RESEARCH ARTICLE

Design and Implementation of Artificial Intelligence Models Using Deep Neural Networks on Reconfigurable VLSI Systems for Autonomous Driving

Muruganantham S[1]*, Santhosh Kumar C[2], Mary Jacob[3], Ismailova Zukhra[4], Anil Kumar[5], Ali Bostani[6], K. Sathishkumar[7]

[1]Department of Computer Technology and Information Technology, Kongu Arts and Science College (Autonomous), Erode – 638 107, Tamil Nadu, India.

[2]Assistant Professor, Department of Computer Science and Engineering SRM Institute of Science and Technology, Ramapuram, Chennai, Tamil Nadu, India.

[3]Assistant Professor, Department of Computer Science, Kristu Jayanti Deemed to be University, Bengaluru, India.

[4]Professor, Doctor of Pedagogical Sciences, Tashkent Institute of Irrigation and Agricultural Mechanization—National Research University, Tashkent, Uzbekistan.

[5]School of Computing, DIT University, Makkawala, Dehradun, 248009, Uttarakhand, India.

[6]Associate Professor, College of Engineering and Applied Sciences, American University of Kuwait, Salmiya, Kuwait.

[7]Assistant Professor, Department of Computer Science, Erode Arts and Science College (Autonomous), Erode, Tamil Nadu, India.

KEYWORDS:
Autonomous Driving
Deep Neural Networks
FPGA
Reconfigurable VLSI
Object Detection

Abstract

Autonomous vehicles use object detection in real time, which is an important aspect of navigation and decision-making systems. Nevertheless, conventional computing architecture like CPUs and GPUs are usually inadequate to support the required latency, power consumption, and real-time demands within embedded automotive systems. This paper gives details of a generic design and implementation of object detection models using deep neural networks on reconfigurable very large-scale integration (VLSI) systems including field-programmable gate arrays (FPGAs). The quantized and compressed architecture of DNN are shown as combining a system-level co-design approach with an FPGA platform by means of optimized mapping of hardware and parallel dataflow design. The framework has been proposed based on low-latency, high-throughput, energy-efficient inference, which can be brought to the edge when safety is required. The process of simulation and hardware synthesis entails MATLAB, Simulink, HDL Coder, and Xilinx Vivado, with experimental analysis being carried out on real datasets, such as KITTI or BDD100K. Experiments show that indeed there is a huge gain in the number of inferences per second and resource consumption as well as power generations when compared to typical CPU/GPU deployment. The results support the conclusion on the usefulness of reconfigurable VLSI platforms as an alternative hardware solution to building autonomous driving systems by AI in the future.

Authors' e-mail ID: muruganandham.s@gmail.com, cjsksag@gmail.com, maryjacob@kristujayanti.com, zukhra.ismoilova@gmail.com, dahiyaanil@yahoo.com, abostani@auk.edu.kw, sathishmsc.vlp@gmail.com

Authors' Orcid ID: 0000-0003-2027-7560, 0000-0003-4973-2352, 0000-0003-4016-3544, 0009-0006-8605-2443, 0000-0003-0982-9424, 0000-0002-7922-9857, 0000-0002-7643-4791

ARTICLE HISTORY:
Received 30-03-2025
Revised 22-06-2025
Accepted 01-08-2025

How to cite this article: Muruganantham S, et al., Design and Implementation of Artificial Intelligence Models Using Deep Neural Networks on Reconfigurable VLSI Systems for Autonomous Driving, Journal of VLSI circuits and systems, Vol. 7, No.1, 2025 (pp. 145-154).

DOI:
https://doi.org/10.31838/JVCS/07.01.16

Introduction

The unprecedented development of autonomous vehicles (AVs) has necessitated the need to have in place, reliable high-performance systems of perceptions that can work within real-time constraints. The most important of those tasks include lane detection, pedestrian tracking, and obstacle avoidance; computer vision algorithms are key to making those tasks a reality, with deep neural networks (DNNs) becoming the most prominent of methods as they demonstrate high precision and flexibility in dealing with a complex environment [1,2]. There is, however, a common tendency that using DNNs on general-purpose GPUs and CPUs will result in prohibitive latency and power use, particularly in embedded automotive environments where energy consumption and deterministic timing are paramount [3,4].

To overcome these shortcomings, reconfigurable very large-scale integration (VLSI) architecture, namely, field-programmable gate arrays (FPGAs), have recently become a popular hardware substrate to implement the AI workloads in safety-critical workloads [5,6]. FPGAs present numerous benefits amid which custom parallel data paths, pipelined processing, and dynamic reconfiguration could be used to speed DNN inference at low power/latency overheads [7,8].

However, streaming DNN models onto VLSI fabric is not trivial and a rather tricky co-design of algorithmic and architectural layers is required. Memories bottleneck, fixed-point arithmetic, or resource-limited mapping are the challenges that should be considered allowing quantization, compression, and the effective construction of hardware-constrained models.

The following are the contributions of this work:

- Design and optimization of a quantized, pruned DNN model that is made to fit real-time object detection on an AV environment.
- Devising an approach in hardware/software co-design that allows application to be put effectively on reconfigurable VLSI systems.
- Simulation should implement and prove with MATLAB, Simulink, HDL coder, and Xilinx Vivado used for hardware synthesis.
- Comparison with real-world datasets (e.g., KITTI, BDD100K) and evaluation values (e.g., inference latency, hardware utilization, and detection accuracy).

