# FPGA-based Hardware Acceleration of Artificial Neural Network Inference

1st Stefan Rothhaupt
*Department of Electrical Engineering*
*Munich University of Applied Sciences*
Munich, Germany
stefan.rothhaupt@hm.edu

2nd Wolfgang Meyerle
*Airbus Defence and Space GmbH*
Manching, Germany
wolfgang.meyerle@airbus.com

3rd Benjamin Kormann
*Department of Electrical Engineering*
*Munich University of Applied Sciences*
Munich, Germany
benjamin.kormann@hm.edu

***Abstract*—Driven by the growing demand for efficient computation, this paper explores the acceleration capabilities of hardware accelerators in neural network inference, with a particular focus on their potential applications and significance within the aviation industry. This paper presents an exemplary neural network inference implementation on the Xilinx Versal VCK190 Evaluation Board, developed in collaboration with Airbus Defence and Space. The work employs a TensorFlow-trained multi-layer perceptron (MLP) to approximate the Mandelbrot set. The inference process harnesses Xilinx AI Engines for accelerated performance, resulting in remarkable results. The computational time required for a singular inference operation is approximately 15 microseconds, presenting a notable enhancement when contrasted with CPU-based inference, which demands approximately 330 microseconds. This translates to a speedup factor of approximately 21, affirming the computational efficiency of the proposed inference methodology. Notably, accuracy remains consistent with the TensorFlow-based approach, courtesy of the FPGA's proficiency in processing float32 data, eliminating the need for retraining or quantization to int8.**

**This research addresses the aviation industry's essential determinism requirements, making it suitable for safety-critical systems. Additionally, it potentially aligns with industry demands for low power consumption and future-proof solutions, reinforcing the relevance of FPGA-accelerated machine learning inference in aviation applications.**

## I. Introduction

The pursuit of hardware acceleration for neural network inference is a prominent topic in modern computational research. In the 'Related Work' chapter, various approaches and research endeavors are elucidated, offering a brief overview of the existing literature in the field. It revolves around the essential principle of parallelization, which is crucial for efficient computation. Graphics Processing Units (GPUs) and Field-Programmable Gate Arrays (FPGAs) are two avenues offering highly parallel architectures for neural network implementation. [1], [5]

The aviation industry places a premium on both computational speed and energy efficiency in its hardware solutions. Efficiency is paramount in an industry that demands reliability and strict determinism in safety-critical airborne systems, aligning with the DO-178B/C standards for stringent safety and certification requirements. Here, GPUs exhibit certain limitations compared to the capabilities of FPGAs. [1]–[3]

This paper focusses on the Xilinx Versal product line to explore neural network inference, particularly in edge devices. Edge devices, often constrained by limited power and intermittent internet connectivity, present unique challenges for hardware acceleration. The Versal product line, known for its versatility, serves as a promising solution for diverse requirements, from edge devices to data centers. [5]

This paper aims to elaborate, implement, and evaluate a exemplary application to assess the suitability of FPGAs, particularly the Xilinx Versal Product line, for Airbus Defence and Space and the aviation industry's neural network inference needs. The focus is on developing a high-performance system using the float32 data type, ensuring accuracy parity when compared to TensorFlow inference conducted in the same data format.

This research was funded by Airbus Defence and Space Manching, Germany. The investigation was conducted through a collaborative effort between the research team and the Department of Electrical Engineering at the University of Applied Sciences Munich.

## II. Related Work

In the domain of hardware acceleration for neural network inference, a substantial body of research and development has been devoted to harnessing the computational capabilities of GPUs and FPGAs. This section offers an overview of the relevant literature and key findings that have informed the current study.

### A. GPU-based acceleration

GPUs, developed by NVIDIA, have long been recognized for their parallel processing capabilities, making them pivotal in accelerating various computational tasks, including neural network inference. Previous research has showcased their effectiveness in significantly expediting training and inference processes. [6], [7]

However, GPUs, despite their prowess in parallel execution, face certain trade-offs. A notable concern is power consumption, particularly in applications where energy efficiency is paramount, exemplified by the stringent requirements of the aviation industry. Additionally, the potential overhead associated with data transfer to and from GPUs can diminish the benefits of acceleration, particularly in cases where low latency is indispensable. [1], [3]

The challenge of non-deterministic behaviour exhibited by CUDA warps, which can significantly impact the reliability and repeatability of parallel thread execution within a warp, a fundamental unit of execution on NVIDIA GPUs is of particular note. Non-deterministic behavior arises when threads within a warp take different code paths due to conditional branching, leading to serialization. This introduces unpredictability in the execution of parallel threads, which may be undesirable in safety-critical applications, such as aviation, where deterministic performance is imperative. [3], [4]

The non-determinism of CUDA warps necessitates careful consideration when utilizing NVIDIA GPUs for neural network inference, especially in contexts where predictability and repeatability are essential. These challenges were addressed by the exploration of FPGA-based acceleration as an alternative approach, given its potential for determinism and optimization, which aligns with the unique requisites of aviation industry applications.

