A Double Approximate Neural Hardware Accelerator

Samira Nazari[1], Ali Azarpeyvand[1,2], and Tara Ghasempouri[2]

[1]University of Zanjan, Zanjan, Iran
[2]Tallinn University of Technology, Tallinn, Estonia
[1]{samira.nazari, azarpeyvand}@znu.ac.ir
[2]{ tara.ghasempouri}@taltech.ee

***Abstract*—Emerging as a prominent approach, approximate computing has found widespread application in diverse fields such as data mining and image processing, effectively curbing area and power consumption in processors. One compelling avenue for integrating approximate computing involves the integration of a neural accelerator alongside the primary core, offering a delicate balance between computational efficiency and precision. A Neural Processing Unit (NPU) emulates regions of approximate code using neural networks. In this article, we introduce ANPiler—an innovative framework that augments the existing neural processing unit with a reconfigurable layer of approximation, achieved through the integration of inexact arithmetic building blocks. ANPiler empowers the design of neural accelerators featuring imprecise adders or multipliers, resulting in reductions of about 5.55% in power consumption and approximately 13.09% in area compared to a precise neural processing unit coupled with a main core. These gains are achieved with almost the same accuracy as before, further underpinning energy savings of 3.0× compared to general-purpose processors, thereby reinforcing the value proposition of an exact NPU in practical applications.**

***Index Terms*—Approximate Computing, Hardware Accelerator, Neural Networks, Approximate Arithmetic Design**

I. Introduction and Related Work

Processor design increasingly focuses on balancing energy efficiency, performance, and area to reduce costs and time-to-market. Programmable and reconfigurable accelerators, such as FPGAs, offer a viable alternative to application-specific instruction set processors (ASIPs), enabling efficient solutions tailored to specific tasks. As big data drives demand for processing diverse data types, edge devices face unique challenges. While many rely on sophisticated algorithms like deep learning, simpler and energy-efficient methods are often sufficient for tasks that can tolerate approximate results [1], [2].

Approximate computing leverages this tolerance for inexactness, reducing power consumption and computation time by intentionally relaxing accuracy. This study proposes a neural accelerator that offloads approximable tasks. Annotated program functions, identified for their error resilience, are approximated using neural networks configured to minimize mean squared error. The accelerator integrates approximate computational blocks to optimize energy and performance.

Research in approximate computing spans various domains, from image processing [3] to health applications [4]. Effective application requires identifying error-resilient code segments with well-defined input-output behavior [5]. Previous works have explored approximation methods at multiple levels, such as reducing data precision, pruning models, and designing approximate arithmetic units [6]–[8]. Innovations include approximate full adders and multipliers, with some designs utilizing memristors for inexact computations [9]–[11].

Accelerators have widely adopted these methods, integrating approximate multipliers on FPGAs [12] and machine learning-based approximations into hardware architectures [13]. Neural Processing Units (NPUs) represent a notable advancement, replacing specific code sections with neural models for improved energy efficiency [14]. While earlier studies explored neural networks with approximate multipliers [15], this work extends these concepts by combining neural models with approximate computational blocks to further enhance efficiency.

Key contributions of this paper include:

- **Inexact Adder:** A novel adder design that reduces area and delays on critical paths.
- **Neural Network with Approximate Blocks:** A neural architecture combining exact and approximate components for higher efficiency.
- **ANPiler Framework:** A novel framework that maps annotated code sections to neural networks and inexact adders, optimizing them for specific benchmarks [14].

The framework is evaluated using JPEG, FFT, and inversek2j applications, employing four approximate adders with varying widths. Compared to prior studies, this paper demonstrates the effective integration of neural accelerators and approximate computational blocks, showcasing improved energy efficiency and performance while maintaining acceptable accuracy levels.

II. Proposed Method

In this article, a compilation workflow for the approximate neural processor (ANPiler) is proposed. First, we discuss why such a workflow is beneficial and when it can be used for the design process. In the traditional digital design, the main core produces outputs via the inputs and some mathematical operations. With the help of accelerators and mainly neural accelerators (discussed in this article), main cores produce outputs more efficiently with fewer math operations. Here, an additional accelerator is coupled with the main core and there is a need for a programmer to identify the parts of code that need acceleration and can tolerate this acceleration without harming harm for the final outputs.

