IIETA International Information and Engineering Technology Association
Advancing the World of Information and Engineering

Ingénierie des Systèmes d'Information
Vol. 30, No. 9, September, 2025, pp. 2433-2445
Journal homepage: http://iieta.org/journals/isi

Image-Based Machine Learning Framework for LiDAR Data Analysis Accelerated on FPGA Platforms

Moteb Alghamdi*, M. Al-Asli

Department of Computer Engineering, College of Computer Science and Engineering, Taibah University, Madinah 42353, Saudi Arabia

Corresponding Author Email: mgamdi@taibahu.edu.sa

https://doi.org/10.18280/isi.300918

Received: 6 July 2025
Revised: 2 September 2025
Accepted: 10 September 2025
Available online: 30 September 2025

Keywords:
FPGA, indoor LiDAR, CNN, machine learning, FPGA accelerator, mobile robot

ABSTRACT

This study presents an image-based machine learning framework for efficient LiDAR data analysis implemented on FPGA hardware. Unlike conventional point cloud processing approaches that demand extensive computational resources, the proposed method converts raw LiDAR measurements from CSV format into compact grayscale image representations, enabling efficient feature extraction and compatibility with hardware accelerators. The workflow comprises data normalization, model training, hardware compilation, and deployment on the PYNQ-Z2 FPGA board using the Tensil AI toolchain. Experimental results show that the FPGA-accelerated convolutional neural network (CNN) achieves a classification accuracy of 98.8%, outperforming software-based implementations while reducing inference time by more than twofold and power consumption by approximately 24 times compared with a GPU. These results confirm the advantages of FPGA acceleration in achieving real-time performance with minimal energy overhead. The proposed framework offers a scalable and energy-efficient solution for LiDAR data classification and can be seamlessly integrated into embedded systems for real-time applications in autonomous navigation, smart city monitoring, and edge computing.

1. INTRODUCTION

The use of LiDAR data has become increasingly crucial for many applications, such as smart cities, autonomous vehicles, forest management, precision agriculture, and robotics. LiDAR technology is used to get a detailed picture of the shape and structure of an area of interest in either 2D or 3D point-cloud data. This data can then be utilized for object detection and classification of the designated area. However, the complex and large amount of data captured by LiDAR poses significant challenges to efficient processing in real-time decision-making applications [1, 2].

Techniques for analyzing LiDAR data have shifted from feature extraction methods that were hand-crafted to the utilization of machine learning (ML) methods such as deep learning. This has completely changed how LiDAR data is processed. The use of ML techniques cleared the way for LiDAR's solutions, which are more robust and adaptive. Training the ML algorithms on diverse and large LiDAR datasets can be beneficial in extracting meaningful patterns and features from them, which can achieve efficient classification, object detection, and semantic segmentation with high accuracy [2, 3].

While machine learning models can be useful, they pose significant challenges as they need a great deal of processing power, especially when they are used in real-time systems. FPGAs present a viable solution for LiDAR dataset analysis because the FPGA technology offers a unique blend of flexibility, energy efficiency, and parallelism. FPGAs are a well-suited processing element for the implementation of ML-based LiDAR datasets, especially for robotic or embedded systems applications. They can customize hardware architectures that align with the ML-based LiDAR datasets algorithm requirements. This allows for a real-time LiDAR implementation that uses little energy and low latency [4-6].

For easier understanding, LiDAR data can be shown as pictures that make the data easier to understand [7, 8]. Also, for machine learning algorithms like CNNs, images are a standard format that can make integrating the ML model into an FPGA easier [9, 10]. The images' hierarchical features can be extracted automatically, which benefits the classification performance greatly without the need for extensive feature engineering [11]. An image represents each processed version of the LiDAR dataset, illustrating intensity, height information, or other attributes. Therefore, different colours can visually represent the image's various features, aiding in data interpretation. In addition, images make the analysis and understanding of its object characteristics or the underlying terrain more accessible, as the pixel values correspond to various measurements (e.g., elevation).

The research in this paper goes into excellent detail about how to use machine learning methods, especially image-based deep learning methods, to look at LiDAR data on FPGA platforms. The suggested framework looks at the difficulties and chances of creating effective and reliable LiDAR processing solutions. It shows how real-time decision-making

and edge computing could be used in various robotic and embedded systems. Although numerous FPGA accelerators for LiDAR exist, most operate on raw point clouds, voxels, or hand-crafted features, which complicate hardware mapping and increase on-chip memory use. Therefore, there is a need for an alternative that can map the LiDAR frame into a compact 2D image that captures intensity/elevation patterns. This image representation enables efficient use of standard 2D CNN layers, simplifies the operator set needed for hardware acceleration, and reduces the memory usage. Therefore, there is a need for an alternative that can map the LiDAR frame into a compact 2D image that captures intensity/elevation patterns. This image representation enables efficient use of standard 2D CNN layers, simplifies the operator set needed for hardware acceleration, and reduces the memory usage. Therefore, the research objectives of this paper aim to design a compact and reproducible pipeline to convert LiDAR scan rows into 2D image representations that are compatible with lightweight CNNs, develop and train a lightweight CNN that can be used for FPGA acceleration, and deploy and evaluate the CNN model on a PYNQ-Z2 board using the Tensil AI toolchain.

This paper proposes a novel method that utilizes FPGAs and CNN algorithms to address challenges in processing LiDAR technology datasets. The proposed method proposes a unique method of converting the LiDAR dataset point cloud to an image format that is mostly compatible with FPGA hardware features. This approach shows an effective technique for achieving a reduction in processing time with low power consumption and increasing LiDAR accuracy.

Our contribution unfolds into three points as follows:

•Convert LiDAR readings to compact image representations compatible with CNNs and FPGA constraints.

•Design and train a CNN optimized for small-footprint hardware acceleration.

•Deploy and evaluate the model on a PYNQ-Z2 FPGA board using the Tensil AI toolchain, reporting accuracy, latency, and power.

The rest of the paper is structured as follows: Section 2 discusses the related work on LiDAR-FPGA-based solutions. Section 3 presents the methodology implemented. Section 4 delves into the experimental results with details on both the hardware and software findings. Section 5 discusses the results, implications, and findings. Section 6 concludes the work by suggesting potential avenues for future research.

2. RELATED WORK

This section discusses the emergence of FPGA utilization as a hardware acceleration for LiDAR data processing in many research studies.

Using quantization techniques and a model of convolutional neural networks. Several studies [12-17] suggest an FPGA solution to improve the performance of processing LiDAR data. This made the analysis of LiDAR data more flexible and reliable. While the proposed approach demonstrated the benefits of hardware acceleration in enhancing LiDAR's computationally intensive tasks, it does not utilize image-based machine learning frameworks. In the same way, Brum et al. [1] used an FPGA and quantization techniques to create a fast and accurate 3D LiDAR data object detection accelerator. Xu et al. [8] did not consider enhancing the performance of their accelerator using image-based machine learning techniques. Xiao et al. [9] used a low-cost FPGA to design an accelerator for processing real-time simultaneous localization and mapping (SLAM) algorithms, which can be adopted for a variety of LiDAR datasets. Their hardware solution uses the flexibility and ability to do many tasks at once that FPGA technology provides to create an efficient method for processing LiDAR data in SLAM. This approach does not look at using machine learning methods with FPGA hardware acceleration for tasks like semantic segmentation or object detection, beyond just SLAM methods.

Carvalho et al. [18] wanted to help automotive applications by making a ground segmentation method that works quickly, accurately, and in real time. This method uses an FPGA to separate LiDAR points that are on the ground from those that are not. However, Sugiura and Matsutani [3] did not consider the integration of machine learning algorithms. On the other hand, Chen et al. [19] utilized neural network-based solutions with FPGAs to provide a real-time accelerator that can process LiDAR data with high object detection and classification performance. While the proposed approach has demonstrated high performance by utilizing a neural network with an FPGA, it does not utilize image-based machine learning frameworks.

Another example is provided by Bai et al. [20], where they offer a solution for improving LiDAR data quality for autonomous cars. A hardware design using an FPGA and machine learning models to quickly fill in missing depth data in LiDAR point clouds in real-time. Li et al. [21] showed a voxel encoding accelerator that works well for object detection tasks. This is another example of an FPGA-LiDAR contribution. This work demonstrates how an FPGA can be an efficient solution for speedup; however, it is recommended to consider integrating machine learning techniques such as CNN, which can provide better overall accuracy and performance efficiency. One of the essential tasks for LiDAR data alignment is the performance of point cloud registration.