### B. FPGA-based acceleration

FPGAs represent an alternative approach that has gained increasing attention in the realm of hardware acceleration for neural network inference. FPGAs offer the distinct advantage of customizable, hardware-level parallelization. The ability to tailor hardware to a specific inference task bestows an unmatched level of optimization. Furthermore, FPGAs are renowned for their potential to minimize power consumption, a critical attribute in safety-critical applications. [1], [8]

Recent studies have delved into FPGA-based acceleration, demonstrating impressive results in terms of both performance and power efficiency. These efforts have underscored the potential of FPGAs in addressing the specific requirements of edge devices and scenarios where determinism and low power consumption are paramount. A noteworthy feature is the capacity of Xilinx Versal products to handle floating-point precision (float32) data without necessitating retraining or quantization to int8, aligning with the objectives of precise neural network inference. [9]

The selection of the Xilinx Versal FPGA, a member of the Versal product line, for the study is underpinned by its reputation as a versatile hardware accelerator capable of meeting the diverse needs of edge devices. Versal FPGAs are explicitly promoted as a suitable fit for both edge devices and data centers, showcasing their adaptability in a range of scenarios. The reconfiguration capabilities inherent in FPGAs render them well-suited for applications necessitating serviceability, updates, or scalability, positioning them as highly resilient candidates for hardware acceleration tasks with considerable future-proof attributes. This paper extends previous research by developing a exemplary application on the Versal platform, with the aim of evaluating its suitability for Airbus Defence and Space and the aviation industry, especially concerning the unique requirements of deterministic, energy-efficient, and future-proof hardware. [5]

The subsequent sections delve into the methodology and results, shedding light on the practical implications of this research.

## III. Implementation of a multi-layer-perceptron on Xilinx AI Engines

### A. Introduction to multi-layer-perceptrons

A Multi-Layer Perceptron, or MLP, is a type of artificial neural network composed of multiple layers of interconnected neurons. It consists of an input layer, one or more hidden layers, and one output layer. Each neuron in a layer is connected to every neuron in the subsequent layer. [10]

In its basic form, an MLP is a feedforward neural network, meaning that data flows in one direction, from the input layer to the output layer. Each connection between neurons is associated with a weight, and each neuron typically employs an activation function to introduce non-linearity into the model. [10], [11] The weights can be represented in a weight-matrix. This weight-matrix $\mathbf{W}$ and the input-vector $\mathbf{x}$ in a Multilayer Perceptron can be multiplied as follows:

$$\mathbf{W} \cdot \mathbf{x} = \begin{bmatrix} w_{11} & \dots & w_{1n} \\ w_{21} & \dots & w_{2n} \\ \vdots & \ddots & \vdots \\ w_{m1} & \dots & w_{mn} \end{bmatrix} \begin{bmatrix} x_1 \\ x_2 \\ \vdots \\ x_n \end{bmatrix} = \begin{bmatrix} w_{11}x_1 + \dots + w_{1n}x_n \\ w_{21}x_1 + \dots + w_{2n}x_n \\ \vdots \\ w_{m1}x_1 + \dots + w_{mn}x_n \end{bmatrix}$$

This relationship can be expanded to represent each layer in the MLP as a calculation involving the matrix-vector-multiplication followed by elementwise application of the activation fucntion [12]. Figure 1 illustrates this expanded relationship.