As a result, the programmer or anyone who designs the whole algorithm can use ANPiler to (1) test the code to see if it is possible to accelerate it with approximation (2) identify which parts of code can be accelerated without losing efficiency and (3) find a proper trade-off between accuracy, power consumption and area. ANPiler is used in the design phase; after verification and testing, there will be no sign of ANPiler, and only the interconnection of the main core and NPU is observed. ANPiler verifies how efficient it is to accelerate the code via approximation in a way that the user of the application does not feel that any approximation is applied to the algorithm.

The structure of ANPiler is depicted in Fig. 1. Before compilation starts, the code needs to be annotated which means that the regions of code that can be run via approximate methods should be selected. This phase is called programming, meaning any code changes and annotations occur at this point. This can be done either by a programmer or an automated tool which is out of scope of this article. Thus, the annotated code is the input of ANPiler.

After programming, code observation and training begin. There should be a training dataset including inputs and outputs that are preprocessed and made ready for the training phase. The inputs of neurons and the weights assigned to each of them produce an output using a specific activation function. The weights and other parameters are learned during the training process. Training takes place by using training data, and then the output is compared with the desired value and the mean squared error (MSE) is produced. In other words, we choose MSE as the loss function, and the weights are set through many learning steps such that the least MSE is obtained. The topology with the least MSE is chosen as the final configuration of the neural network which is a multi-layer perceptron with simple hardware implementation.

As mentioned before, the neural processing unit is composed of a specific neural network that behaves almost the same as the annotated code with a negligible amount of error. Each neural network includes many computing blocks such as adders and multipliers in different layers. Thus, it is crucial to design these blocks with more efficient schemes to save power, cost, and area to achieve higher performance.

For example, when we employ the neural approach discussed earlier in various applications like FFT, JPEG, or Inversek2j, we discover optimal neural network setups illustrated in Table I. The table presents two network configurations to show that each search exploration may yield slightly different results. Notably, the first selected configuration for the FFT benchmark requires approximately 88 multipliers, a rather substantial quantity. Consequently, there is a significant advantage in devising more efficient designs for these multipliers.

TABLE I: Selected neural network configurations

Benchmark	Network Config.	Error	Network Config.	Error
FFT	2-8-8-1	5.4e-6	2-4-8-1	2.2e-6
JPEG	64-8-64	2.1e-6	64-8-64	1.7e-6
Invesek2j	2-8-8-2	1.8e-3	2-8-8-2	1.6e-3

In this study, the application of approximate blocks to a neural processing unit is proposed and the design and implementation of the networks are performed via inexact computing blocks. In other words, to replace the annotated code with a neural network, first, the training data is collected and aggregated. At this point, the compile parameters encompassing the learning rate, epoch number, sampling rate, test data fraction, maximum number of layers, and maximum number of neurons per layer are provided by the user. Subsequently, diverse possible neural network topologies are explored, and accordingly, the best topology with the least mean squared error is selected.

Afterward, before compiling, the annotated code is replaced by the best neural network topology [14] and inner exact computing blocks. In ANPiler, a neural network is designed using different types of approximate blocks. Consequently, the output of this step is different errors for the different schemes compiled and executed by ANPiler. In this step, any model of computing blocks can be applied to the code, regardless of the results they might have. Put differently, double approximation is done through this step when any inexact computing block can be coupled with the neural accelerator.

Therefore, any computing block with any level of complexity can be placed in the new library piece, which is automatically compiled and executed. As an example, in this research, five types of blocks were applied to evaluate how they could affect the design. The structures of these examples are shown in Fig. 2. Fig. 2a is representative of the exact and common adder used in many other applications.