Sugiura and Matsutani [22] proposed an FPGA-based accelerator to speed up the point cloud registration. This work does not consider the integration of image-based machine learning methods, as it could be a useful method for enhancing the versatility and adaptability of point cloud registration.

Zolanvari et al. [23] leverage the FPGA capability with the deep learning models to improve real-time LiDAR data analysis. An example of an FPGA-image-based solution for the LiDAR data is proposed by Wu et al. [24]. This work shows a case of scene reconstruction and object detection using a range of image generation and image-based machine learning frameworks for enhancing real-time analysis of LiDAR data.

Table 1 summarizes the contribution details of the presented studies of the FPGA-ML used to optimize solutions for LiDAR dataset processing. Different FPGA platforms, LiDARs, and machine learning methods for various application domains show the vital effort and interest in contributing to this research area. However, there is clear evidence that the effort in exploring combining image-based machine learning techniques with the FPGA acceleration is rare. Therefore, our work tries to fill that gap by suggesting a new framework that uses both FPGA and deep learning to make LiDAR data analysis faster, more flexible, and scalable. This will also make current systems more useful by giving them more options. In addition, Table 2 summarizes the several platforms and toolchains that were utilized by the various presented studies, which allow compiling ML models for FPGA deployment, and illustrates why Tensil AI, which this work utilized, might be suitable for users with basic knowledge of FPGA design flow.

Table 1. Summary of key contributions in FPGA-based LiDAR data processing

Paper Ref.	Used FPGA Platform	LiDAR Type	Indoor/Outdoor	Target Application
[8]	N/A	N/A	Outdoor	Automotive Object Detection
[9]	Pynq-Z2	2D LiDAR	Indoor (SLAM)	SLAM for mobile robots and indoor navigation
[10]	ALFA FPGA	LiDAR (VLP-16, HDL-32, HDL-64, VLS-128)	Outdoor	Ground segmentation for road navigation
[11]	Enclustra Mars ZX3 FPGA	LiDAR (SPAD-based d-TOF LiDAR)	Outdoor	LiDAR histogram processing for ADAS
[12]	PYNQ-based FPGA	LiDAR (Velodyne HDL-64E)	Outdoor	Depth completion for autonomous vehicles
[13]	N/A	LiDAR (General 3D LiDAR)	Outdoor	3D object detection in LiDAR point clouds for Autonomous Vehicles
[14]	Avnet Ultra96v2, Xilinx ZCU104	LiDAR (3D Point Cloud)	Outdoor	Point cloud registration for odometry and SLAM for Autonomous Vehicles
[15]	N/A	Aerial LiDAR (ALS)	Outdoor	Classifying urban elements and semantic segmentation
[16]	N/A	N/A	Outdoor	Range image generation and LiDAR processing for Automotive, Robotics

Table 2. Comparison of AI model compilation platforms for FPGA

AI Tool	Platform	Features
Vitis AI	Xilinx FPGAs	- Optimized for Xilinx FPGAs - Uses Deep Learning Processing Unit (DPU) for acceleration - Requires quantization and model-specific compilation - Best suited for high-end Xilinx FPGAs
Vivado HLS	Xilinx FPGAs	- Converts C/C++ models into FPGA-compatible hardware - Allows designing custom ML accelerators
PYNQ (Python on Zynq)	Xilinx FPGAs	- Enables FPGA-based ML deployment using Jupyter Notebooks - Supports pre-compiled bitstreams - AI inference on FPGAs
Tensil AI	Xilinx FPGAs	- Converts ONNX/TensorFlow models into hardware-optimized execution - Works with PYNQ-Z2 and low-power FPGAs - Suitable for low-power embedded AI
OpenVINO for FPGA	Intel FPGAs	- Optimized for deep learning models on Intel FPGAs
oneAPI DPC++ Compiler	Intel FPGAs	- Uses SYCL-based programming for FPGA acceleration - Best suited for Intel FPGAs in data centers
HLS4ML	Vendor-Neutral	- Converts ML models into RTL for FPGA deployment
FINN (Xilinx Research)	Vendor-Neutral	- Optimized for quantized ML models - Designed for low-latency inference on FPGAs
TVM / Apache TVM	Vendor-Neutral	- Compiles ML models into FPGA-optimized code - Supports various FPGA architectures
DPU Compiler	Vendor-Neutral	- Converts trained models into hardware-optimized bitstreams for AI acceleration

3. METHODOLOGY

This section presents the methodology followed and details the tools, LiDAR datasets, ML models, and hardware integration that are utilized in this work. In addition, the process of converting the LiDAR dataset to images and the deployment of machine learning models on the FPGA PYNQ-Z2 platform is detailed.

To conduct this study, the publicly available LiDAR dataset on Kaggle [25] is used. The data is generated by an RPLIDAR-A1 laser range scanner that is equipped in a mobile robot. It contains 360 different LiDAR signals for four different environments: corridors, doorways, halls, and rooms. Each of the four different environments (corridors, doorways, halls, and rooms) corresponds to one of the 411 instances in the data. The dataset is a near-balanced class distribution where each environment type has a similar number of instances.

The Xilinx PYNQ-Z2 [26] is chosen as the FPGA platform, which is an evaluation board that has the Zynq-7000 All Programmable SoC (System on Chip) devices. PYNQ-Z2 is part of the PYNQ (Python + Zynq) ecosystem, which offers the programming of the FPGA using Python to simplify the development process of hardware programming. Besides the Zynq-7000 SoC, the board contains an ARM Cortex-A9 processor, which makes it a flexible platform for applications such as robotics, signal processing, and embedded systems.

Figure 1 shows the framework methodology followed, which consists of five phases. The framework starts with the model development, which consists of several steps. Then in the second phase, the model is converted. The third phase involves preparing the FPGA platform. Phases two and three serve the fourth phase, which relates to the development of the model. And the final phase is where the inferencing takes place. The following subsections present these different phases.

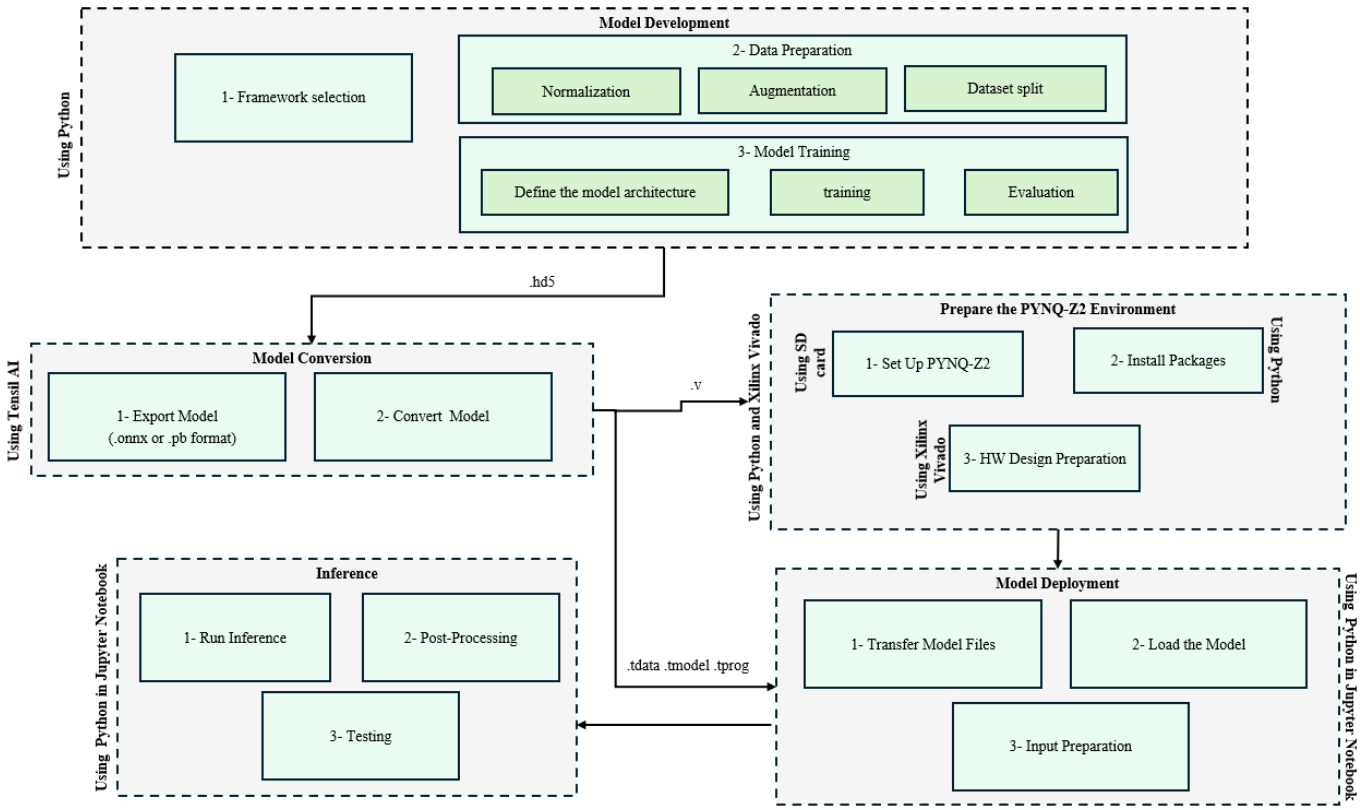

Figure 1. Overall methodology framework of the proposed system

3.1 Model development phase

This phase has three steps, which start with the framework selection to specify the environment that is best suited for the model building.

