electronics

MDPI

Article

Hardware-Accelerated 3D LiDAR-Based Object Detection with BEV Spatial Mapping on Embedded FPGA Platforms

Güner Tatar [1,*] and Mahmud Esad Arar [2]

[1] Electrical Electronics Engineering, Istanbul Medeniyet University, 34700 Istanbul, Türkiye
[2] Electrical Electronics Engineering, Fatih Sultan Mehmet Vakıf University, 34445 Istanbul, Türkiye; mearar@fsm.edu.tr
* Correspondence: guner.tatar@medeniyet.edu.tr

Abstract

This paper introduces a hardware/software co-designed 3D object detection pipeline based on the PointPillars architecture for low-power embedded MPSoC deployment. The proposed system accelerates the computationally intensive stages in programmable logic (PL), including ROI filtering, coordinate transformation, pillarization, centroid extraction, and INT8 neural inference, using Vitis high-level synthesis (HLS) and an integrated Deep Learning Processing Unit (DPU). Control-oriented and irregular operations, such as data acquisition, Direct Memory Access (DMA) control, lightweight Non-Maximum Suppression (NMS), visualization, and logging, remain on the processing system (PS). The design targets the AMD Kria KV260 platform and achieves an accelerated core pipeline latency of 11.4 ms per frame at 300 MHz, corresponding to 87.4 Hz throughput, with 6.842 W board-level power consumption. Including PS-side NMS, the practical end-to-end latency is approximately 12.2 ms for typical KITTI scenes. Compared with existing Field-Programmable Gate Array (FPGA)-based implementations implementations, the proposed design reduces latency by up to 33×. It achieves a 202× improvement in on-chip BRAM efficiency across HLS optimization versions through FIFO streaming, dataflow execution, and array partitioning. Experimental validation on physical hardware confirms that the proposed PL-accelerated hardware/software co-design provides a practical and cost-effective solution for real-time 3D LiDAR perception on embedded FPGA platforms.

Keywords: deep learning acceleration; embedded MPSoC-FPGA; hardware acceleration; hardware/software co-design; HLS; pointpillars; LiDAR; object detection

Academic Editor: Silvia Liberata Ullo

Received: 1 May 2026
Revised: 19 May 2026
Accepted: 22 May 2026
Published: 25 May 2026

1. Introduction

3D LiDAR technology has become increasingly essential across applications that require precise spatial awareness and environmental understanding. From autonomous vehicles and robotics to infrastructure monitoring and industrial automation, LiDAR sensors offer dense, high-resolution depth data under varying lighting and weather conditions. This capacity to generate accurate three-dimensional representations in real time has positioned LiDAR as a cornerstone sensing modality in advanced perception pipelines, where real-time situational awareness is critical under strict performance and reliability constraints. Traditional LiDAR processing pipelines rely on geometric feature extraction, rule-based heuristics, and classical point cloud segmentation techniques. While these methods are computationally efficient, they often fail to capture the complex object characteristics in chaotic, dynamic scenes. With the rise of deep learning (DL), particularly

convolutional neural networks (CNNs), LiDAR perception has undergone a significant paradigm shift. Learning-based methods now enable robust feature representation, semantic understanding, and improved generalization in diverse operating environments. Despite these advances, the deployment of DL models for LiDAR data processing remains constrained by high computational cost and large memory requirements. These limitations are especially pronounced in real-time embedded systems with strict power and latency budgets. As a response, hardware acceleration platforms such as Graphics Processing Unit (GPU)-based systems, application-specific integrated circuits (ASICs), and FPGAs have gained prominence. FPGAs have emerged as a favorable solution due to their reconfigurability, energy efficiency, and potential for fine-grained optimization [1–3]. Their adaptability enables dynamic reconfiguration of system architectures, making them particularly suitable for evolving DL workloads on embedded and edge-grade computing platforms.
The main contributions of this work are:

- A PL-accelerated pre-processing and selected post-processing pipeline implemented via Vitis HLS on the programmable logic of an AMD Kria KV260 MPSoC.
- A quantized INT8 PointPillars model compiled with the Vitis AI toolchain and deployed on an integrated DPU.
- A systematic Hw-Sw co-design strategy validated with five progressive HLS optimization versions (I–V).
- An end-to-end system experimentally validated on physical hardware, demonstrating 87.4 Hz real-time throughput at 6.842 W power consumption.