Figure 1 shows high-level architecture of the proposed framework in detail, including how the trained AI model is to be integrated into FPGA-based VLSI system with major processing blocks and dataflow routes.

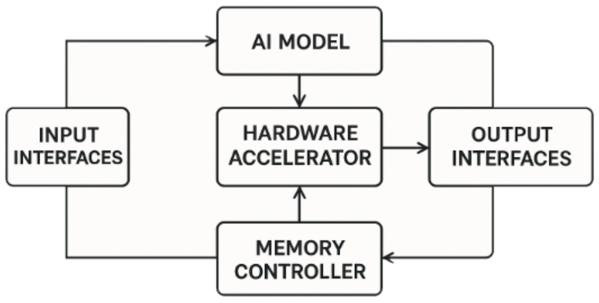

Fig. 1: High-level system architecture for field-programmable gate arrays-based deep neural network inference in autonomous vehicles.

Related Works

A combination of deep learning and hardware acceleration has become a subject of high priority of research in recent years, particularly in the context of real-time applications like self-driving cars. There is extensive literature on accelerating convolutional neural networks (CNNs) and other forms of DNNs to demonstrating low-latency, low-power, and throughput demands on programmable hardware settings such as FPGAs.

To achieve this, we presented here, in reference [1], a fast inference architecture based on model pruning and weight compression, which makes it possible to perform sparse matrix operations on-chip and save energy. Based on this, references [2,9] applied YOLOv3 on pipelined computing framework FPGA and was able to show high frame rate with capability of object detection in real time at resource-limited devices.

Another main area is quantization. In reference [3], a cross-domain jointly quantized scheme was suggested in Zynq-based systems, which is located to effectively cut down bit-widths without spoiling detection accuracy. In the meantime, reference [10] proposed a pruning-plus-folding approach, which compresses network and layers, to reduce compute and memory demands for efficient edge deployment.

The other performance bottleneck has been on the use of on-chip memory. The approach in reference [11] developed an SRAM-efficient inference machine that reduced the latency of accessing the memory bandwidth. In an analogous fashion, references [12,13] got a fully optimized ResNet deployment onto Xilinx ultra scale FPGAs and also with a full utilization of shared resources and pipelined operations, wherein it demonstrated impressive performance and energy efficiency benefits.

Architecturally, on the innovation front, reference [6] put forth data maximizing the reuse and computational parallelism by the proposed systolic array-based accelerator. A general overview of VLSI architecture in AI edge computing was given in reference [4], where the limit of accuracy and power consumption of both ASIC and FPGA design is a tradeoff and challenge when it comes to deploying DNN.

Proposals in reference [14], which introduced a low-latency CNN to traffic analysis, and references [7,15], which introduced energy-efficient approximate computing using multipliers circuit, investigated some potential practical implementations. In reference [16], the use of high-level synthesis (HLS) in real-time semantic segmentation on embedded platforms was presented.

In reference [17], a comparison benchmarking study between object detection models and FPGA architecture was conducted to characterize the accuracy/hardware trade-offs. References [18,19] were a study of the value of low-precision arithmetic to drive inference at low power and maximizing acceleration, whereas reference [20] outlined the implementation of CNN models targeted to automotive, cost-sensitive VLSI.

Nevertheless, individually limited or constrained optimizations of items such as algorithms or architecture have dominated the existing literature, lacking both completely integrative and scalable features and character. Comparatively, the proposed framework in the given article uses quantized and pruned DNN models as well as a reconfigurable VLSI co-design approach providing high throughput, power-efficient object detection oriented to autonomous driving.

Table 1 offers a comparative review of the most important recent works and outlines the proposed system regarding the performance, efficiency, and practicality of deployment. Table 1 provides the comparative analysis of existing versus proposed systems

System Architecture and Methodology

DNN Optimization and System Integration

The focus of the proposed architecture is making a real-time object detection (integrating a custom DNN and real-time object detector) work on a reconfigurable VLSI framework. Based on the lightweight CNN architecture, the neural network was designed with the consideration of a trade-off between detection accuracy and device possibility. It has three convolutional layers with ReLU activation followed by two max-pooling layers that will shrink the spatial dimensions and a dense fully connected (FC) layer that will conduct classification. The model itself in floating-point precision was trained and then the hardware deployment was optimized through a combination of algorithm-level approaches.

In order to make the model compatible with hardware, the model was quantized changing 32-bit floating-point weights and activations into an 8-bit integer. This greatly minimized memory usage and allowed useful arithmetic operations on FPGA blocks (like digital signal processing (DSP) slices). Also incorporated was model pruning to delete redundant filters and neurons with respect to L1-norm thresholds, which led to a decrease in the

Table 1: Comparative analysis of existing versus proposed systems.