The three other types of adders are shown in Fig. 2b, Fig. 2c, and Fig. 2d. The difference between these adders is in the calculation of the Least Significant Bits(LSBs). They can either ignore the computing process for the LSBs completely (Fig. 2b, Fig. 2c) or use much simpler circuits (Fig. 2d). To be precise, Approximate Adder 1 (AFA1) replaces the full adder for LSBs with only one "OR" gate, and Approximate Adder 2 (AFA2) replaces it with one "XOR" gate. However, Approximate Full Adder 3 (AFA3) replaces the full adder with

TABLE II: Truth table of different adders

Input			**FA**		**AFA1**		**AFA2**		**AFA3**		**AFA4**	
A	**B**	**C**	**S**	**C**	**S**	**C**	**S**	**C**	**S**	**C**	**S**	**C**
0	0	0	0	0	0	-	0	-	0	0	0	0
0	0	1	1	0	0	-	0	-	1	0	1	0
0	1	0	1	0	1	-	1	-	1	0	1	0
0	1	1	0	1	1	-	1	-	1	1	0	0
1	0	0	1	0	1	-	1	-	1	0	1	0
1	0	1	0	1	1	-	1	-	1	1	0	0
1	1	0	0	1	1	-	0	-	0	1	0	1
1	1	1	1	1	1	-	0	-	1	1	1	1

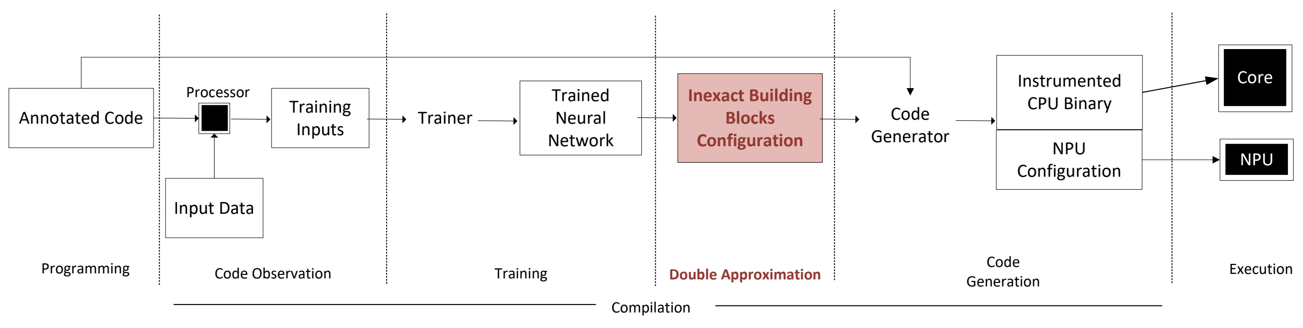
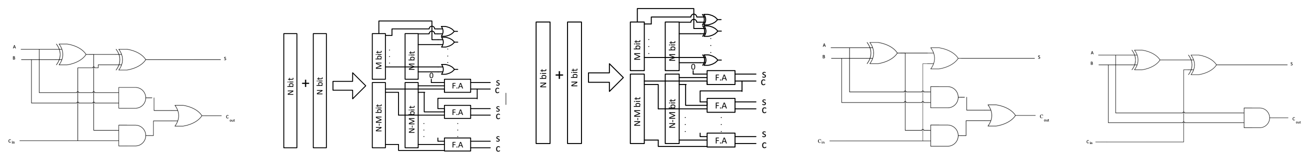


Fig. 1: Structure of ANPILER



(a) FA: Full Adder (b) AFA1 (c) AFA2 (d) AFA3 (e) AFA4

Fig. 2: Structures of different adders

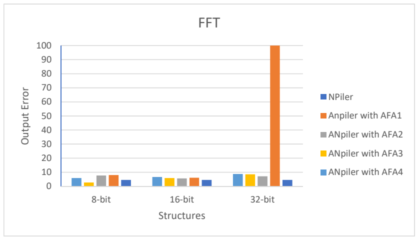
Algorithm 1: ANPiler Algorithm

```
if any region of code is approximable then
    Annotate approximate region;
    Prepare dataset;
    Send dataset to NPU;
    Train;
    for different configurations do
        Calculate MSE;
        Choose the configuration with the least MSE;
    end
    for different adders do
        Replace adders of selected neural network configuration
        with the approximate adder;
        Calculate total error for each configuration;
        Choose the Least;
    end
    Generate code for NPU and main core;
    Run the whole code;
    Infer the approximate region;
    Send the results to main core;
    Report the output;
else
    Run without NPU;
end
```