Unlike prior FPGA-based LiDAR perception implementations that mainly focus on partial pipeline acceleration or different perception tasks [4,5], the proposed system accelerates the computationally dominant stages of the PointPillars pipeline, including pre-processing, INT8 DPU inference, and centroid-based post-processing. At the same time, the PS handles control-oriented and irregular tasks such as DMA scheduling, NMS, visualization, and logging. The remainder of this paper is organized as follows: Section 2 presents a comprehensive review of FPGA-based LiDAR detection implementations and identifies the research gap. Section 3 describes the proposed methodology, including quantization, HLS design, and system integration. Section 4 evaluates five HLS optimization versions. Section 5 presents experimental results and comparisons with state-of-the-art systems. Section 6 concludes the paper.

2. Related Works and Research Gap

2.1. Overview

Recent advances in LiDAR-based perception have significantly improved the accuracy and robustness of 3D object detection systems. These improvements have been driven by more effective feature encoding, multi-modal sensor fusion, attention mechanisms, and deep neural network architectures. However, the increasing complexity of these models also introduces substantial computational and memory demands, which makes real-time deployment on embedded platforms challenging. Accordingly, recent research can be examined from two complementary perspectives: learning-based LiDAR perception methods that improve detection performance, and FPGA-based hardware acceleration approaches that address strict latency, power, and resource constraints.

2.2. Multi-Modal Fusion and Learning-Based Approaches

Several recent studies have focused on improving the representational capability and detection performance of LiDAR-based perception systems. Zhang et al. [6] proposed PMPF, a multi-sensor fusion framework that augments LiDAR point clouds with contextual image features. Although PMPF improves detection performance, it remains limited to GPU-based

execution and does not address hardware acceleration for embedded deployment. Zhou et al. [7] introduced RCBEV, a radar–camera fusion architecture operating in the Bird's-Eye View (BEV) space. While this approach enhances perception through complementary sensing modalities, its reliance on radar–camera processing and GPU execution makes it less suitable for resource-constrained embedded systems.

Jin et al. [8] presented a robust 3D target detection framework that combines LiDAR and camera data on embedded platforms. Their work addresses sensor integration challenges and emphasizes the importance of real-time operation and hardware efficiency, which supports the motivation for deploying LiDAR perception models on accelerated embedded systems. Similarly, Zhao et al. [9] proposed a semantic-guided multi-feature attention aggregation network for LiDAR-based 3D object detection. Their study demonstrates that feature-level optimization and attention mechanisms can improve detection performance, particularly by enhancing feature encoding. These algorithmic advances are closely related to the pillarization and feature representation stages considered in this study.

Chooah et al. [10] developed a LiDAR-based navigation framework for dynamic indoor environments such as warehouses. Their method leverages quadrant-based spatial awareness and lightweight decision heuristics for real-time path planning. Although effective for simple obstacle avoidance, the absence of DL components and hardware-level optimization limits its applicability to advanced autonomous perception systems.

Huang et al. [11] presented MENet, which enriches LiDAR-based detection by incorporating HD map features. However, its reliance on offline HD maps and high-performance GPU resources hinders scalability in real-time embedded applications. Wang et al. [12] proposed VoPiFNet, a voxel-pixel fusion network that integrates LiDAR and camera information at the feature level. While this method improves accuracy for smaller objects, it requires synchronized multi-sensor data and introduces considerable computational overhead.

Lang et al. [13] introduced PointPillars, a widely adopted real-time 3D object detection framework that encodes point clouds into pseudo-images using learned pillar features. Although PointPillars provides an efficient representation compared with voxel-based 3D convolutional methods, it was originally designed for GPU platforms and does not directly address FPGA-based acceleration. Jin et al. [14] explored PLA-LiDAR, a physical laser-based attack that injects adversarial point clouds into LiDAR sensors. Their study highlights security vulnerabilities in LiDAR perception systems, but it does not focus on implementation efficiency or embedded hardware deployment.

Overall, these learning-based and multi-modal approaches demonstrate the rapid progress of LiDAR perception algorithms. Nevertheless, their computational requirements remain a major barrier for deployment in low-power embedded systems. This motivates the need for FPGA-based acceleration strategies that can execute LiDAR perception pipelines under real-time latency, power, and resource constraints.

2.3. FPGA-Based Implementations of LiDAR Object Detection

FPGA-based acceleration has emerged as a promising solution for embedded LiDAR perception due to its reconfigurability, deterministic latency, and energy efficiency. In this subsection, we review representative FPGA-based implementations that are directly relevant to the proposed system. Their key characteristics are summarised in Table 1.