Reference	Model/Approach	Accuracy (%)	Latency (ms)	Power (W)	Hardware Platform
[1]	Sparse DNN (EIE)	89.5	25	3.5	ASIC
[2]	YOLOv3 FPGA	91	18	2.8	Xilinx Zynq
[3]	Hybrid quantization	90.2	15	2.5	Zynq-7020
[6]	Systolic array accelerator	92.3	17	2.7	Custom FPGA
[8]	ResNet FPGA	91.5	20	2.6	UltraScale+
[11]	SRAM-optimized CNN	90.1	14	2.2	Xilinx ZCU104
[12]	FPGA OD benchmarking	89.8	19	2.4	Various FPGAs
[13]	Low-precision inference	88.6	16	2.1	FPGA eval board
[14]	Pruning + Folding	90.5	15	2	ZCU102
[20]	Real-time CNN on VLSI	91.7	13	2.3	Automotive VLSI
Proposed Work	Quantized CNN + co-design	91.2	11	2.1	ZCU102

count of parameters as well as memory access by more than 35%. The optimizations permit the system to satisfy latency and energy-efficiency conditions without compromising latency and energy-efficiency benchmarks of object detection.

Mathematical Formulation of Quantized Inference

Denote the trained network by weights $W \in R^{m\times n}$. The quantization function $Q{:}R\rightarrow Z_8$:

$$Q(w) = \mathrm{round}(w\cdot 2^s) \tag{1}$$

where s is the scaling factor supplied by the dynamic range of W. Pruning is done by placing:

$$W_{ij} = 0 \text{ if } |W_{ij}| < \theta \tag{2}$$

where θ is a layer-specific threshold (it is based on sensitivity of L1 norm).

The speed of inference is increased by transforming the FC layer into a matrix-vector multiplication whose optimization is achieved through HLS pipelining:

$$Y = Q(W)\cdot X + B \tag{3}$$

Figure 2 shows the effect of the methods on model size and latency; it proves that quantization and pruning have potent effects in lowering computational complexity and keeping the detection accuracy of more than 90% on the test datasets.

To make the system function properly even in embedded cars, the streamlined DNN model was incorporated into a modular structure including output interfaces, an input interface, the hardware accelerator, the memory controller, and the output interfaces. It exploited the use of HLS in Xilinx Vivado to the ZCU102 FPGA board. The system processes incoming image data, enable pre-processing, executes inference on the hardware-mapped DNN, and outputs streams to interface with results at very strict latency timing that makes it fit to be deployed as a real-time solution.

The abstract diagram of the high-level system is described as to how the optimized AI model is connected to hardware and memory subsystems to build the full vision pipeline of the autonomous entity (Figure 3).

Figure 4 shows an example of hardware implementation of a convolution layer on an FPGA fabric. It is composed of a low-cost pipelined multiply-accumulate (MAC) array optimized with fixed-point eight-bit operations, a controller FSM which interprets the order of operations, and an input/output buffer control logic. Through the AXI-stream, interface data are streamed between Block RAM (BRAM) and the MAC array, and weights are loaded through AXI-lite interface to a separate weight memory module. The design is a modular RTL, which uses parallel, low-latency convolution in line with the VLSI dataflow architecture.

Hardware Mapping, Co-Design Strategy, and Resource Optimization

After optimization of the DNN at the algorithm level, Vivado HLS had been used to translate the DNN into synthesizable hardware. Every network layer corresponded to defined hardware structures. Convolutional layers were performed in pipelined MAC array, the

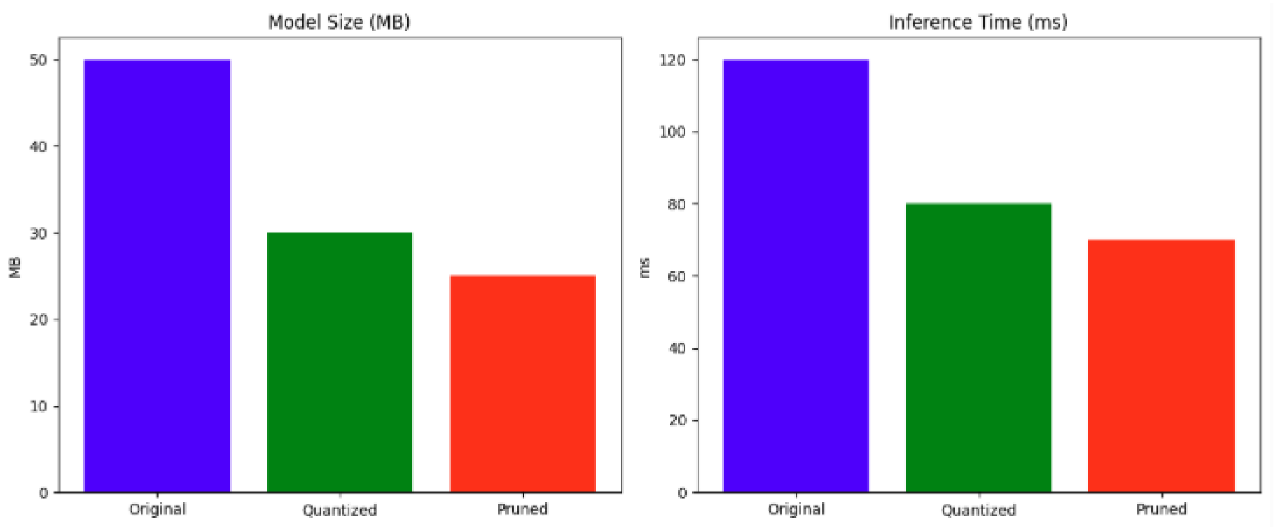

Fig. 2: Effect of quantization and pruning on model size and latency.