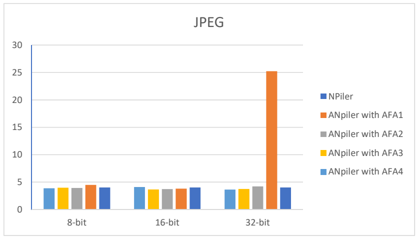
a similar full adder that changes one "XOR" to "OR" gate.
In addition to this, a new scheme for approximate adder is proposed (Fig. 2e) that similar to previous adders, calculates the LSBs with simpler circuits and thus shorter critical paths. Approximate Full Adder 4 (AFA4) has two gates fewer than the exact full adder. The truth table of all different structures is listed in Table II. FA refers to the exact and AFAs refer to the approximate outputs. For instance, our proposed adder (AFA4) raises errors in two states. Obviously, the number of gates is fewer than the exact method as the carry bit is computed using one “AND” gate instead of two “AND” and one “OR” gate. All five adders are evaluated using different benchmarks, and the results can be found in the last part of the article. Therefore, the NPU is implemented by ANPiler with five designs, the results of which are evaluated.
After this phase, the code is generated and it can be sent to both the main core and NPU (Algorithm 1). At the approximation code, the main core waits for NPU to send the results and then it continues with the NPU results which means the main core skips the execution of approximate code and only uses the results.
In this article, we not only introduce our approximate full adder but also claim that for every application, different structures of inexact building blocks along with NPU can be explored to see which one performs more efficiently considering output error, power consumption, and area. Therefore, after this exploration and evaluation the mentioned trade-offs, the best neural network topology, and the best inexact building block are selected. The selections totally depend on the algorithm and behavior of the desired application.

III. Experimental Results

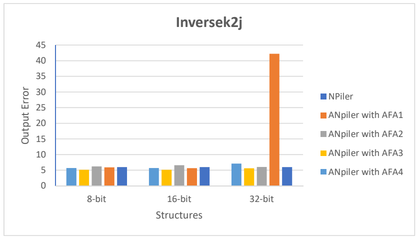
To evaluate the results, approximate blocks were applied to 8, 16, and 32 bits of partial products needed by neural networks, respectively, to three benchmarks: FFT, JPEG, and inversek2j. These benchmarks were chosen from AxBench, a benchmark suite for approximate computing [16]. This is implemented through the NPiler framework [14] which is modified to implement inexact building blocks. Three factors were evaluated through a process that includes the output error, power, and area. Figure 3 depicts the impact of different approximate structures on the benchmarks regarding the output error. The first approximate structure is approximated



(a) FFT

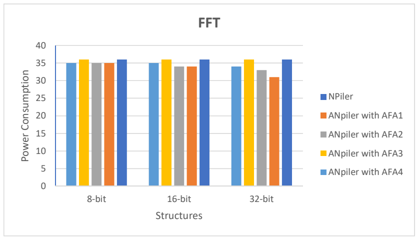


(b) JPEG

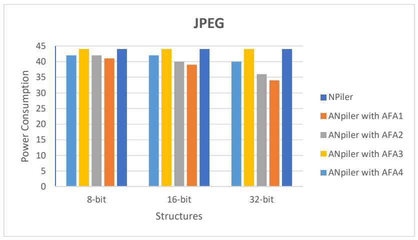


(c) Inverek2j

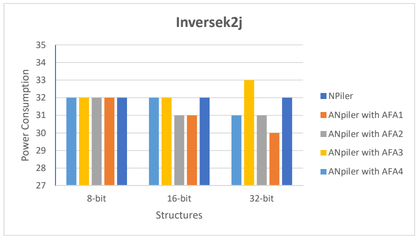
Fig. 3: Impact of Approximation on Output Error



(a) FFT

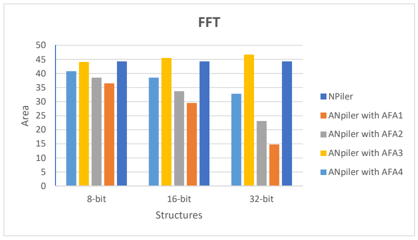


(b) JPEG

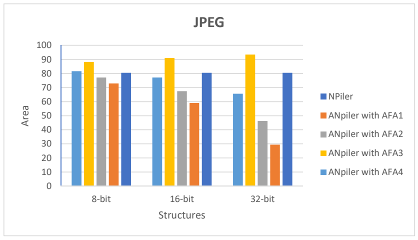


(c) Inverek2j

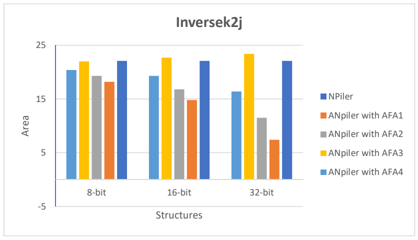
Fig. 4: Impact of Approximation on Power Consumption (milli watts)