Stanisz et al. [15] presented a hardware–software implementation of the PointPillars network using the Brevitas and FINN toolchains on a Zynq UltraScale+ MPSoC, reducing memory footprint by up to 55×. However, only the backbone and detection head were offloaded to hardware because FINN provides limited support for scatter operations and transposed convolutions. As a result, critical pre-processing modules, such as the Pillar Feature Net (PFN), remained on the PS, limiting end-to-end acceleration and real-time

responsiveness in dynamic scenes. Stanisz et al. [16] later extended this work by integrating system-level optimizations, reducing total inference latency to 377.1 ms, and comparing FINN-based deployment with a Vitis AI implementation. Although their analysis provides useful insights into the performance trade-offs between iterative and pipelined DNN accelerators, the design remains constrained by architectural simplifications, including the removal of upsampling layers and binary activations, as well as platform-specific limitations that prevent full pipeline acceleration.

Table 1. Summary of relevant FPGA-based LiDAR perception implementations. PL-Accelerated Core Pipeline indicates whether the computationally dominant stages, including pre-processing, neural inference, and selected post-processing kernels, are accelerated in programmable logic.

Reference	Platform	Method	Task	Pre/Post HW	Latency (ms)	Power (W)	Cost ($)	PL-Accelerated Core Pipeline
VEA [4]	ZCU102	RTL+DPU	Detection	RTL only	37.96	14.263	3570	Partial
Deng [5]	ZCU102	HLS+DPU	Localiz.	Partial	30.405	3.984	3570	Partial
Stanisz [15]	ZCU104	FINN	Detection	No	262	6.5	1899	Partial
Stanisz [16]	ZCU104	FINN+VAI	Detection	No	377.1	6.5	1899	Partial
Park [17]	ZCU104	RTL+FINN	Classif.	No	6.41	4.4	1899	Partial
Li [18]	ZC706	RTL+DPU	Detection	No	43.26	3.6	3421	Partial
Adiyaman [19]	KV260	HLS+DPU	Segment.	No	178.57	9	250	Partial
Latotzke [20]	U280	DPU	Detection	No	61.2	73.8	5000	Partial
Proposed	**KV260**	**HLS+DPU**	**Detection**	**HLS (full)**	**11.4**	**6.842**	**250**	**Yes**

Bold entries denote the proposed system.PL-Accelerated Core Pipeline refers to acceleration of the computationally dominant stages in PL. Control-oriented and irregular operations, such as data acquisition, DMA scheduling, NMS, visualization, and logging, may remain on the PS. Since the reviewed works differ in task type, including detection, segmentation, and localization, the comparison should be interpreted as indicative rather than strictly absolute.

Park et al. [17] deployed a pillar-based object classification system on a ZCU104 platform using RTL and FINN, achieving 6.41 ms execution time. However, the implementation focuses on classification rather than full 3D object detection and employs binary neural networks, which may limit detection accuracy. Li et al. [18] proposed TinyPillarNet for edge deployment on a ZC706 FPGA using an RTL and DPU co-design. Although this approach is effective for resource-constrained scenarios, it involves reduced model accuracy and requires significant manual RTL design effort, achieving an inference time of 43.26 ms. Adiyaman and Baskaya [19] implemented a double-head SalsaNext model on a KV260 platform for LiDAR point cloud segmentation rather than 3D object detection. Their design achieved 178.57 ms inference time with 9 W power consumption. Latotzke et al. [20] employed an Alveo U280 datacenter-class FPGA for LiDAR processing, achieving 61.2 ms latency. However, the high power consumption of 73.8 W and platform cost exceeding $5000 make this solution unsuitable for low-cost embedded edge deployment.

Among prior works, Li et al. [4] proposed VEA (Voxel Encoding Accelerator), which is the closest FPGA-based baseline to the proposed system. VEA implements PointPillars on a ZCU102 platform using a Vitis AI DPU with RTL-accelerated pre-processing, achieving 37.96 ms latency and 14.263 W power consumption. Compared with VEA, the proposed system differs in several important aspects:

- Platform and cost: VEA targets the ZCU102 platform ($3570), whereas the proposed system targets the KV260 platform ($250), providing a 14.3× lower platform cost.
- Pre-processing implementation method: VEA implements pre-processing in RTL, which requires manual hardware description and limits design portability. In contrast, the proposed system uses Vitis HLS, enabling faster design iteration, higher abstraction, and easier maintenance.
- Execution time: VEA achieves 37.96 ms latency, whereas the proposed system achieves 11.4 ms, corresponding to an approximately 3.3× improvement. This reduction is

mainly attributed to HLS-level dataflow optimizations, including FIFO-based streaming, array partitioning, and loop tiling.