Data Preparation: For the purpose of this work, the LiDAR dataset [25] is converted from its original CSV format into images for every class in order to prepare it for CNN machine learning tasks. Firstly, the data is obtained using the Pandas library, concentrating on only the relevant measurement data. The core of the conversion process involves reshaping the data into a three-dimensional array suitable for image representation. Specifically, the measurements are restructured into a defined height and width of 20 by 18 pixels in this case—resulting in a format of (num_samples, height, width). Normalization is then applied to the pixel values, scaling them to a range between 0 and 1, which is essential for improving the performance of machine learning models. In addition, the dataset is split into two subsets using the 80-20 ratio, one subset for training and the other for validation. It is a vital step to preventing overfitting and performance assessments. The dataset used in this work is provided in the CSV format and contains a total of 411 examples. Each row contains an ID column, 360 flattened feature values, and a final label column. The LiDAR dataset is stored as CSV by removing non-signal columns (ID and label) and extracting the 1D scan vector of length L for each sample. Each vector was reshaped into a compact 2D image of size height × width, such that height × width = L (used 20 × 18 = 360). Missing values were imputed using column means, and values outside the 0.5th–99.5th percentiles were clipped. Dataset-wise min–max normalization is applied to scale values to 0,1, expanded the tensors to single-channel format (Height, Width, 1). Finally, the images were saved, and arrays were also saved as .npy files for reproducibility. This compactness is essential for the FPGA implementation on the PYNQ-Z2, where on-chip BRAM and DSP resources are limited. Table 3 summarizes the dataset and its class naming and distribution.

Table 3. Dataset summary

Item	Value
Total samples	411
Image size (H × W)	20 × 18 (360 features)
Class distribution (total)	
Class 0	109
Class 1	100
Class 2	99
Class 3	103
Train / Val	
Train (total)	328
Class 0	77
Class 1	80
Class 2	88
Class 3	83
Validation (total)	83
Class 0	32
Class 1	20
Class 2	11
Class 3	20

Model Training: To facilitate the organization of the images for the model training and validation tasks, directories are established for each class label within the two subsets. And the final step is generating and saving the images. The corresponding LIDAR data is visualized in both the training and validation sets using Matplotlib. Then the visualized images are saved as greyscale images within their respective class directories. This process helps with the image classification tasks. These steps are essential for transforming the LiDAR dataset [25] into the desired format that is suitable for image classification tasks in this work. Furthermore, these

steps enhance the deep learning techniques for analysis and prediction. The model was compiled using the Adam optimizer with the sparse categorical cross-entropy loss and trained to monitor accuracy. Training was performed with a batch size (B) of 32 and up to 300 epochs (E), learning rate (α) of 0.0001 and no momentum/decay parameter (β). A Model Checkpoint callback was used to save the best model according to validation loss. The training call used an augmented data generator and after training the best model for final evaluation is reloaded.

During training on-the-fly augmentation is applied using the Keras ImageDataGenerator with the following parameters: rotation_range = 20, width_shift_range = 0.2, height_shift_range = 0.2, shear_range = 0.2, zoom_range = 0.2, horizontal_flip = True, and fill_mode = 'nearest'. The augmentation generator was fit on the training set and then used to provide augmented batches to the network during training.

For model selection, the checkpoint with the minimum validation loss is saved and is used for evaluation. Final performance on the held-out validation set was obtained using model.evaluate(X_val, y_val), which reports validation loss and accuracy. To assess per-class behavior, predictions are computed to convert probabilities into discrete labels. The confusion matrix and a full classification report (precision, recall, F1-score, and support) are generated. The confusion matrix was visualized, displaying integer counts with class labels 0–3 on the axes.

3.2 Model conversion phase

Prior to the model training, the TensorFlow platform [27] output in HD5 format is utilized for the model conversion phase. The TensorFlow output is exported using a Python script as a .pb file. It is then compiled and converted using Tensil AI framework that is designed for the ML model to the FPGA integration process. Upon successful conversion, Tensil AI produces an RTL (Register Transfer Level) written using Verilog HW language, which is used for the preparation of the board's internal logic, and three model files: .tdata, .tmodel and .tprog. The .tdata files typically contain information related to the dataset structure and input shape. The .tmodel file represents the architecture and parameters of the trained model. The .tprog contains programming or configuration data related to deploying the model on the PYNQ-Z2 board. The Verilog file is used to prepare the HW design of the board using the Xilinx Vivado tool, while the other three files are used later to deploy the model into the PYNQ-Z2 board using Jupyter Notebook. Figure 2 illustrates the Tensil AI framework and the files it is outputting. It is worthwhile to mention that, unlike other frameworks, Tensil AI only requires the HD5 format file for the model to generate the HW files of the CNN model, which does not require a high level of experience in HW design flow.

Tensil AI facilitates the compilation and generation of the HW files of the CNN model.

3.3 Prepare the PYNQ-Z2 environment phase

The PYNQ-Z2 platform is selected for deployment because its Zynq-7000 SoC integrates an ARM Cortex-A9 processing system with programmable logic (PL), allowing the Tensil runtime and Jupyter notebooks to run directly on-board while the accelerator overlay executes in the PL. The PYNQ framework provides overlay support and Python APIs that significantly accelerate development, debugging, and reproducibility. This enables flexible integration between the host code and the Hardware files loading.

In order to deploy the CNN model in the PYNQ-Z2 FPGA board, the board must be prepared in order to receive the HW and CNN model files and perform the inference. To prepare the FPGA platform for the implementation of our model, three steps are necessary. First is the setting up of the PYNQ-Z2 board. Setting up the board involves loading the PYNQ-Z1 image, officially available on the PYNQ IO website, into an SD card and then inserting the SD card image to the board to boot the board from it. The second step is the installation of necessary packages, in which software tools such as Xilinx Vivado, the PYNQ framework, and Python with its libraries are installed. Xilinx Vivado is installed inside the host computer, while the PYNQ framework and Python are installed inside the PYNQ-Z2 board. The last step involves preparation of the HW design of the PYNQ-Z2 fabric. The HW design is an RTL file written using Verilog language and is produced by Tensil AI. It describes the internal logic that should be inside the FPGA fabric and is compiled, synthesized, and installed using the Xilinx Vivado tool.

After preparing the PYNQ-Z2 board, it can be booted as a network component with a specific IP address, and ML models can be downloaded to it using an Ethernet cable and Jupyter Notebook. It is worthwhile to mention that this preparation can be done only once, and every time the board is needed, it can be booted normally without any further preparation.

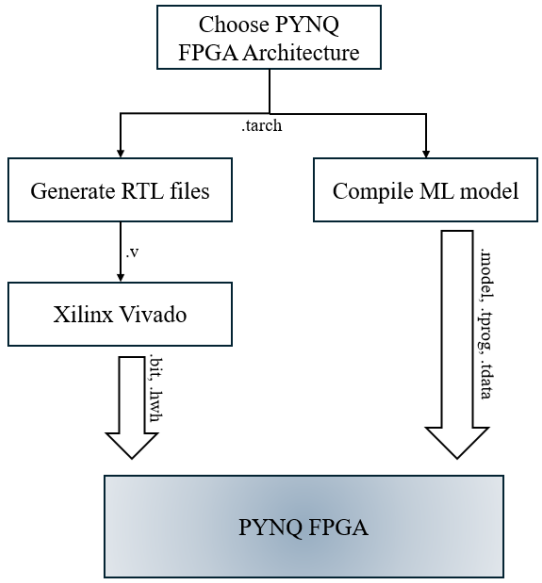

Figure 2. Tensil AI framework and its role

3.4 The model deployment phase

The CNN model was chosen to process and sort the LiDAR image dataset that was made by using multiple convolutional and pooling layers to pull out features. The CNN model architecture starts with an input layer that processes the raw data; see Figure 3. The next layer is a group of convolutional layers (Conv2D), each with a set of filters for capturing spatial hierarchies in the data. As you go deeper, the number of parameters increases. The MaxPooling2D layers disperse throughout the model, aiding in the reduction of the spatial dimensions of the map's features. In addition, they maintain

essential information, which ensures computational efficiency and the avoidance of overfitting; see Figure 3. To avoid overfitting and encourage generalization, the model includes a dropout layer that randomly disables a portion of neurons during training; see Figure 3. The final layer of the model architecture flattens the output into a one-dimensional vector, preparing the data for the dense layers. The dense layer has 128 neurons and combines the information learned in the earlier layers. It ends with an output layer that has four neurons that handle the classification tasks.