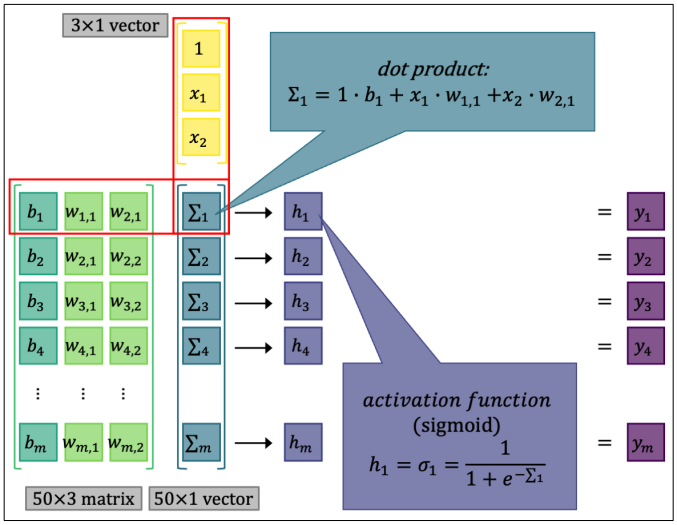


Fig. 1. Illustration depicting the matrix-vector multiplication of a Multilayer Perceptron (MLP) layer. The subsequent activation function applied elementwise to the resulting vector, makes the layer complete. [12]

The initial hidden layer manifests as a matrix-vector multiplication, coupled with subsequent element-wise activation functions, as illustrated in Figure 1. Subsequent layers undergo analogous computational processes. This matrix-vector-multiplication (or matmul) can be computed in a highly

parallelized way by the Xilinx AI Engines, that are introduced in the next subsection. [13]

### B. Introduction to Xilinx Versal VCK190 evaluation board

Xilinx AI Engines are a key component of Xilinx's Versal system-on-chip (SoC) family. These Engines offer highly customizable, power-efficient hardware acceleration for AI workloads, while being optimzed for inference tasks and not training of neural networks. These engines excel in parallel processing, supporting various data precision formats, and are tightly integrated into the Versal SoC. Xilinx provides development tools and frameworks to facilitate AI workload design and programming. [13]

The Graph Programming Model is the design model chosen for this task, multiple kernels are connected to each other making up a graph that represents the inference. Each kernel will be executed on one AI Engine, therefore the parallelization is the task of the developer. In this project the inference will be split up into kernels that compute the matmuls and the activation function. The kernels can compute multiple vector-elements at the same time. The amount of data processed in parallel varies depending on the datatype. The diagram in Figure 2 shows a comparision of the various datatypes. [14]

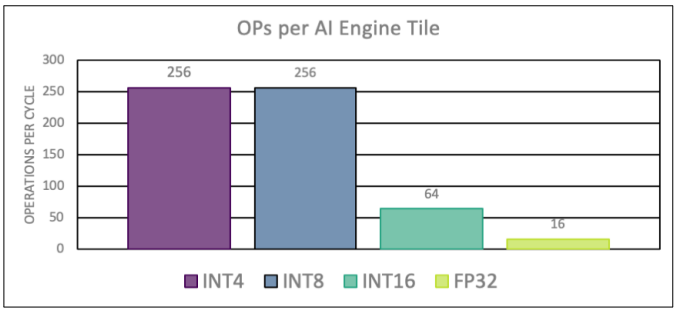


Fig. 2. Comparison of operational capacity per cycle relative to data types. Highlights the support of 16 operations for the float32 data type, affirming its preferred utilization in this paper.

The primary objective of this research was to achieve the seamless implementation of the float32 data type. The aim was to enable inference without the necessity for quantization or retraining procedures. The goal was to establish a direct pathway for the effortless migration of a TensorFlow model to the Xilinx System-on-Chip (SoC) platform.

### C. Inference build

The inference model employed in this approach is a MLP designed with distinct architectural elements. The typology used for this inference is provided by Airbus Defence and Space to facilitate a comparative analysis of the results against internal research findings. It consists of an input layer that receives two-dimensional coordinates, denoted as 'x' and 'y.' Subsequently, the model features a hidden layer comprising 50 neurons, followed by a second hidden layer with 43 neurons. The output layer, which produces the final inference, comprises a single neuron. Each layer of the MLP is characterized by the utilization of the sigmoid activation function. Notably, a skip connection has been introduced, facilitating a direct linkage from the input layer to the second hidden layer. The machine learning framework TensorFlow was employed for model construction and training execution. The model was leveraged for a regression task, delineated by the binary classification of pixels, represented by coordinates, as either part or not part of a predefined set. The model was provided with coordinates representing individual pixels, and its training was centered on the task of determining, based on the learned features, whether a given pixel belongs to the Mandelbrot set. Figure 3 shows the original Mandelbrot set, plotted with a resolution of 100x100 pixels and the TensorFlow inference also plotted with 100x100 pixels.