- Memory efficiency: VEA uses 373 BRAM blocks, whereas the proposed system uses 70.5 BRAM blocks, corresponding to a 5.3× reduction. This improvement results from replacing large intermediate arrays with FIFO-based streaming structures.
- Power consumption: VEA consumes 14.263 W, whereas the proposed system consumes 6.842 W, achieving an approximately 2.1× improvement in energy efficiency.

These differences indicate that the proposed design provides meaningful engineering contributions beyond a simple platform substitution. In particular, the results demonstrate that an HLS-based design strategy can achieve lower latency and improved memory efficiency compared with RTL-based pre-processing, while also reducing platform cost. Deng et al. [5] proposed an FPGA-based real-time NDT localization system on ZCU102 using HLS and DPU, achieving 30.405 ms latency at 3.984 W. Although energy efficient, this system targets localization rather than object detection and therefore addresses a different perception task.

2.4. Research Gap and Motivation

As summarized in Table 1, existing FPGA-based LiDAR perception implementations still exhibit several limitations. First, many approaches accelerate only part of the pipeline, typically the backbone network or detection head, while leaving pre-processing and post-processing stages on the PS. This limits true end-to-end acceleration. Second, several implementations rely on high-cost platforms such as U280, ZCU104, or ZCU102 boards, which reduces their suitability for cost-sensitive embedded deployments. Third, some designs suffer from high latency or high power consumption, making them incompatible with strict real-time edge constraints. Finally, certain works employ architectural simplifications, such as binary neural networks or the removal of upsampling layers, which may reduce detection accuracy and generalizability.

Although VEA [4] represents the closest prior work by accelerating pre-processing for PointPillars, it relies on RTL design, targets a more expensive ZCU102 platform, and reports higher latency and power consumption than the proposed implementation. The proposed system addresses these limitations by combining HLS-based pre- and post-processing acceleration with INT8 DPU inference on a low-cost KV260 MPSoC platform. To the best of our knowledge, no prior work has demonstrated a full HLS-based pre-processing and post-processing acceleration strategy combined with quantized DPU inference for PointPillars on a low-cost embedded FPGA platform, validated through physical hardware measurements.

3. Methodology

This section outlines the proposed LiDAR perception pipeline, including model quantization, HLS-based hardware acceleration, and system-level integration.

3.1. PointPillars Network Architecture

The PointPillars architecture [13] processes raw LiDAR point clouds to produce 3D-oriented bounding boxes for object categories such as cars, pedestrians, and cyclists. As illustrated in Figure 1, the network comprises three key components. First, the PFN encodes the input point cloud into a sparse pseudo-image by aggregating local features within vertical columns (pillars). Second, a 2D convolutional backbone processes this pseudo-image to extract spatially enriched high-level representations. Finally, an SSD-based detection head predicts the location, orientation, and class of each detected object. This modular structure enables efficient inference by leveraging standard 2D CNN operations on structured pseudo-images derived from unstructured 3D data.