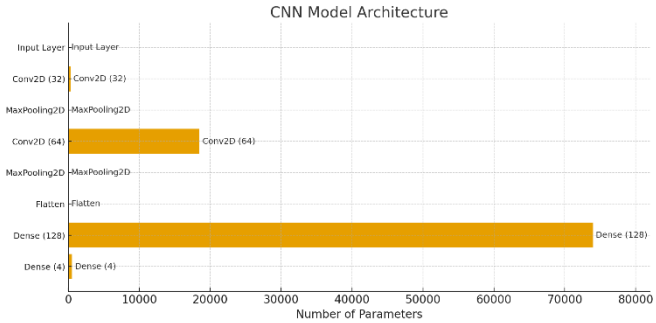

Figure 3. Model architecture and layers

3.5 Inference phase

The final phase in our framework is the inference phase, which is where the model is run to make predictions on new data. This step includes refining the output using post-processing and ensuring that it achieves the application requirements. Following that, as a last step, a comprehensive testing procedure was conducted for the model's performance validation.

This framework gives a planned way to create and use machine learning models on FPGA platforms, making sure that each step is done carefully to get the best performance and dependability.

4. RESULTS

This section presents the findings from the analysis of LiDAR data using image-based methods. The results demonstrate the model's performance in classifying various features extracted from processed images. This section is divided into two parts; the first part describes the performance of the SW model, and the second part details the PYNQ-Z2 implementation of the model and the acceleration gained.

The LidarDataFrames dataset [25] contains a collection of LiDAR point cloud data. A point cloud is a collection of data points in a 3D coordinate system that represents the spatial structure of an environment. This dataset includes various indoor attributes such as rooms, doorways, corridors, and halls. To convert this LiDAR data into greyscale images, techniques such as rasterization were used, where the point cloud data is transformed into a 2D image. This involves projecting the 3D points onto a 2D plane to create an intensity image. By assigning pixel values based on elevation or intensity metrics from the LiDAR data, images that represent the spatial features of the environment were generated. These images were then used as input for training, validating, and testing the CNN model. Each row of height values is reshaped into a 20×18 grid, scaled to 0-255, and converted into greyscale images using the PIL library. The last column is separately extracted, class_ids, to represent the classification labels.

4.1 SW version

The CNN model was trained on a desktop workstation equipped with an Intel Core i7 11800H, operating at a clock speed of 2.30 GHz and working on a 64-bit version of Windows 11. It features 16GB of DDR4 RAM, and the workstation is powered by an NVIDIA GeForce RTX 3050 graphics card. The software environment includes CUDA version 11.8.522 and cuDNN version 11.2.0. The training was conducted using Python 3.9.0, with TensorFlow 2.1.0 and Keras 2.3.1 as the primary libraries for building and training the CNN model.

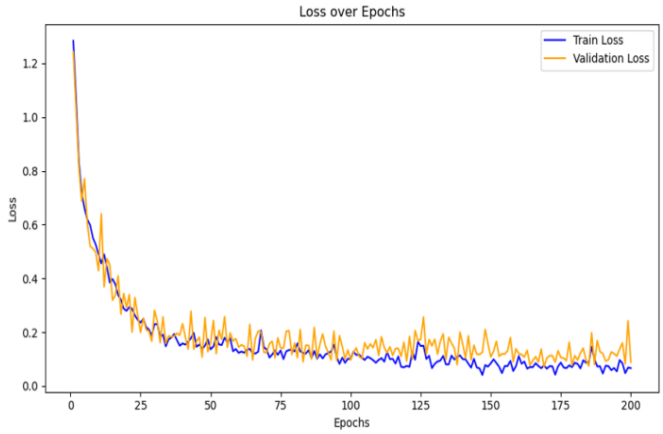

Figure 4. Training and validation loss vs. epoch

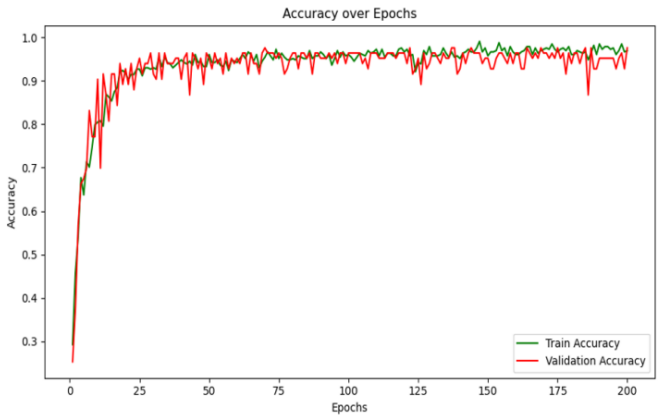

Figure 5. Training and validation accuracy vs. epoch

Figure 4 presents the training and validation loss over 200 epochs during the model training process. Initially, both training and validation losses are relatively high, and as training progresses, the training loss exhibits a steady decline, reflecting the model's ability to learn from the training data effectively. This trend continues until about epoch 150, where the training loss stabilizes at a lower value, suggesting that the model has reached a satisfactory level of fit to the training data. The validation loss also decreases throughout the early epochs, which indicates that the model generalizes well to unseen data. However, it is important to note that after epoch 150, the validation loss begins to fluctuate, suggesting potential instability in the model's performance. Despite these fluctuations, the final validation loss remains relatively low, which demonstrates that the model maintains a good fit to the

validation dataset. The gap between the training and validation loss curves is minimal, which is indicative of a well-regularized model that is not overfitting the training data. The results suggest that the model is effective in learning the underlying patterns in the data while maintaining generalizability.

Curves are from a representative run (no smoothing). The model was trained for up to 300 epochs with ModelCheckpoint saving the best model by validation loss.

Similarly, Figure 5 illustrates the training and validation accuracy of the model over 200 epochs. The training accuracy shows a steady increase and stabilizes at a high value, which suggests that the model effectively learns from the training data. The validation accuracy closely follows the training accuracy throughout the epochs. This alignment suggests that the model generalizes well to unseen data without significant overfitting. Both accuracy curves converge towards 1, indicating that the model achieves a high level of performance on both the training and validation datasets. The relatively smooth lines with minor fluctuations suggest stability in the model's learning process. The plot indicates that the model is performing well with high training and validation accuracies and reflects the effective learning and generalization capabilities.