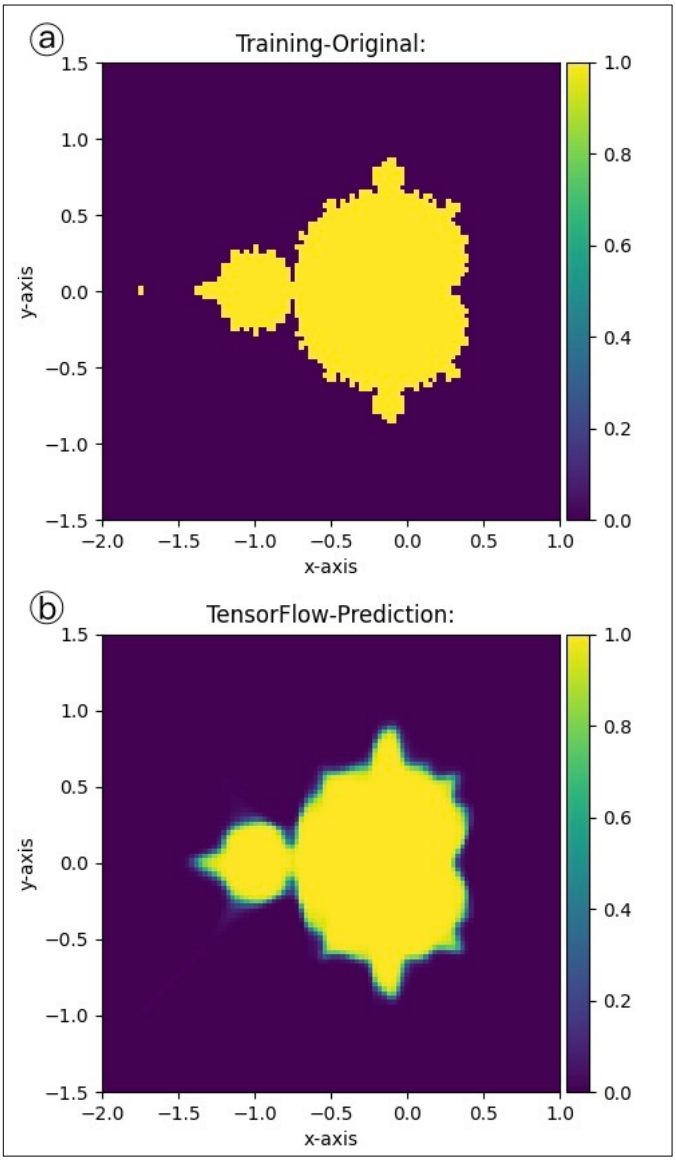


Fig. 3. Visualization of the Original Mandelbrot set ⓐ and the TensorFlow-inferred Mandelbrot set ⓑ. Demonstrates TensorFlow's capability in approximating the intricate details of the Original Mandelbrot set.

Subsequent to training, the model's weights were extracted.

This weight extraction process was pivotal in enabling the subsequent design and implementation of the inference model on the FPGA hardware, emphasizing the essential continuity between the training phase and the deployment phase.

### D. Kernel for the matrix-vector-multiplication

The primary computational method employed to achieve matrix-vector multiplication entailed the utilization of the *fpmac* (floating point multiply and accumulate) intrinsic function, sourced from the Xilinx intrinsic functions C++ library. Notably, the *matmul* (matrix multiplication) function within the Xilinx AI Engine API C++ library was not deemed suitable for the given purposes due to its restriction on supporting vectors as integral components of the operation, necessitating a minimum matrix size of 2x2. Consequently, the intrinsic *fpmac* function was adopted to implement the algorithm for matrix multiplication, designed to process vectors of eight float32 elements within one cycle. The algorithm iteratively traversed the weight matrix until the complete matrix had been computed, subsequently producing the resulting vector. [14]–[16]

In the case of the initial layer, which featured 50 neurons, zero-padding was introduced to accommodate the last six elements of the resultant vector, aligning with the vector size of 56. Subsequent layers were constructed accordingly, tailored to suit the matmul kernel.