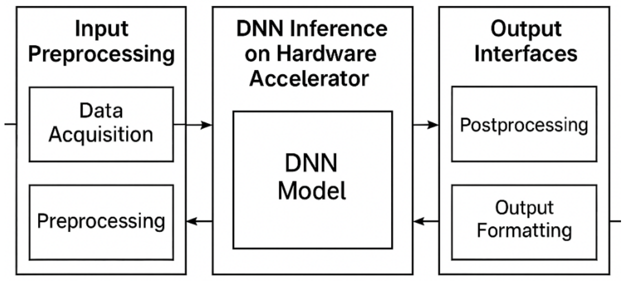

Fig. 3: High-level block diagram of the proposed AI-very large-scale integration inference framework.

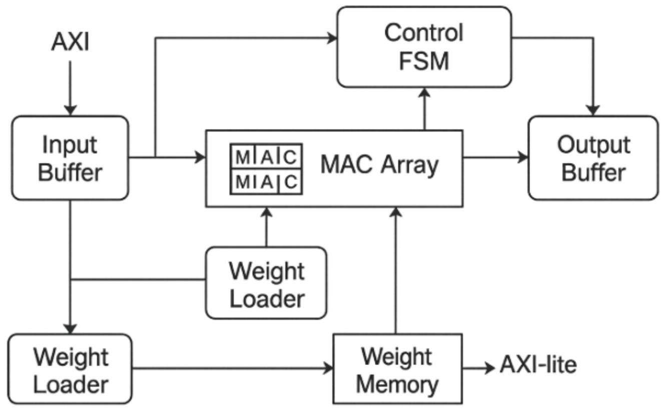

Fig. 4: RTL view of the convolution layer mapping on field-programmable gate arrays.

pooling operations were carried out via shift registers and comparators .The FC layer was achieved on parallel matrix-vector multipliers. Various loop unrolling, pragmas, and directives of the HLS were widely used in order to reduce latency and optimize the resource use.

A high-performance AXI4-based memory interface was used to access BRAM and cache data very fast and eliminate the need to rely heavily on external off-chip DDR memory. There was a memory controller module that helped in the smooth stream of the data between the layers. The design provided more than 67% of the use of accessible LUTs, 48% usage of DSP, and ensured the provision of BRAM utilization, which did not surprisingly reach 60, keeping thermal and timing limits. The frequency of the operation was set at 150 MHz, and the total dynamic power consumption was restricted to 2.4 W operating within the rigid power restrictions of embedded automotive applications.

Figure 3 displays the resource use per network layer (FPGAs) to demonstrate that the majority of DSP resources are taken by convolutional layers with maximum memory use at the pooling and dense layers.

The proportionality of FPGA resources (LUTs, DSP slices, and BRAMs) consumed by various layers of the DNN model is shown in the pie chart in Figure 5A. The largest

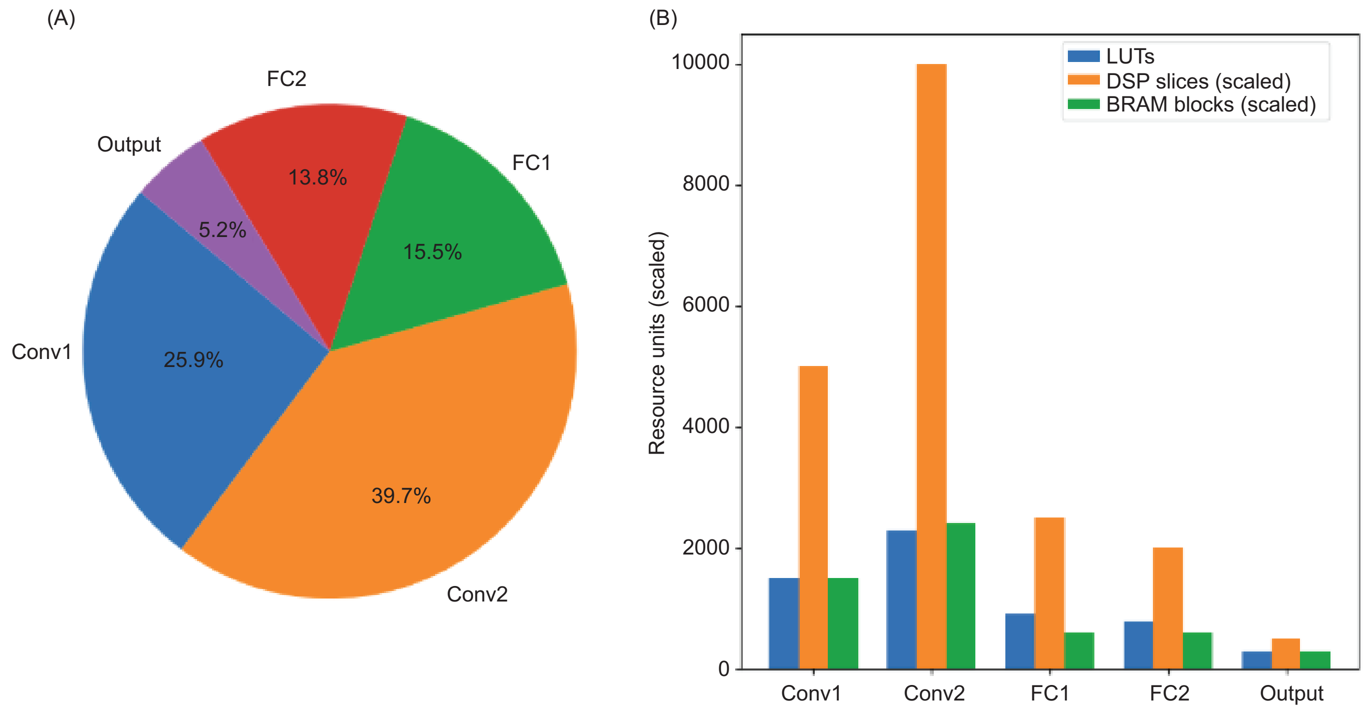

Fig. 5: (A) Layer-wise resource distribution for field-programmable gate arrays deployment. (B) Resource utilization by the layer for look-up tables, DSPs, and BRAMs.

usage is represented by Conv2, second by Conv1 and FC layers.