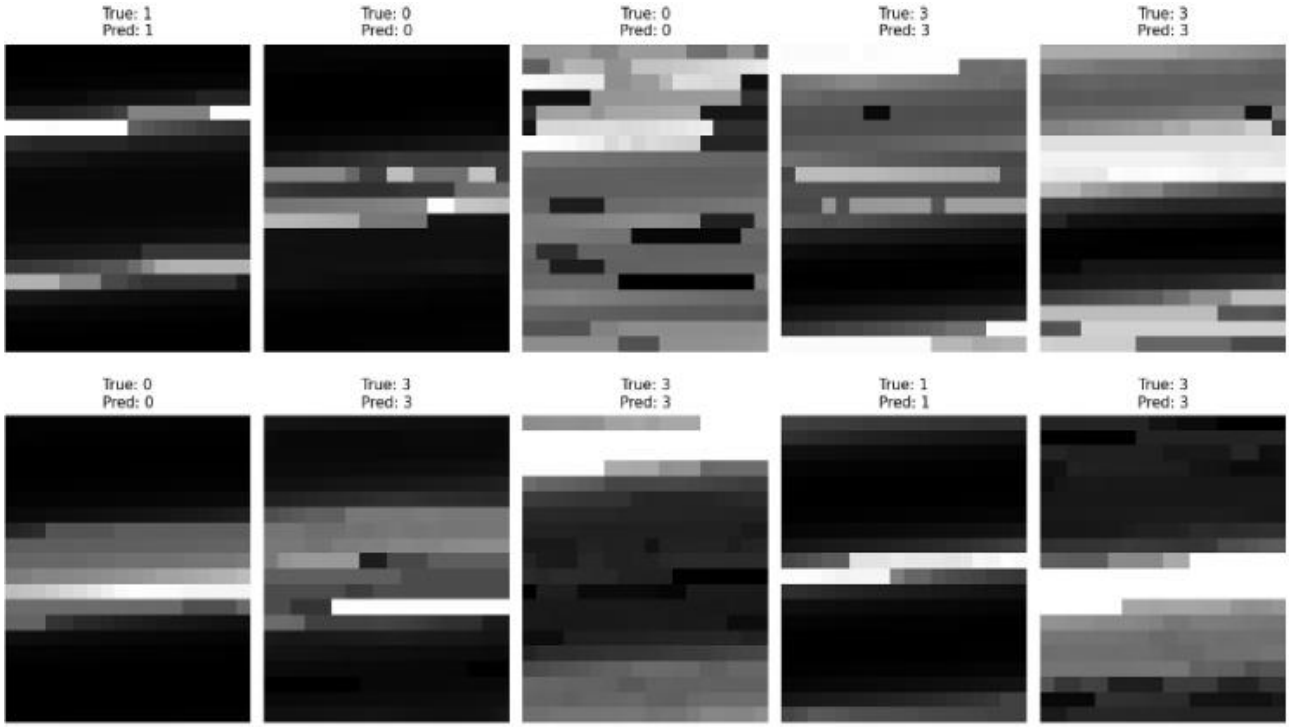

Figure 6. Samples of true and predicted images (part a)

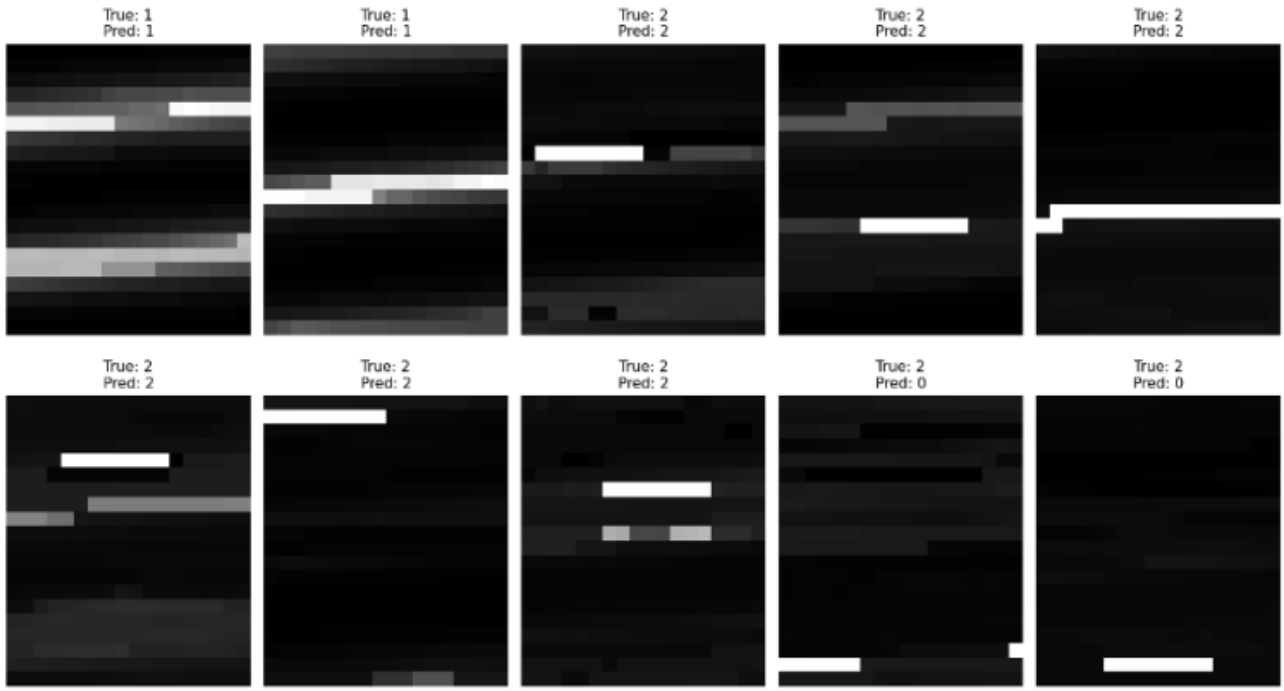

Figure 7. Samples of true and predicted images (part b)

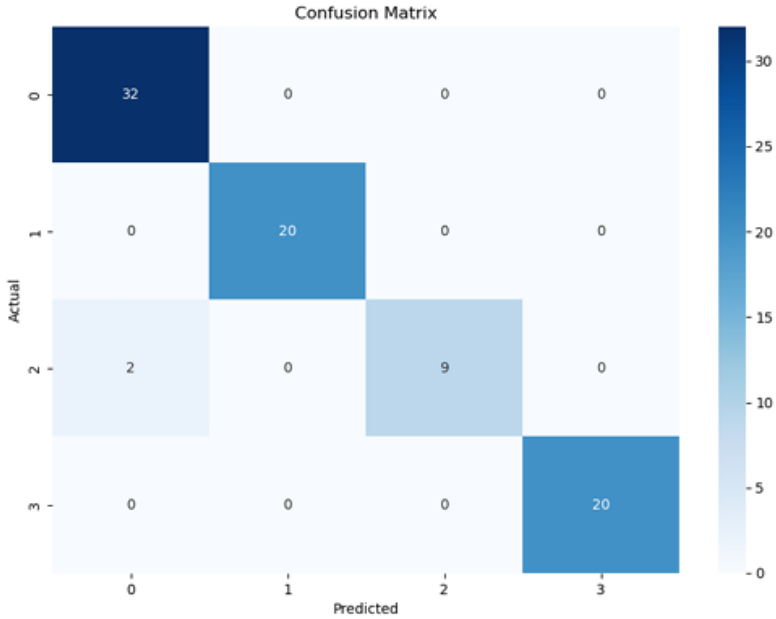

Figure 8. The confusion matrix

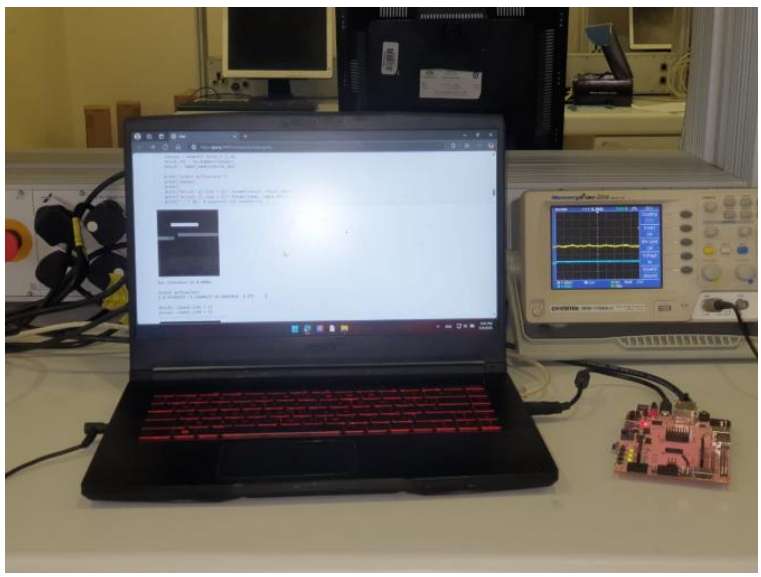

Figure 9. The hardware setup used in this work

Samples of the images for the predicted classes, along with the actual class, are shown in Figure 6 and Figure 7. It is worthwhile to mention that the pattern in the images is different among the images, and therefore, the model should be trained efficiently to correctly classify them. As can be seen in Figure 8, the confusion matrix visualizes the performance of a classification model across the four classes labelled 0, 1, 2, and 3. Each cell in the matrix indicates the number of instances predicted by the model versus the actual instances. The model was able to correctly classify all the classes expected for two images from class 3, which were classified as class 0. The reason for this misclassification could be due to the imbalance in the dataset, where class 0 had more samples than class 2. On the held-out validation set of 83 samples, the best saved model achieved a high overall accuracy as reported. The classification report further breaks down performance into precision, recall, and F1-score for each of the four classes (corridors, doorways, halls, and rooms). The confusion matrix highlights correct predictions along the diagonal and misclassifications off-diagonal. Notably, class 2 (halls) shows fewer validation samples (11 instances), which should be considered when interpreting its per-class metrics. The heatmap visualization generated provides an intuitive overview of these results, with darker diagonal cells reflecting stronger class-specific accuracy.