### E. Kernel for the skip connection

The skip connection was implemented by shifting the elements in the output vector of the first hidden layer. Then the inputs to the neural network were added as elements to the vector. Intrinsic functions, that are able to compute a "vector-wise" shift, are not supported by Xilinx. Thus the shift has to be computed scalar-wise, introducing a potential bottleneck to the inference.

### F. Kernel for the activation function

The activation function employed in the neural network architecture is the sigmoid function, which serves as a vital element for non-linearity in neuron response modeling. It is defined as follows:

$$\sigma(x) = \frac{1}{1 + e^{-x}} \tag{1}$$

Where: $\sigma(x)$ is the sigmoid function, $x$ is the input to the function. To facilitate efficient computation, the representation of the sigmoid function is to be implemented through the utilization of look-up tables (LUTs), as dealing with exponentials on an FPGA can be computationally demanding. The sigmoid function is discretized within the range of -7.5 to 7.5, yielding 1024 discrete values sampled at regular intervals of 0.014. These discrete sigmoid values are subsequently stored in the internal 32kB memory of the AI Engine Tile. Each value is being stored in a float32 format. This allocation of memory space implies that approximately 1/8 of the available memory capacity on a single tile is utilized to store the precomputed values of the sigmoid function. This efficient use of memory resources enables rapid and precise computation of the sigmoid function during neural network inference. Figure 4 illustrates the sigmoid's characteristic 'S'-shaped curve, where its output gradually transitions from 0 to 1 as the input varies from -7.5 to 7.5, showcasing the function's suitability for modeling non-linear relationships in artificial neural networks. [8], [14], [16]

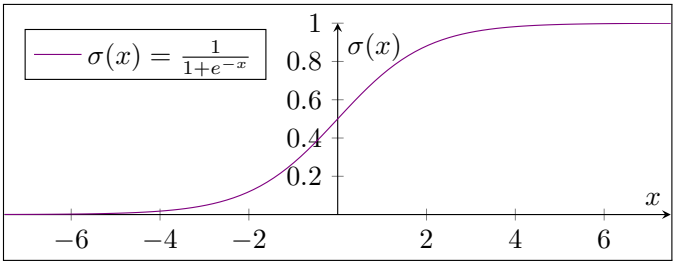


Fig. 4. Sigmoid function plot. 1024 data points were sampled for the LUT.

### G. Graph for the inference

In the final step, all the kernels were combined to form the complete inference system. This was achieved through Xilinx's graph programming model, allowing for the independent design, testing, and optimization of individual kernels before their integration into the overall system. This approach facilitated smooth data transfer from one kernel's output to another's input. The entire inference process was executed within the Xilinx AI Engines, eliminating the need for external memory access or transitioning out of the AI Engine segment, resulting in efficient execution without data transfer delays. [14]

## IV. Results

This section discusses the performance of the inference implementation. All SoC simulation results were evaluated in the Xilinx AI Engine Emulator in Xilinx Vitis IDE and Xilinx Vitis Analyzer. As this inference can be interpreted as a regression problem, metrics such as accuracy and loss of the model are used to compare the performance of TensorFlow-based inference with that of FPGA-based inference.

### A. Model performance

The performance of the inference was evaluated using the accuracy and the loss as metrics. In the context of neural networks, accuracy is a commonly used metric to evaluate the performance of a model. It is calculated as the ratio of correctly predicted instances to the total number of instances in the dataset. The accuracy metric is expressed mathematically as:

$$\text{Accuracy} = \frac{\text{Number of Correct Predictions}}{\text{Total Number of Predictions}} \tag{2}$$

In practice, accuracy is reported as a percentage. This metric is crucial for assessing the model's overall performance, especially in classification tasks, and it helps gauge how well

the neural network's predictions match the actual labels in the dataset. [17]

In neural network regression tasks, the log(cosh) loss function measures the dissimilarity between predicted values and true labels. It is expressed as:

$$\text{Log(cosh) Loss} = \frac{1}{N}\sum_{i=1}^{N}\log(\cosh(y_i - \hat{y}_i)) \tag{3}$$

Where:

$N$ : Total number of data points.

$y_i$ : True labels for data point $i$.

$\hat{y}_i$ : Predicted values for data point $i$.