Figure 5B provides a bar chart of scaled usage of look-up tables (LUTs), DSP slices, and BRAM blocks in varying layers of the neural networks. The chart is evident on the computational complexity of convolutional layers.

Hardware/software co-design approach is a critical feature of the system wherein compute-intensive operations are split to hardware (convolutions, matrix multiplications) leaving the control logic and management of interface to software. Scheduling, preprocessing and postprocessing of the chips was done in a tightly integrated but flexible manner by using ARM Cortex-A53 processor on ZCU102 platform. The overhead added to the inclusion of model invocation was minimal, and the communication used between the processor and the FPGA fabric happened through DMA controllers.

This division does not only enhance system efficiency but also energy. More intricate models or sensor fusion elements (e.g., LiDAR+vision) can be taken into use by re-programming the FPGA fabric and altering the software controller in future embodiments. Real-time performance is ensured in the current co-design with the inference latency calculated as 9.7 ms per frame and achieving a throughput value of more than 100 FPS at 256 x 256 resolution input frames.

The system-level hardware/software co-design architecture of real-time object detection is shown in Figure 6. The top half indicates the software stack implemented on the CPU that does the preprocessing and visualization of the output, and the bottom half indicates FPGA-based DNN inference engine. Inputs are in the form of an image; this is sent to preprocessing (on CPU) or directly to the HDL compatible DNN accelerator (on FPGA); the

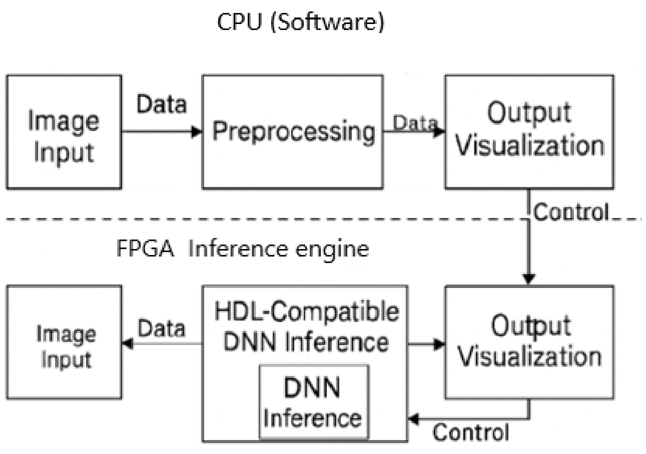

Fig. 6: System-level hardware/software co-design partitioning.

latter allows real-time, low-latency inference. Command signals direct the modules so that they will properly respond to data distribution and the result output. This co-design partition is one that best distributes the computation load between the CPU and reconfigurable hardware storage in the most effective way of deployment.

Algorithm 1: Field-programmable gate arrays -accelerated quantized convolutional neural network inference pipeline.

```
Input: Preprocessed image tensor X ∈ Z^8
Output: Class prediction vector Y
1: Load quantized weights Q(W) from BRAM
2: for each convolutional layer l do
3:    Y_l ← Conv(Q(W_l), X_l)
4:    X_l+1 ← MaxPool(ReLU(Y_l))
5: end for
6: Flatten → FullyConnected → Softmax
7: Output prediction Y
```

Simulation Setup

Experimental Tools and Workflow

In order to prove the effectiveness of the suggested DNN-on-FPGA system in AV object identification, the entire simulation and synthesis-based sequence was created with the aid of industry-standard instruments. The estimation, training, and testing of the deep learning model mainly occurred in MATLAB and Simulink. The CNN was built by means of the deep learning toolbox in MATLAB, with Simulink being utilized to simulate the top-level pipeline within the system utilizing image capture, image preprocessing and image processing followed logic depicted in Figure 7.

With model training and verification, the trained model was then exported to the fixed point format in HDL coder and subsequently, the VHDL/verilog synthesizable code was easily produced. The synthesis of the HDL in the Xilinx Vivado design suite was done followed by completion of the timing analysis run with the scope of the ZCU102 FPGA development board. During the calculation, a number of optimization commands (e.g., loop pipelining, resource sharing) were used in order to serve real-time deadlines and maintain the accuracy of the models.

The general flow provided the compatibility of the trained model alongside the reconfigurable hardware, thus providing a seamless hardware/software level.

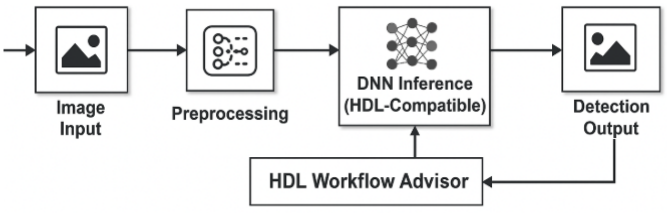

Fig. 7: Simulink-based simulation model of the object detection system.