(a) FFT



(b) JPEG



(c) Inverek2j

Fig. 5: Impact of Approximation on Area (fraction of used gates %)

by a neural network accompanying exact computing blocks, whereas the other four structures are approximated by neural networks and inexact computing blocks.

Each structure was approximated in three formats, depending on the number of inexact bits. As shown in Fig. 3, the output error of all structures except the second one is almost the same as that of the structure with a neural network and exact computing block. The fifth structure is the neural network with the approximate adder proposed in this study, the error of which is negligible. Thus, it can be deduced that our proposed structure (AFA4) performs better in the case of power and area but with around 2% increase in total error.

The impact of the aforementioned approximate structures on the power consumption and area is shown in Fig. 4 and Fig. 5. In both cases, it is clear that the power consumption and area decreased in the approximate structures. Although some unexpected increases can be observed in a few structures, the overall result is a decrease in the approximate methods.

It is clear that with more approximate bits, the decrease in power consumption and area is more obvious. Thus, 32-bit approximation seems more efficient but with proper full adder. AFA1 is not a good choice and shows that it is important to choose a good structure for approximate building blocks. The same thing happens for the area and AFA4 shows a stable behavior in all benchmarks with negligible error. To visualize, the outputs of the JPEG benchmarks with different structures are shown in Fig. 6. Structure 2 refers to a neural network with AFA1 that almost destroys the output picture. However, structures 3, 4, and 5 which are outputs of neural networks with AFA2, AFA3, and AFA4 respectively, are almost the same as the exact method and Structure 1 (neural network with exact full adder).

As we know, each time our tool searches for the best configuration, it chooses it according to the least error. Interestingly, it is clear that the neural network configurations are small. For instance, the neural network configuration selected for the FFT benchmark is often 2-8-8-1 which contains only two hidden layers with a maximum of 16 neurons and is considered nothing among deep neural networks.

In JPEG, the neural network architecture is slightly different (64-8-64) due to the input and output dimensions which results in more area in comparison with FFT. We can observe a similar

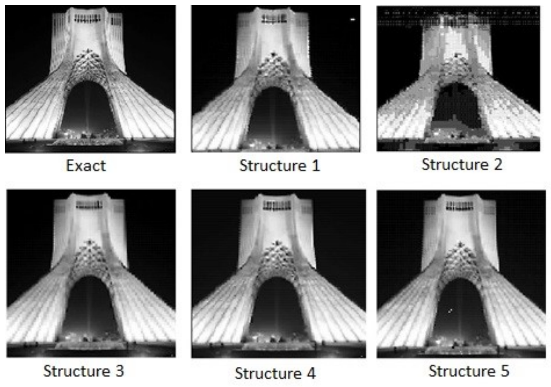


Fig. 6: Impact of Approximation on Output Error (JPEG)

behavior for 8 and 16 bits of approximation. However, the interesting part is in 32 bits of approximation where not only can we see a decrease in power and area, but also a decrease of 0.39% in error is detected.

In 8 and 16 bits of approximation of Inversek2j, the selected neural network configuration is 2-8-8-2 and our proposed structure (AFA4) performs better in all aspects including power, area, and error; however, in 32 bits of approximation, all except error have better values which are still accepted. Consequently, it is observed that each benchmark can show different behavior in approximation techniques that is due to structural differences in the algorithms.

IV. Conclusion

There are many applications that do not require exact results; meanwhile, power consumption, area, and cost of production are considered crucial factors in accomplishing effective production of such applications. As a result, an approximation tool (ANPiler), a neural network with inexact computing blocks, and a new approximate adder are proposed in this study to obtain effective inexact results with negligible error and an acceptable decrease in power consumption and area. For instance, in a typical FFT application accompanied by the method proposed in this article, a decrease of 5.55% in power consumption and 13.09% in area is observed, which can play important roles in improving the production quality of this application.