The log(cosh) loss is a smooth, differentiable function often used for regression tasks. It penalizes large errors while being less sensitive to outliers, making it suitable for training neural networks in regression problems. [18]

The F1 score encapsulates the balance between precision and recall in a binary classification model's performance assessment. Precision reveals how accurately the model identifies true positives within its positive predictions, while recall quantifies the model's ability to capture the actual positives by measuring the ratio of correctly identified true positives. The F1 score can be expressed as:

$$\mathrm{F}_1 = 2 \times \frac{\text{precision} \times \text{recall}}{\text{precision} + \text{recall}} \tag{4}$$

Where:

$$\text{precision} = \frac{\text{True Positives}}{\text{True Positives} + \text{False Positives}}$$

$$\text{recall} = \frac{\text{True Positives}}{\text{True Positives} + \text{False Negatives}}$$

By combining these metrics using the harmonic mean, the F1 score offers a comprehensive assessment. With a range from 0 to 1, higher F1 scores denote superior model performance, making it a pivotal metric for robustly evaluating classification models. [19]

In Table I, a noticeable reduction in accuracy and the F1-score is observed, attributed to the approximation of the sigmoid function through the use of the LUT.

TABLE I
COMPARISON OF THE ACCURACY, LOSS AND F1-SCORE OF THE TENSORFLOW- AND THE FPGA-INFERENCE

| Platform | Accuracy | Loss | F1-score |
|---|---|---|---|
| TensorFlow on CPU | 98.82% | 0.0039 | 0.9660 |
| FPGA @AI Engine | 98.80% | 0.0039 | 0.9654 |

The implementation of the proposed neural network model was successful, and the use of float32 predictions has proven to be instrumental in achieving results. In Figure 5 the inference is visualized. The upmost graphic depicts the FPGA-based inference results within the AI Engine simulation, followed by the TensorFlow-based inference results. The third sub-figure shows a visual comparison to highlight differences between the two inference implementations, mainly due to the LUT implementation on the FPGA compared to the MacBook Pro realisation in TensorFlow.

### B. Computation time comparison

The experiments demonstrated significant decrease in computation time for the FPGA-inference when compared with CPU-inference. The comparative results are presented in Table II, illustrating the significant advantage of this approach in terms of computational speed. The tests were produced, by making predictions for 1000 samples and measuring the execution time in simulation.

TABLE II
COMPARISON OF THE COMPUTATION-TIME OF THE TENSORFLOW- AND THE FPGA-INFERENCE

| Hardware | execution time / prediction |
|---|---|
| TensorFlow on CPU (MacBook Pro M1 Pro with 16GB RAM | $330.000\,\mu s$ |
| FPGA AI Engine Simulation | $15.709\,\mu s$ |

The efforts culminated in the successful implementation of a deterministic inference system, including comprehensive control over the codebase. Notably, the system operates comparably well to TensorFlow inference while ensuring the precision and predictability essential for aviation-centric applications. This achievement underscores the viability and potential of the solution in meeting the stringent demands of the industry, promising a robust and controlled framework for future developments.

## V. DISCUSSION

### A. Activation function: LUT vs. Linearisation

As the results show, the accuracy goes slightly down due to the sampling of the sigmoid function. Enhancing the LUT by increasing the number of entries allows for a more precise approximation of function values, thereby bringing the FPGA-inference closer to the TensorFlow-inference in terms of accuracy. Xilinx has of summer 2023 announced that they will release a AI-ML Engine that has more internal memory, thus allowing the implementation of bigger LUTs [20]. An alternative approach would be to use a linearized version of the sigmoid function e.g. the hard_sigmoid [21]. This approach would save memory space, as the computation of the value is similar to the look-up-step for the LUT-based implementation, without the need to actually store the function value. To further advance this study and explore its implications, future investigations could delve into the application and evaluation of deeper neural network architectures. Conducting research on power demand would present an intriguing prospect for further exploration and experimentation.