Table 4. The classification report of our model

	Precision	Recall	F1-Score	Accuracy
0	0.94	1.00	0.97	0.98
1	1.00	1.00	1.00	
2	1.00	0.82	0.90	
3	1.00	1.00	1.00	

The classification report is shown in Table 4 and summarizes the performance metrics of a model across the four classes, including precision, recall, F1-score, and accuracy metrics. For class 0, the model has a precision of 0.94, meaning that 94% of instances predicted as class 0 were correct. The recall is 1.00, indicating that all actual instances

of class 0 were correctly identified. The F1-score, which balances precision and recall, is 0.97, reflecting strong performance. For class 1, the model achieved a perfect precision and recall of 1.00, resulting in an F1-score of 1.00 as well. Similarly for class 3, the model also achieved perfect scores across all metrics. The model is highly effective in identifying instances of these classes without any false positives or negatives. Class 2 shows a precision of 1.00, which suggests that all predicted instances of this class were correct. However, the recall is 0.82, which suggests that 82% of actual instances were correctly identified, leading to an F1-score of 0.90.

The metrics used are denoted by the following Eqs. (1-4) as follows:

$$Accuracy = \frac{TP + TN}{TP + TN + FP + FN} \tag{1}$$

$$Precision = \frac{TP}{TP + FP} \tag{2}$$

$$Recall = \frac{TN}{TN + FP} \tag{3}$$

$$F1\,Score = 2 \times \frac{Precision \times Recall}{Precision + Recall} \tag{4}$$

4.2 HW version

The results presented in this subsection demonstrate the use of PYNQ-Z2 FPGA platform for the proposed framework for the analysis of LiDAR data. The hardware implementation of the proposed framework, shown in Figure 9, Figure 10, and Figure 11, was carried out and was tested using the same 83 images used for testing the SW version. The FPGA-based solution completed the inference time in just 0.6207 seconds with an average inference time of about 0.0077 seconds per image. As can be shown in the Jupyter Notebook in Figure 10, first the HW design bitstream file (.bit) generated by Xilinx Vivado to the FPGA fabric is loaded, and using the helper functions in the script, a sample image along with its true class is shown (Figure 11). Then, the compiled model is loaded to the FPGA, and test images are fed as input to the model inside the FPGA. The FPGA, in turn, performs the prediction and produces the output. The script then reports the number of images, average inference time, and accuracy. For more comprehensive image analysis and illustration, a summary of more sample images, including inference time, actual, and predicted classes, is also reported in Figure 11.

