximate multipliers. In *International Conference on Computer-Aided Design (ICCAD)*.

[16] H. Saadat, H. Javaid, A. Ignjatovic, and S. Parameswaran. 2020. Realm: reduced-error approximate log-based integer multiplier. In *Proceedings of the 23rd Conference on Design, Automation and Test in Europe* (DATE '20). EDA Consortium, Grenoble, France, 1366–1371.

[17] K. Y. Kyaw, W. L. Goh, and K. S. Yeo. 2010. Low-power high-speed multiplier for error-tolerant application. In *Proc. EDSSC*, 1–4.

[18] K. Bhardwaj, P. S. Mane, and J. Henkel. 2014. Power- and area-efficient approximate wallace tree multiplier for error-resilient systems. In *Fifteenth International Symposium on Quality Electronic Design*, 263–269.

[19] P. Kulkarni, P. Gupta, and M. Ercegovac. 2011. Trading accuracy for power with an underdesigned multiplier architecture. In *Internatioal Conference on VLSI Design*, 346–351.

[20] S. Hashemi, R. I. Bahar, and S. Reda. 2015. DRUM: a dynamic range unbiased multiplier for approximate applications. In *International Conference on Computer-Aided Design*.

[21] S. Narayanamoorthy, H. A. Moghaddam, Z. Liu, T. Park, and N. S. Kim. 2015. Energy-efficient approximate multiplication for digital signal processing and classification applications. *IEEE Transactions on Very Large Scale Integration (VLSI) Systems*, 23, 6, 1180–1184.

[22] J. N. Mitchell. 1962. Computer multiplication and division using binary logarithms. *IRE Trans. on Electronic Computers*, EC-11, 4, 512–517.

[23] M. S. Ansari, B. F. Cockburn, and J. Han. 2019. A hardware-efficient logarithmic multiplier with improved accuracy. In *Design, Automation Test in Europe Conference Exhibition*.

[24] J. Y. F. Tong et al. 2000. Reducing power by optimizing the necessary precision/range of floating-point arithmetic. *IEEE Trans. on Very Large Scale Integration (VLSI) Systems*, 8, 3, 273–286.

[25] H. Zhang, W. Zhang, and J. Lach. 2014. A low-power accuracy-configurable floating point multiplier. In *International Conference on Computer Design)*.

[26] S. Lee and A. Gerstlauer. 2019. Approximate high-level synthesis of custom hardware. In *Approximate Circuits: Methodologies and CAD*. S. Reda and M. Shafique, editors. Springer.

[27] S. Lee and A. Gerstlauer. 2018. Data-dependent loop approximations for performance-quality driven high-level synthesis. *IEEE Embedded Systems Letters (ESL)*, 10, 1, 18–21.

[28] S. Lee and A. Gerstlauer. 2013. Fine grain word length optimization for dynamic precision scaling in DSP systems. In *IFIP/IEEE International Conference on Very Large Scale Integration (VLSI-SoC)*.

[29] S. Lee, D. Lee, K. Han, T. Kim, E. Shriver, L. K. John, and A. Gerstlauer. 2016. Statistical quality modeling of approximate hardware. In *IEEE International Symposium on Quality Electronic Design (ISQED)*.

[30] A. Raha and V. Raghunathan. 2017. Towards full-system energy-accuracy tradeoffs: a case study of an approximate smart camera system. In *Design Automation Conference*, 1–6.

[31] G. Zervakis, H. Amrouch, and J. Henkel. 2020. Design automation of approximate circuits with runtime reconfigurable accuracy. *IEEE Access*, 8, 53522–53538.

[32] G. Bulman, P. Barletta, J. Lewis, N. Baldasaro, M. Manno, A. Bar-Cohen, and B. Yang. 2016. Superlattice-based thin-film thermoelectric modules with high cooling fluxes. *Nature Communications*, 7, (2016), 10302.

[33] T. Alan, A. Gerstlauer, and J. Henkel. 2020. Runtime accuracy-configurable approximate hardware synthesis using logic gating and relaxation. In *Design, Automation & Test in Europe Conference & Exhibition (DATE)*.

[34] K. He, A. Gerstlauer, and M. Orshansky. 2011. Controlled timing-error acceptance for low energy IDCT design. In *Design, Automation and Test in Europe (DATE) Conference*.

[35] B. Boroujerdian, H. Amrouch, J. Henkel, and A. Gerstlauer. 2018. Trading off temperature guardbands via adaptive approximations. In *IEEE International Conference on Computer Design (ICCD)*.

[36] H. Kim, J. Kim, H. Amrouch, J. Henkel, A. Gerstlauer, K. Choi, and H. Park. 2020. Aging compensation with dynamic computation approximation. *IEEE Transactions on Circuits and Systems I: Regular Papers*, 67, 4, 1319–1332.

[37] H. Amrouch, B. Khaleghi, A. Gerstlauer, and J. Henkel. 2017. Towards aging-induced approximations. In *ACM/IEEE Design Automation Conference (DAC)*.

[38] P. Stanley-Marbell et al. 2020. Exploiting errors for efficiency: a survey from circuits to applications. *ACM Computing Surveys (CSUR)*, 53, 3, 51:1–51:39.