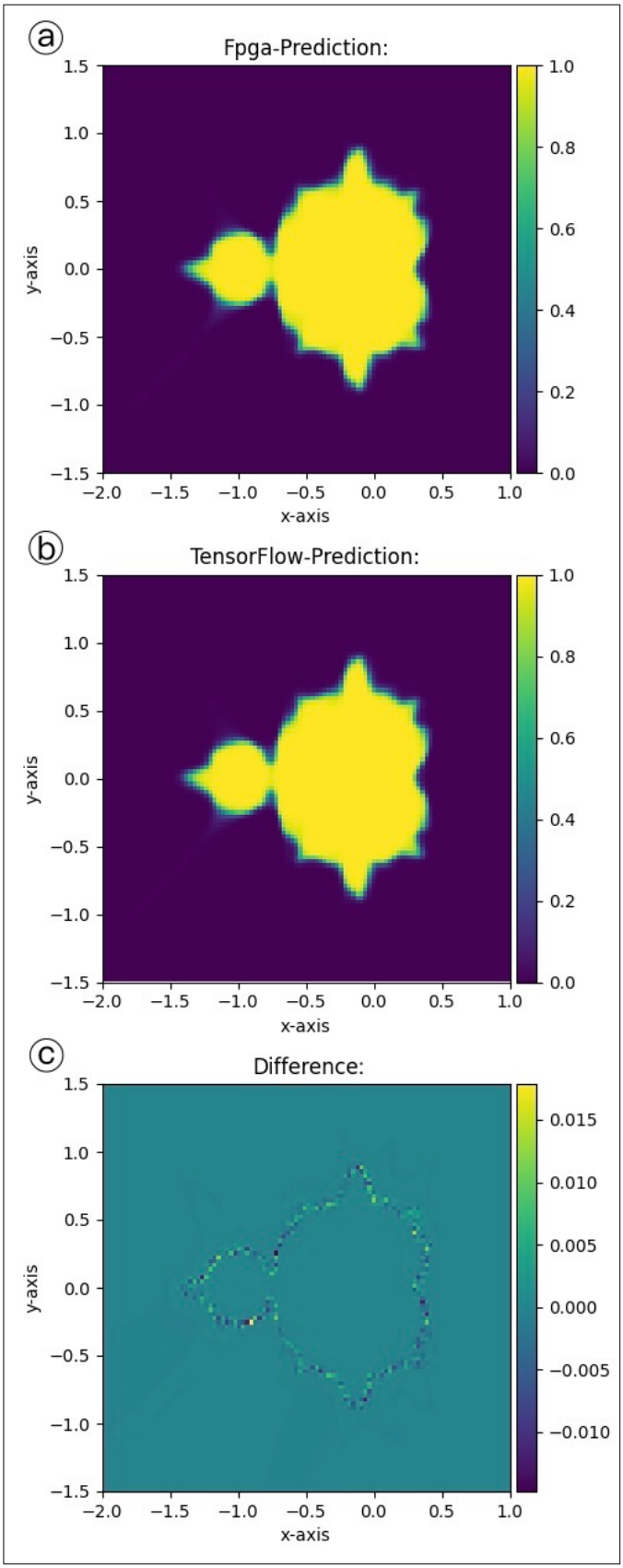


Fig. 5. Visual representation of inference capabilities through 100x100 pixel plots: TensorFlow-inference ⓐ, FPGA-inference ⓑ, and their difference ⓒ. Demonstrates nearly consistent accuracy and minimal loss across both inference methods.

## VI. Summary

In summary, this work underscores the inference capabilities and expeditious processing afforded by the AI Engines integrated into the Xilinx Versal series. The proposed implementation on the Versal FPGA SoC yielded significant acceleration when contrasted with traditional CPU-based inference methods. The achieved performance boost is indeed promising, signaling the potential of FPGA-based acceleration for machine learning workloads especially in the aviation domain. It is crucial to acknowledge that this endeavor represents merely an initial step in harnessing the full potential of the Versal AI Engines. Future avenues for research encompass further parallelization of the code and the utilization of additional AI Engines, promising even greater strides in terms of inference speed and efficiency. The synergy between adaptable hardware and versatile software frameworks, as exemplified in this study, offers a promising trajectory for the field of hardware-accelerated machine learning inference.

Furthermore, the forward-looking perspective of the Xilinx Versal series positions it as an intriguing candidate for the aviation industry's burgeoning requirements. The aviation sector places a premium on the concept of future-proofness, where hardware longevity and adaptability are of paramount importance. Xilinx Versal FPGAs align seamlessly with this vision, given their capacity to evolve with changing technological landscapes. Additionally, their potential to meet low power-demand requirements renders them a suitable candidate. As these devices are poised for certification in safety-critical applications following the DO-178B/C standards, their appeal to the aviation industry is further enhanced.

## Acknowledgment

We thank the reviewers and our colleagues Stephan Blokzyl and Thomas Petermann for their comments and discussions on this study.

## References

[1] M. Qasaimeh, K. Denolf, J. Lo, K. Vissers, J. Zambreno and P. H. Jones, "Comparing Energy Efficiency of CPU, GPU and FPGA Implementations for Vision Kernels," 2019 IEEE International Conference on Embedded Software and Systems (ICESS), Las Vegas, NV, USA, 2019, pp. 1-8, doi: 10.1109/ICESS.2019.8782524.