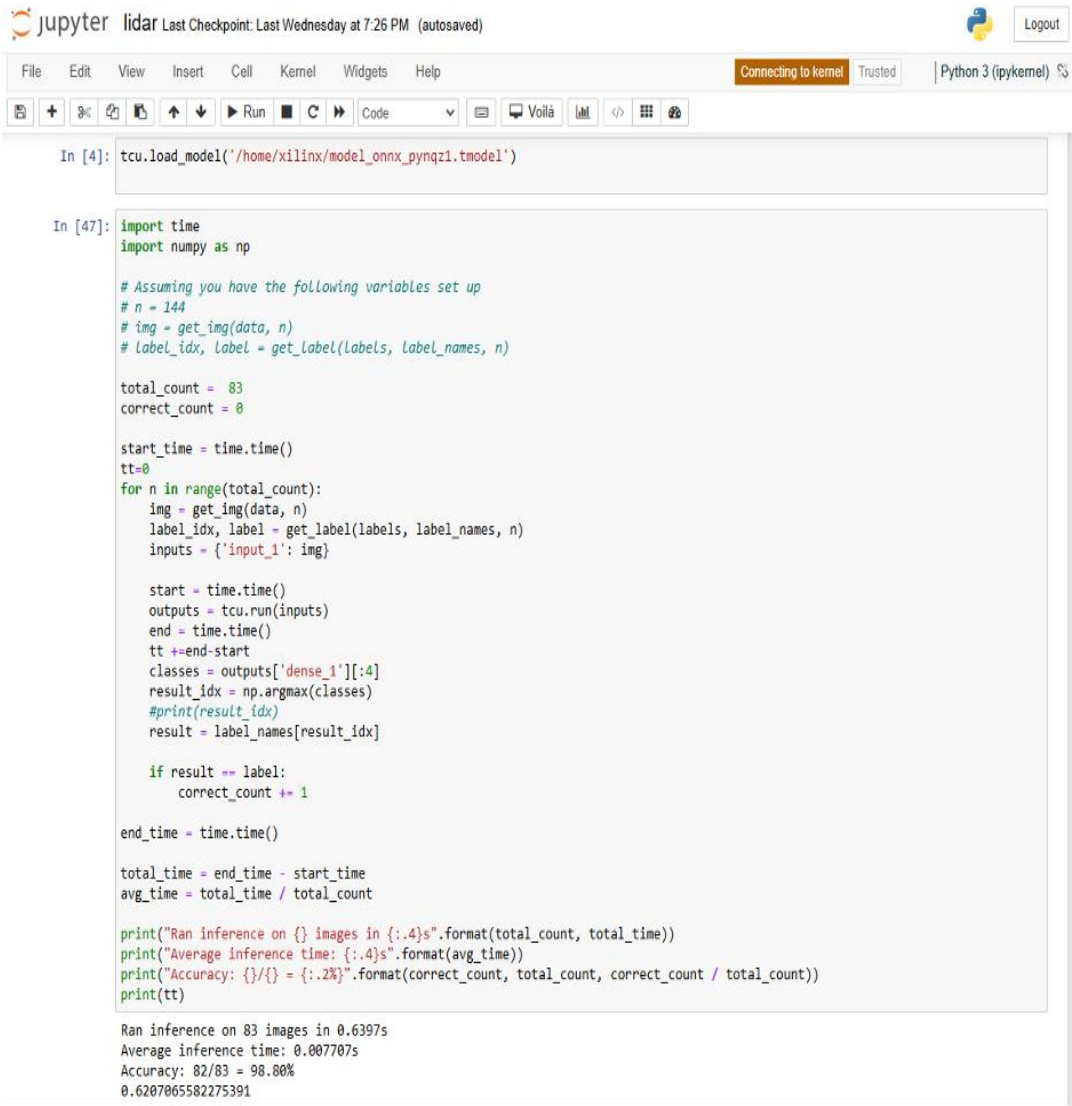

Figure 10. Snippet of Python script used for FOGA inference on Jupyter Notebook

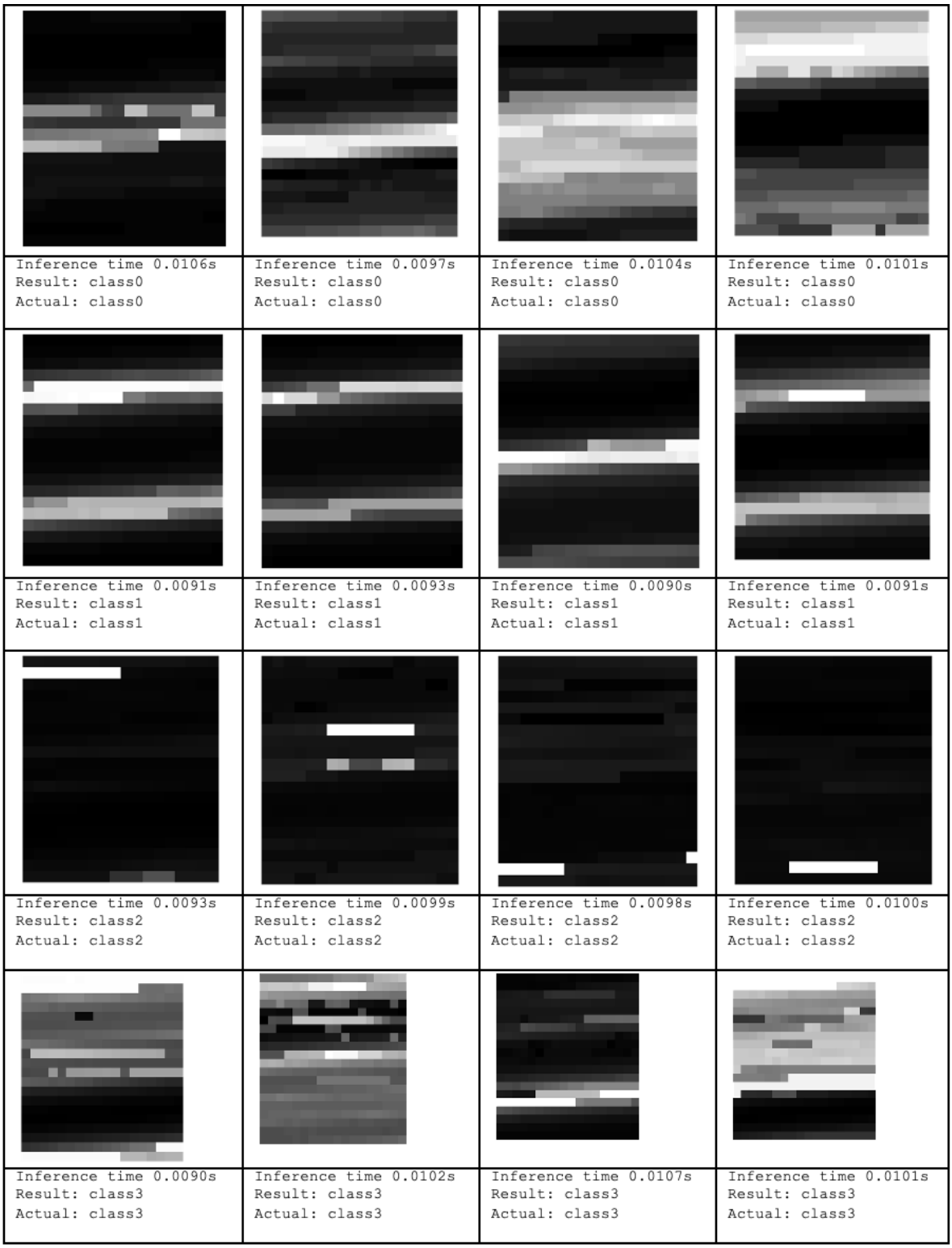

Figure 11. A summary of more sample images, including inference time, actual and predicted classes

Table 5 summarizes the findings of the model for both the SW and HW implementations using the test dataset, which consists of 83 images. The hardware version is faster than the software version by more than 2x, and the average inference time per image is also improved, making HW more efficient for real-time. The hardware optimization does not compromise model performance, as the accuracy of the SW and the HW versions are almost identical. The FPGA-based implementation of the model consumes only 1.616W in total, with 1.472W attributed to dynamic power usage. In contrast, GPU-based implementations, such as the NVIDIA RTX 3050, measured and used in the comparative study, consume around 37.7W. The FPGA-based solution consumes nearly 24x less power than the SW GPU-based solution.

The results highlight the advantages of the proposed real-time image-based framework for LiDAR data analysis using the PYNQ-Z2 FPGA. The convolutional neural networks and the performance capabilities of the PYNQ-Z2 FPGA could create many innovative applications in indoor/outdoor robotics equipped with LiDAR and the medical field, where real-time and low-cost devices are necessary for running ML models. Furthermore, the energy efficiency of FPGA-based

implementations makes them suitable for edge AI LiDAR applications to reduce the reliance on cloud computing while maintaining high accuracy and low latency. Future research directions may include the exploration of end-to-end deep learning solutions for LiDAR data analysis, the incorporation of advanced neural network architectures, and the investigation of hardware-aware model optimization techniques to further enhance the efficiency and real-time performance of the system. Further support for FPGA platforms for ML models, such as pre-trained models, could hugely impact the adoption of the FPGA platforms in various AI applications.

Table 5. Summary of the findings of the model for both the SW and HW implementations using the test dataset

	SW Version	HW Version
Inference Time	1.3888 s	0.6207 s
Average Inference Time Per Image	0.01673 s	0.0077 s
Accuracy	98%	98.80%
Power consumption	37.70W	1.61W

5. DISCUSSION

The results indicated that the proposed image-based machine learning framework for analyzing the LiDAR dataset is effective, particularly its deployment on an FPGA platform. Transforming the raw LiDAR data into images has facilitated the use of the convolutional neural network (CNN) method, which helps in classification accuracy and feature extraction tasks. In addition, further performance optimization resulted from integrating FPGA-based acceleration, which provided real-time inference along with efficient energy consumption.

The main achievement of this work is improving the accuracy and inference speed of the LiDAR dataset classification. The FPGA-based implementation achieved an accuracy of 98.80%, which is slightly higher than the software-based implementation (98%). It also took 2 times less time for the FPGA version to draw conclusions than the software version (0.6207 seconds for 83 test images, or 0.0077 seconds per image), which shows that it can be used in real-time situations. In real-time applications, the inference time is a vital element, particularly in autonomous navigation, robotics, and edge computing. The performance boost comes from the PYNQ-Z2 FPGA's efficient hardware and the ability to do computing in parallel. This work's FPGA implementation consumed ultra-low power, just 1.616 W, which is about 24 times more efficient than the GPU-based processing used in the study. These enhancement results show that our proposal is an efficient real-time candidate solution for applications such as autonomous navigation, robotics, and edge computing.

In comparison with the related studies listed in the related work section, our work provides a different approach that facilitates the integration of deep learning models on FPGA platforms. It leverages CNN-based classification on LiDAR data, represented as images. Unlike our approach, which is highly compatible with CNN architectures, the other approaches [1, 12, 28, 29] are focused mostly on traditional feature extraction techniques or direct point cloud computations. These approaches did not explore image transformation techniques that enable seamless deployment of the ML models to FPGAs. Studies [20, 22] utilized a CNN-based method for processing the LiDAR dataset; however, it utilized high FPGA resources, while our method achieved a balanced solution with high accuracy and speed and low power consumption (1.616W).

Furthermore, the LiDAR data-to-image transformation facilitates the data interpretation and allows for the reuse of well-established image processing and deep learning methodologies. The CNN architecture works well for classifying images, and the proposed framework takes advantage of this to find intensity-related and spatial patterns in the dataset that might be hard to handle with traditional point-cloud-based processing methods. The results provided a more intuitive interpretation of the LiDAR dataset features and improved the classification capabilities.

Although the proposed framework showed many advantages, some drawbacks should be acknowledged. One of them is that the dataset used does not represent the complexity of real-world environments and might not be a candidate that can be generalized. Therefore, the incorporation of diverse datasets that have different outdoor scenes, complex obstacles, varying lighting conditions, and different environmental settings can address this limitation. Also, the CNN model can be made even more efficient on FPGA platforms by using techniques like quantization, hardware-aware neural architecture search (NAS), and pruning, along with other optimizations.

The preprocessing overhead associated with the initial conversion of the LiDAR dataset to the image format presents another limitation. Therefore, exploring more efficient direct feature extraction techniques or efficient encoding strategies to convert LiDAR raw data into images is recommended. In addition, the performance can be further enhanced by integrating extra neural networks, like lightweight deep learning frameworks or transformer-based models, which could be useful to explore in future work.

The findings in this work proved the viability of combining FPGA acceleration with the CNN method to analyze and classify LiDAR data. The result paves the way for integrating the energy-efficient hardware elements and real-time-capable LiDAR classification systems, particularly in applications such as autonomous navigation, edge computing, and robotics. In addition, this work shows a promising solution for elevating the real-time LiDAR processing.

6. CONCLUSIONS

In this work, a novel LiDAR dataset analysis framework combines FPGA acceleration and image-based machine learning methods to improve the inference efficiency and classification performance of the LiDAR dataset. The work showcases a significant enhancement in processing, accuracy, and power efficiency by utilizing image representations for the LiDAR point cloud data, which are then processed using CNN methods.

The findings show that the FPGA implementation version achieved better performance than the traditional software-based models as a result of a reduction in inference time by more than 50% and a high accuracy rate. In addition, the energy savings that the FPGA platform has offered underscore the importance of such solutions for applications such as autonomous navigation, edge computing, and robotics, where low-power accelerators are a vital element.

Not only a solution for LiDAR data processing, the proposed framework can be adapted for many applications that operate on low-power real-time sensors, such as smart cities monitoring, robotic vision, and autonomous vehicles. So, bridging the gap between hardware acceleration based on FPGAs and CNN methods can give us a reliable and scalable way to process LiDAR data in real-time situations. The proposed framework has great potential with advancements in FPGA technology and machine learning methods to enhance LiDAR data utilization across a wide range of applications.