References

[1] Mahdi Taheri, Natalia Cherezova, Samira Nazari, Ali Azarpeyvand, Tara Ghasempouri, Masoud Daneshtalab, and Jaan Raik. Adam: Adaptive approximate multiplier for fault tolerance in dnn accelerators. *IEEE Transactions on Device and Materials Reliability*, 2024.

[2] Mahdi Taheri, Natalia Cherezova, Samira Nazari, Ahsan Rafiq, Ali Azarpeyvand, Tara Ghasempouri, Masoud Daneshtalab, Jaan Raik, and Maksim Jenihhin. Adam: Adaptive fault-tolerant approximate multiplier for edge dnn accelerators. *The 29th European Test Symposium (ETS)*, 2024.

[3] Samira Nazari, Mostafa Charmi, Maryam Hassani, and Ghazale Ahmadi. A simplified method in human to robot motion mapping schemes. In *2015 3rd RSI International Conference on Robotics and Mechatronics (ICROM)*, pages 545–550. IEEE, 2015.

[4] Ardalan Najafi, Amir Najafi, Julia Müller, Wanli Yu, and Alberto Garcia-Ortiz. Approximate computing in critical applications: Ecg arrhythmia classification. In *2023 12th International Conference on Modern Circuits and Systems Technologies (MOCAST)*, pages 1–4. IEEE.

[5] Amir Yazdanbakhsh, Divya Mahajan, Bradley Thwaites, Jongse Park, Anandhavel Nagendrakumar, Sindhuja Sethuraman, Kartik Ramkrishnan, Nishanthi Ravindran, Rudra Jariwala, and Abbas Rahimi. Axilog: Language support for approximate hardware design. In *2015 Design, Automation Test in Europe Conference Exhibition (DATE)*, pages 812–817. IEEE.

[6] Jasleen Kaur Brar. *Multi-Layer Approximate Computing*. Thesis, 2022.

[7] Samane Asgari, Mohammad Reza Reshadinezhad, and Seyed Erfan Fatemieh. Energy-efficient and fast imply-based approximate full adder applying nand gates for image processing. *Computers and Electrical Engineering*, 113:109053, 2024.

[8] Bahareh Vakili, Omid Akbari, and Behzad Ebrahimi. Efficient approximate multipliers utilizing compact and low-power compressors for error-resilient applications. *AEU - International Journal of Electronics and Communications*, 174:155039, 2024.

[9] Mohsen Nourazar, Vahid Rashtchi, Ali Azarpeyvand, and Farshad Merrikh-Bayat. Code acceleration using memristor-based approximate matrix multiplier: Application to convolutional neural networks. *IEEE Transactions on Very Large Scale Integration (VLSI) Systems*, 26(12):2684–2695, 2018.

[10] Muhammad Abdullah Hanif, Vojtech Mrazek, and Muhammad Shafique. *Approximate Computing Architectures*, pages 1–41. Springer, 2022.

[11] Mohsen Nourazar, Vahid Rashtchi, Farshad Merrikh-Bayat, and Ali Azarpeyvand. Towards memristor-based approximate accelerator: application to complex-valued fir filter bank. *Analog Integrated Circuits and Signal Processing*, 96(3):577–588, 2018.

[12] R Loo. *Investigating Approximate FPGA multiplication for increased power-efficiency*. Thesis, 2022.

[13] Mahmoud Masadeh, Osman Hasan, and Sofiene Tahar. Machine learning-based self-compensating approximate computing for autonomous systems.

[14] Hadi Esmaeilzadeh, Adrian Sampson, Luis Ceze, and Doug Burger. Neural acceleration for general-purpose approximate programs. *Communications of the ACM*, 58(1):105–115, 2014.

[15] Vojtech Mrazek, Syed Shakib Sarwar, Lukas Sekanina, Zdenek Vasicek, and Kaushik Roy. Design of power-efficient approximate multipliers for approximate artificial neural networks, 2016.

[16] Amir Yazdanbakhsh, Divya Mahajan, Pejman Lotfi-Kamran, and Hadi Esmaeilzadeh. Axbench: A benchmark suite for approximate computing across the system stack. Report, Georgia Institute of Technology, 2016.