[2] RTCA,"DO-178C Errata" RTCA, 2020 [Online]. Available: https://www.rtca.org

[3] Y. H. Chou et al., "Deterministic Atomic Buffering," 2020 53rd Annual IEEE/ACM International Symposium on Microarchitecture (MICRO), Athens, Greece, 2020, pp. 981-995, doi: 10.1109/MICRO50266.2020.00083.

[4] NVIDIA, "CUDA C++ Programming Guide," NVIDIA, 2023. [Online]. Available: https://docs.nvidia.com/cuda/cuda-c-programming-guide/index.html

[5] Xilinx, "Versal: The First Adaptive Compute Acceleration Platform (ACAP)" Xilinx, 2022. [Online]. Available: https://docs.xilinx.com/v/u/en-US/wp505-versal-acap

[6] G. Zhou, J. Zhou and H. Lin, "Research on NVIDIA Deep Learning Accelerator," 2018 12th IEEE International Conference on Anti-counterfeiting, Security, and Identification (ASID), Xiamen, China, 2018, pp. 192-195, doi: 10.1109/ICASID.2018.8693202.

[7] R. Xu, F. Han and Q. Ta, "Deep Learning at Scale on NVIDIA V100 Accelerators," 2018 IEEE/ACM Performance Modeling, Benchmarking and Simulation of High Performance Computer Systems (PMBS), Dallas, TX, USA, 2018, pp. 23-32, doi: 10.1109/PMBS.2018.8641600.

[8] Ed. Amos R. Omondi and Ed. Jagath C. Rajapakse, "FPGA Implementations of Neural Networks", New York: Springer, 2006
[9] G. Holst und C. Krull, "Classification of Radar Targets Using Neural Networks on Systems-on-Chip", Department of Computer Science and Engineering Chalmers University of Technology and University of Gothenburg, 2021
[10] C. M. Bishop, "Pattern Recognition and Machine Learning," Springer, 2006.
[11] I. Goodfellow, Y. Bengio und A. Courville, "Deep Learning", http://www.deeplearningbook.org: MIT press, 2016.
[12] D. E. Rummelhart und J. L. McClelland, "Parallel Distributed Processing: Explorations in the Microstructure of Cognition: Foundations", The MIT press, 1986
[13] Xilinx, "AI Engines and Their Applications" Xilinx, 2018. [Online]. Available: https://www.xilinx.com/content/dam/xilinx/support/documents/white_papers/wp506-ai-engine.pdf
[14] Xilinx, "AI Engine Kernel and Graph Programming Guide" Xilinx, 2022. [Online]. Available: https://docs.xilinx.com/r/en-US/ug1079-ai-engine-kernel-coding?tocId=Bk5qNxVt~yrjda~iVyOTeQ
[15] Xilinx, "AI Engines API User Guide" Xilinx, 2022. [Online]. Available: https://www.xilinx.com/htmldocs/xilinx2022_2/aiengine_api/aie_api/doc/index.html
[16] Xilinx, "AI Engine Intrinsics" Xilinx, 2019. [Online]. Available: https://www.xilinx.com/htmldocs/xilinx2021_2/aiengine_intrinsics/intrinsics/index.html
[17] TensorFlow, "TensorFlow metrics: Accuracy" TensorFlow, 2023. [Online]. Available: https://www.tensorflow.org/api_docs/python/tf/keras/metrics/Accuracy
[18] TensorFlow, "TensorFlow losses: log_cosh" TensorFlow, 2023. [Online]. Available: https://www.tensorflow.org/api_docs/python/tf/keras/losses/log_cosh
[19] scikit-learn developers, "Precision-Recall" scikit-learn, 2023. [Online]. Available: https://scikit-learn.org/stable/auto_examples/model_selection/plot_precision_recall.html
[20] Xilinx, "Power Design Manager User Guide (UG1556)" Xilinx, 2023. [Online]. Available: https://docs.xilinx.com/r/en-US/ug1556-power-design-manager/Import-Flow
[21] TensorFlow, "TensorFlow activations: hard_sigmoid" TensorFlow, 2023. [Online]. Available: https://www.tensorflow.org/api_docs/python/tf/keras/activations/hard_sigmoid