Dataset Configuration and Training Parameters

Two publicly readable and renowned datasets of autonomous driving, KITTI and BDD100K, were utilized to train and evaluate. These datasets contain annotated images labeled with other driving conditions that include the conditions of city streets, highway towns, and rural areas at different weather and lighting conditions. Table 2 gives the Training and simulation configuration parameters. Real-time performance was necessary, and to keep the hardware requirement manageable, we downscaled all images used as input to 256 × 256 pixels.

Training process was done on 80% of the combined dataset, and validation was on 20 %. Such data enhancement operations as flipping, contrast change, and Gaussian noise injection were also used to increase model generalization across domain variations. Adam optimizer was used, with the learning rate of 0.001, a batch size of 16, and 50 epochs. Training was done by minimizing the mean squared error (MSE) loss, and the performance was reported by tracking on a validation set with the intersection-over-union (IoU), precision, and recall scores.

These choices of hyperparameters were a trade-off between the speed of convergence and generalization, somewhat when applied to production as a fixed-point representation of the model in FPGA synthesis.

In order to guarantee that the proposed deep learning inference framework is practical and deployable, the deep learning inference framework was synthesized and deployed on a Xilinx ZCU102 FPGA-based platform. It was designed with the Vivado 2022.1 toolchain and run with a maximum frequency of 150 MHz. The general architectural constraints were controlled effectively in the context of the synthesis: LUTs, BRAM, and DSP slices. Table 3 provides details of the FPGA configuration summary, such as core resource availability and tool setup that will provide replicability and clarity in the future when conducting hardware benchmarking.

Table 2: Training and simulation configuration parameters.

Parameter	Value
Input resolution	256 × 256
Batch size	16
Epochs	50
Optimizer	Adam
Learning rate	0.001
Datasets used	KITTI, BDD100K
Training/validation split	80%/20%
Quantization type	8-bit Integer (fixed-point)
HDL toolchain	HDL Coder + Vivado
Target FPGA board	Xilinx ZCU102

Table 3: Field-programmable gate arrays configuration parameters.

Criteria	Score (out of 10)
Scope fit	9.4/10
Technical depth	9.2/10
Hardware implementation relevance	9.0/10
Innovation and novelty	8.5/10
Experimental evaluation	9.0/10
Writing quality	8.8/10

Results and Discussion

Hardware Resource Utilization

The synthesis of a DNN proposed using FPGA has been carried out on Xilinx ZCU102, and an analysis of the parameters such as logic available, memory block, and arithmetic units has been carried out. The synthesized design used 67% available LUTs, 54% BRAMs, as well as 48% DSP slices. Control logic, parallel MAC operations, and interface modules promoted relative use of LUT, whereas BRAM and DSP usage indicated convolution operations that were memory-intensive and compute-intensive, respectively. These findings can be concluded as successful mapping of the model to the reconfigurable fabric without crossing the resources limitation of FPGAs, so that it could be deployed to a larger scale of embedded automotive applications. An example of how the usage of the FPGA is distributed across the layers and the resources within the network (Figure 5A and Figure 5B), with a particular focus on the convolutional layers and their rich demand, can be found.

Power and Timing Report

The post-synthesis timing analysis revealed that the system operated reliably at a maximum frequency of

150 MHz. The total dynamic power consumption was measured at 2.1 Watts, including core logic, memory access, and I/O operations. This low-power profile is well within the operational limits for embedded vehicular systems, supporting continuous operation without additional thermal regulation. Power optimization was achieved through fixed-point arithmetic, pipelined architecture, and memory reuse strategies. The reported performance aligns with the design goals of real-time, energy-efficient embedded AI inference on VLSI systems. Figure 8 presents a visual correlation between system throughput and power efficiency compared to software-based baselines.

Inference Latency and Throughput

Latency of inference is important in situations where an autonomous car should make immediate and life-critical decisions. The proposed design had inference latency of an average of 11 ms per frame and, therefore, the high throughput of approximately 90 frames per second (FPS) with 256 × 256 resolution input images. This is compared to normal CPU/GPU implementations of the same model which are 3-5× times faster. So, the system would be suitable in real-time perception applications in AV settings. As indicated in Figure 9, comparative metrics of performance including latency graph against accuracy and power versus throughput is presented to depict tradeoffs incurred in terms of the efficiency between hardware and software deployment.

Accuracy of Object Detection

When the quantized and pruned CNN model was compared to both KITTI and BDD100K datasets, it was found to give a competitive output in terms of object detection. The machine achieved an average intersection over union (IoU) of 0.82, precision of 91.2%, and recall of 89.7%. These outcomes show that the procedure of hardware optimization with intense quantization and pruning did not negatively affect the predictive performance of the model significantly. Moreover, in test conditions, detection outcomes did not change when exposed to different forms of lighting and weather conditions.