ACKNOWLEDGMENT

This work is supported by Taibah University.

REFERENCES

[1] Brum, H., Véstias, M., Neto, H. (2024). LiDAR 3D object detection in FPGA with low bitwidth quantization. In International Symposium on Applied Reconfigurable Computing, pp. 90-105. https://doi.org/10.1007/978-3-031-55673-9_7

[2] Wang, R., An, M., Shao, S., Yu, M., Wang, S., Xu, X. (2021). Lidar sensor-based object recognition using machine learning. Journal of Russian Laser Research, 42(4): 484-493. https://doi.org/10.1007/s10946-021-09986-x

[3] Sugiura, K., Matsutani, H. (2022). A universal LiDAR SLAM accelerator system on low-cost FPGA. IEEE Access, 10: 26931-26947. https://doi.org/10.1109/ACCESS.2022.3157822

[4] Bai, L., Lyu, Y., Xu, X., Huang, X. (2020). Pointnet on FPGA for real-time lidar point cloud processing. In 2020 IEEE International Symposium on Circuits and Systems (ISCAS), Seville, pp. 1-5. https://doi.org/10.1109/ISCAS45731.2020.9180841

[5] Flottmann, M., Eisoldt, M., Gaal, J., Rothmann, M., Tassemeier, M., Wiemann, T., Porrmann, M. (2021). Energy-efficient FPGA-accelerated LiDAR-based SLAM for embedded robotics. In 2021 International Conference on Field-Programmable Technology (ICFPT), Auckland, New Zealand, pp. 1-6. https://doi.org/10.1109/ICFPT52863.2021.9609934

[6] Lis, K., Kryjak, T., Gorgoń, M. (2025). LiFT: Lightweight, FPGA-tailored 3D object detection based on LiDAR data. In International Workshop on Design and Architectures for Signal and Image Processing, pp. 28-40. https://doi.org/10.1007/978-3-031-87897-8_3

[7] Lyu, Y., Bai, L., Huang, X. (2018). ChipNet: Real-time LiDAR processing for drivable region segmentation on an FPGA. IEEE Transactions on Circuits and Systems I: Regular Papers, 66(5): 1769-1779. https://doi.org/10.1109/TCSI.2018.2881162

[8] Xu, Y., Tong, X., Stilla, U. (2021). Voxel-based representation of 3D point clouds: Methods, applications, and its potential use in the construction industry. Automation in Construction, 126: 103675. https://doi.org/10.1016/j.autcon.2021.103675

[9] Xiao, A., Huang, J., Guan, D., Cui, K., Lu, S., Shao, L. (2022). Polarmix: A general data augmentation technique for lidar point clouds. Advances in Neural Information Processing Systems, 35: 11035-11048.

[10] Shinohara, T., Xiu, H., Matsuoka, M. (2020). FWNet: semantic segmentation for full-waveform LiDAR data using deep learning. Sensors, 20(12): 3568. https://doi.org/10.3390/s20123568

[11] Koo, J., Klabjan, D., Utke, J. (2020). Combined convolutional and recurrent neural networks for hierarchical classification of images. In 2020 IEEE International Conference on Big Data (Big Data), Atlanta, GA, USA, pp. 1354-1361. https://doi.org/10.1109/BigData50022.2020.9378237

[12] Silva, J., Pereira, P., Machado, R., Névoa, R., Melo-Pinto, P., Fernandes, D. (2022). Customizable FPGA-based hardware accelerator for standard convolution processes empowered with quantization applied to LiDAR data. Sensors, 22(6): 2184. https://doi.org/10.3390/s22062184

[13] Pistellato, M., Bergamasco, F., Bigaglia, G., Gasparetto, A., Albarelli, A., Boschetti, M., Passerone, R. (2023). Quantization-aware NN layers with high-throughput fpga implementation for edge AI. Sensors, 23(10): 4667. https://doi.org/10.3390/s23104667

[14] Tasci, M., Istanbullu, A., Tumen, V., Kosunalp, S. (2025). FPGA-QNN: Quantized neural network hardware acceleration on FPGAs. Applied Sciences, 15(2): 688. https://doi.org/10.3390/app15020688

[15] Zhang, Q., Cao, J., Zhang, Y., Zhang, S., Zhang, Q., Yu, D. (2019). FPGA implementation of quantized convolutional neural networks. In 2019 IEEE 19th International Conference on Communication Technology (ICCT), Xi'an, China, pp. 1605-1610. https://doi.org/10.1109/ICCT46805.2019.8947168

[16] Sun, M., Li, Z., Lu, A., Li, Y., et al. (2022). Film-qnn: Efficient fpga acceleration of deep neural networks with intra-layer, mixed-precision quantization. In Proceedings of the 2022 ACM/SIGDA International Symposium on Field-Programmable Gate Arrays, pp. 134-145. https://doi.org/10.1145/3490422.3502364

[17] Wei, X., Liu, W., Chen, L., Ma, L., Chen, H., Zhuang, Y. (2019). FPGA-based hybrid-type implementation of quantized neural networks for remote sensing applications. Sensors, 19(4): 924. https://doi.org/10.3390/s19040924

[18] Carvalho, J., Cunha, L., Pinto, S., Gomes, T. (2024). FESTA: FPGA-enabled ground segmentation technique for automotive LiDAR. IEEE Sensors Journal, 24(22): 38005-38014. https://doi.org/10.1109/JSEN.2024.3470591

[19] Chen, G., Kirtiz, G.A., Wiede, C., Kokozinski, R. (2021). Implementation and evaluation of a neural network-based lidar histogram processing method on fpga. In 2021 IEEE 34th International System-on-Chip Conference (SOCC), Las Vegas, NV, US, pp. 1-6. https://doi.org/10.1109/SOCC52499.2021.9739527

[20] Bai, L., Zhao, Y., Elhousni, M., Huang, X. (2020). DepthNet: Real-time LiDAR point cloud depth completion for autonomous vehicles. IEEE access, 8: 227825-227833. https://doi.org/10.1109/ACCESS.2020.3045681

[21] Li, X., Ren, A., Tan, Y., Li, X., et al. (2022). Vea: An fpga-based voxel encoding accelerator for 3d object detection with lidar. In 2022 IEEE 40th International Conference on Computer Design (ICCD), Olympic Valley, CA, USA, pp. 509-516. IEEE. https://doi.org/10.1109/ICCD56317.2022.00081

[22] Sugiura, K., Matsutani, H. (2023). An efficient accelerator for deep learning-based point cloud registration on FPGAs. In 2023 31st Euromicro International Conference on Parallel, Distributed and Network-Based Processing (PDP), Naples, Italy, pp. 68-75. https://doi.org/10.1109/PDP59025.2023.00018
[23] Zolanvari, S.M., Ruano, S., Rana, A., Cummins, A., Da Silva, R.E., Rahbar, M., Smolic, A. (2019). DublinCity: Annotated LiDAR point cloud and its applications. arXiv preprint arXiv:1909.03613. https://doi.org/10.48550/arXiv.1909.03613
[24] Wu, T., Fu, H., Liu, B., Xue, H., Ren, R., Tu, Z. (2021). Detailed analysis on generating the range image for lidar point cloud processing. Electronics, 10(11): 1224. https://doi.org/10.3390/electronics10111224
[25] Alhmiedat, T. (2024). LidarDataFrames. https://www.kaggle.com/datasets/tareqalhmiedat/lidardataframes.
[26] AMD. PYNQ-Z2. https://www.amd.com/en/corporate/university-program/aup-boards/pynq-z2.html.
[27] TensorFlow. TensorFlow. https://www.tensorflow.org/.
[28] Abdoune, L., Fezari, M., Dib, A. (2024). Indoor sound classification with support vector machines: State of the art and experimentation. International Journal of Computational Methods and Experimental Measurements, 12(3): 269-279. https://doi.org/10.18280/ijcmem.120307
[29] Seniguer, A., Iratni, A., Aouache, M., Yakoubi, H., Mekhermeche, H. (2025). A machine learning-based tool for indoor lighting compliance and energy optimization. International Journal of Computational Methods and Experimental Measurements, 13(2): 259-271. https://doi.org/10.18280/ijcmem.130205

NOMENCLATURE

N	total number of samples in the dataset
H	image height
W	image width
F	total number of features per sample
B	batch size used in training
E	maximum number of training epochs

Greek symbols

α	learning rate of optimizer
β	momentum/decay parameter

Subscripts

Train	training set
Val	validation set