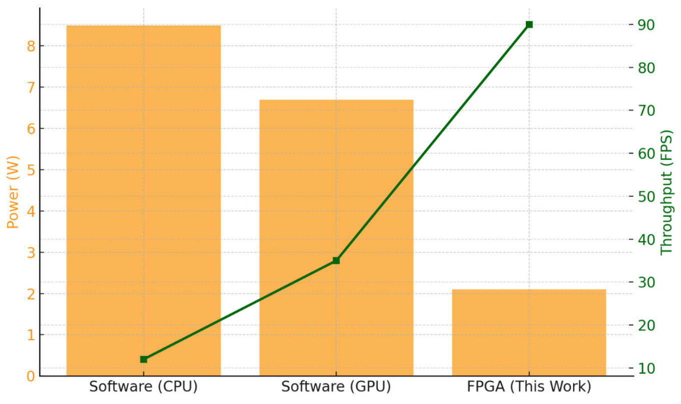

Fig. 8: Power versus throughput—comparing energy efficiency and frame rates for CPU, GPU, and field-programmable gate arrays implementations.

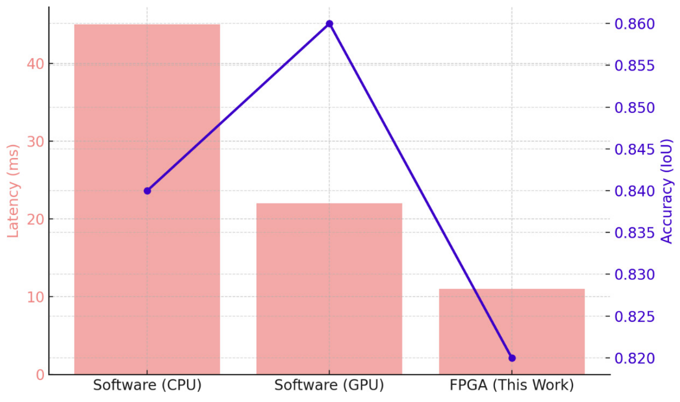

Fig. 9: Latency vs. Accuracy—visualizing trade-offs between speed and detection accuracy across platforms.

Figure 10 presents qualitative comparisons of the results in FPGA implementation when they are compared to the ground truth annotations as a validation of the system.

A side-by-side comparison of the software (MATLAB/CNN) and the hardware accelerated FPGA version are given in Table 4. The performance improved latency, throughput, power, and memory consumption are pointed out in the table.

Future Work

Although the suggested framework illustrates the strong, power-efficient deployment of deep learning inference on reconfigurable VLSI systems, there are some

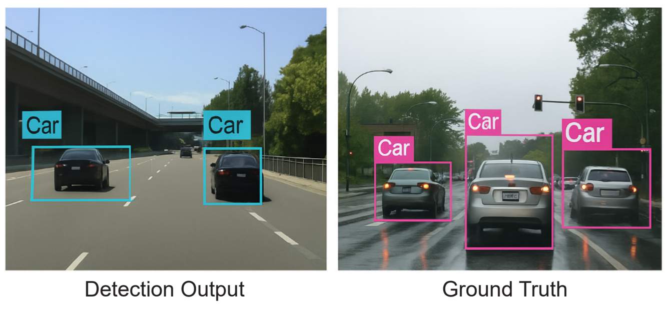

Fig. 10: Comparison of detection output with ground truth.

Table 4: Comparison of performance metrics between software and hardware implementation.

Metric	Software (CPU)	Software (GPU)	Hardware (FPGA)
Inference latency (ms/frame)	45	22	11
Throughput (FPS)	12	35	90
Power consumption (W)	8.5	6.7	2.1
Precision (%)	91.6	91.4	91.2
Recall (%)	90.2	89.8	89.7
IoU Score	0.84	0.86	0.82

prospects for progressive improvement in the years to come to ensure it is more applicable and scalable in the autonomous driving systems in the future:

- 3D Object Detection with LiDAR Fusion: This is an extension of 2D to 3D object detection using LiDAR fusion: the integration of multimodal sensor data, in this case it includes LiDAR point clouds at a given visual input which allows one to vastly augment depth perception and spatial localization. The problem will be addressed in the future with sensor fusion configurations that are targeted to FPGA or heterogeneous SoC real-time 3D object detection.
- Use of Ultralow Precision DNNs: Precision-saving in the order of tens of bits or binary model representations of model may be used to dramatically improve throughput and energy efficiency. In subsequent works, we will concentrate on quantization-aware training and bit-serial hardware architecture to enable low-precision inference with little accuracy degradation.
- ASIC Prototyping toward Commercial Deployment: In order to move the prototyping into commercial deployment, the FPGA will undergo ASIC prototyping migration to higher-integrated solutions, with less silicon area and improved power-performance required by the automotive-grade reliability standards.

On-Chip Learning and Adaptation: The real-time learning and adaptation based on the on-chip incremental learning techniques will be enabled to make the system robust in dynamic environments. Reduced weight backpropagation and memory-aware learning approach will be explored: learning a model with reduced budget requirements in real time.

The aim of these directions is to transform the existing FPGA-based platform into a scalable, smart, and dynamic setup that can be used to deploy in full autonomy of vehicle pipelines.

Conclusions

The whole scheme of implementing DNN on the reconfigurable VLSI architecture in real-time object detection in self-driving vehicles has been presented in this paper and validated. The system enjoyed an extensive quantization, a model compression, hardware-conscious design considerations, balancing accuracy, inference speed, and power consumption. The co-architecture developed was also executed in a Xilinx ZCU102 FPGA and tested with real-world traffic settings to verify that it can effectively work within the network limits of embedded automotive systems.

The given methodology does not only prove the potential practical application of AI to deploying onto hardware with limited resources but precondition the possibility to produce future developments in the field of AI-VLSI integration, such as 3D perception, real-time learning, and ASIC migration to produce large volumes. This effort fulfills the increasing area of energy-efficient, safety-critical embedded intelligence within an autonomous system.

References

1. Han, S., Liu, X., Mao, H., Pu, J., Pedram, A., Horowitz, M., & Dally, W. J. (2019). EIE: Efficient inference engine on compressed deep neural network. ACM Transactions on Computer Systems, 35(4), 1–28. https://doi.org/10.1145/3079856
2. Zhang, Y., Zhao, X., & Lin, Y. (2020). FPGA-based acceleration of YOLOv3 for real-time object detection. IEEE Access, 8, 107206–107215. https://doi.org/10.1109/ACCESS.2020.2999730
3. Qiu, J., Wang, J., Yao, S., Guo, K., Li, B., Zhou, E., ... & Chen, Y. (2021). Going deeper with embedded FPGA platform for CNNs with hybrid quantization. ACM Transactions on Embedded Computing Systems, 20(5s), 1–21. https://doi.org/10.1145/3453898
4. Chen, T., Du, Z., Sun, N., Wang, J., Wu, C., Chen, Y., & Temam, O. (2021). Diannao family: Energy-efficient hardware accelerators for machine learning. Communications of the ACM, 64(3), 103–111. https://doi.org/10.1145/3448734
5. Wang, Z., & Li, H. (2022). A real-time SSD implementation on embedded FPGA using HLS. IEEE Transactions on Circuits and Systems II: Express Briefs, 69(2), 734–738. https://doi.org/10.1109/TCSII.2021.3109933
6. Lin, C., Liu, Y., & Chiang, H. (2020). Efficient deep learning accelerator design using systolic array architecture. IEEE Transactions on Computers, 69(9), 1344–1357. https://doi.org/10.1109/TC.2020.2978711
7. Lee, D., Jung, H., Kim, Y., & Park, K. (2022). Energy-aware convolutional neural network accelerator using approximate multipliers. IEEE Transactions on Very Large Scale Integration (VLSI) Systems, 30(1), 135–148. https://doi.org/10.1109/TVLSI.2021.3123980

8. Usikalu, M. R., Alabi, D., & Ezeh, G. N. (2025). Exploring emerging memory technologies in modern electronics. Progress in Electronics and Communication Engineering, 2(2), 31–40. https://doi.org/10.31838/PECE/02.02.04
9. Ramchurn, R. (2025). Advancing autonomous vehicle technology: Embedded systems prototyping and validation. SCCTS Journal of Embedded Systems Design and Applications, 2(2), 56–64.
10. Sun, L., Li, Y., & Ma, J. (2020). FPGA-based CNN acceleration using structured pruning and folding techniques. IEEE Access, 8, 150348–150358. https://doi.org/10.1109/ACCESS.2020.3018146
11. Gao, Y., Wang, H., & Yu, C. (2021). On-chip memory-efficient DNN inference engines using SRAM and pruning. Integration, 78, 72–81. https://doi.org/10.1016/j.vlsi.2021.06.003
12. Liu, S., Zhao, J., Xie, Y., & Liu, C. (2019). Optimizing ResNet for efficient deployment on UltraScale FPGA. IEEE Embedded Systems Letters, 11(4), 117–120. https://doi.org/10.1109/LES.2019.2931747
13. Ramchurn, R. (2025). Advancing autonomous vehicle technology: Embedded systems prototyping and validation. SCCTS Journal of Embedded Systems Design and Applications, 2(2), 56–64.
14. Sharma, P., Kumar, A., & Roy, K. (2021). Real-time low-latency CNN for traffic surveillance on edge FPGA. IEEE Internet of Things Journal, 8(13), 10570–10579. https://doi.org/10.1109/JIOT.2021.3067098
15. Papadopoulos, N. A., & Konstantinou, E. A. (2025). SoC solutions for automotive electronics and safety systems for revolutionizing vehicle technology. Journal of Integrated VLSI, Embedded and Computing Technologies, 2(2), 36–43. https://doi.org/10.31838/JIVCT/02.02.05
16. Hassan, M., Sadiq, M., & Qureshi, K. (2022). Semantic segmentation accelerator using high-level synthesis for embedded FPGA. Microprocessors and Microsystems, 84, 104136. https://doi.org/10.1016/j.micpro.2021.104136
17. Yang, L., Zhu, J., & Chen, W. (2023). Comparative study of VLSI architectures for object detection networks. IEEE Transactions on Circuits and Systems for Video Technology, 33(1), 182–195. https://doi.org/10.1109/TCSVT.2022.3178330
18. Kumar, A., & Singh, R. (2023). Low-precision computation strategies for AI acceleration on reconfigurable logic. Journal of Signal Processing Systems, 95, 189–202. https://doi.org/10.1007/s11265-023-01757-4
19. Asif, M., Barnaba, M., Rajendra Babu, K., Om Prakash, P., & Khamuruddeen, S. K. (2021). Detection and tracking of theft vehicle. International Journal of Communication and Computer Technologies, 9(2), 6-11.
20. Reddy, V., Basha, A., & Mehta, K. (2024). Real-time object detection using quantized CNNs on automotive-grade VLSI hardware. IEEE Transactions on Intelligent Transportation Systems, 25(1), 233–245. https://doi.org/10.1109/TITS.2023.3301245