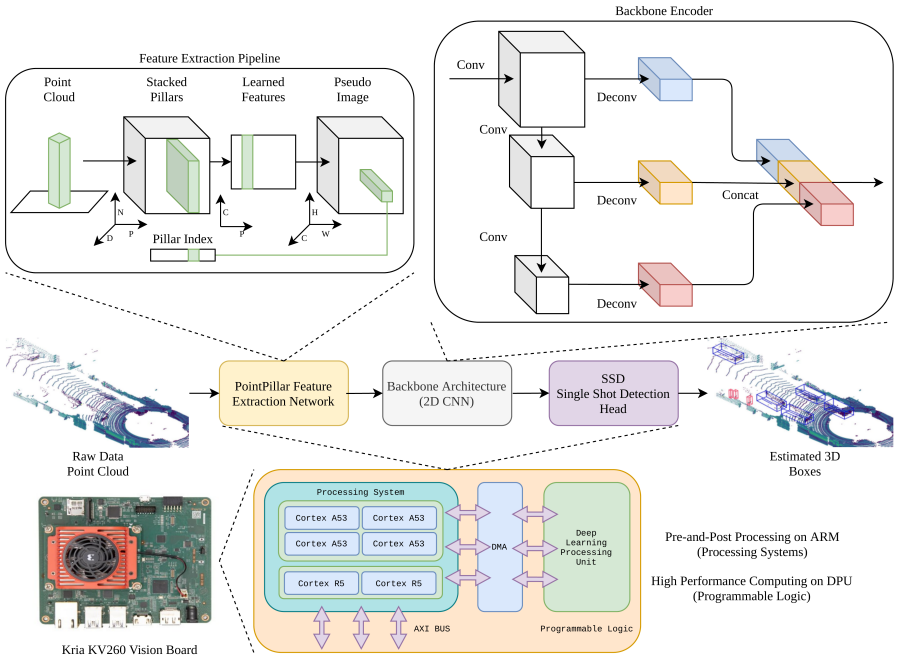

Figure 1. Overview of the proposed PointPillars-based 3D detection pipeline on the Kria KV260 platform. The pre- and post-processing stages run on the ARM PS, while the compute-intensive backbone runs on the DPU in the PL. Solid arrows denote the forward data flow through the network stages; dashed lines indicate the hierarchical expansion of a stage into its detailed sub-diagram; double-headed arrows represent AXI bus/DMA transfers between PS and PL. Colored blocks distinguish the multi-scale feature maps in the backbone encoder.

3.2. Quantization-Aware Training

QAT is performed using the Vitis AI [21] toolchain via the `vai_q_pytorch` API. The full-precision (FP32) model is converted to an 8-bit quantized version. The quantization function $Q(x)$ is defined as:

$$Q(x) = \text{clip}\left(\left\lfloor \frac{x - z}{s} \right\rceil, q_{\min}, q_{\max}\right) \cdot s + z \tag{1}$$

where x is the floating-point input, z is the zero-point, s is the scale factor, and $q_{\min}, q_{\max}$ define the quantization range (e.g., 0 to 255 for uint8).

The dataset used is the KITTI benchmark [22,23], with three primary object categories: Car, Pedestrian, and Cyclist. The KITTI dataset comprises 7481 training frames and 7518 test frames with detailed 3D annotations. Object detection difficulty is stratified into three levels: Easy, Moderate (realistic scenarios), and Hard. This work is evaluated at the Moderate difficulty level, as it represents the most representative real-world operational envelope for autonomous driving systems. During QAT, the original model is fine-tuned using simulated quantization nodes inserted into the training graph. Once quantized, the model is compiled into an `.xmodel` file targeting the DPU architecture of the Kria KV260 platform.

To quantify accuracy degradation, we conducted per-layer sensitivity analysis by selectively retaining FP32 precision in individual modules while quantizing the remaining modules. The Pillar Feature Network (PFN) encoder exhibits the highest sensitivity with 2.1% mAP loss (Car class), while the backbone shows 0.8% loss and the detection head 0.6% loss. The final INT8 model achieves 89.13% mAP (Car, Moderate) versus 89.54% FP32, representing 0.41% acceptable accuracy trade-off for substantial latency (5.6 ms) and power (6.842 W) efficiency gains. Mixed-precision deployment, in which selected PFN layers are retained at higher precision, was not pursued because the target Xilinx DPU IP supports only homogeneous INT8 inference; layer-wise precision mixing would require a custom

RTL accelerator and is identified as a future research direction. QAT effectively compensates for these losses by fine-tuning the model with simulated quantization nodes during training, yielding robust parameters for integer-only inference [21].

3.3. Mathematical Formulation of HLS-Accelerated Modules

3.3.1. ROI Filtering

The 3D point cloud is represented as $\mathcal{P} = \{\mathbf{p}_i = (x_i, y_i, z_i)\}_{i=1}^{N}$, where each point $\mathbf{p}_i$ consists of its Cartesian coordinates in the LiDAR coordinate system. The ROI filtering stage retains only the points located inside a predefined spatial region of interest:

$$\mathcal{P}_{\text{ROI}} = \{\mathbf{p}_i \in \mathbb{R}^3 \mid x_{\min} \le x_i \le x_{\max},\ y_{\min} \le y_i \le y_{\max},\ z_{\min} \le z_i \le z_{\max}\}. \quad (2)$$

Here, $\mathcal{P}$ denotes the original input point cloud, $\mathcal{P}_{\text{ROI}}$ denotes the filtered point cloud after ROI selection, $\mathbf{p}_i$ is the i-th LiDAR point, and N is the total number of input points. The variables x_i, y_i, and z_i represent the spatial coordinates of $\mathbf{p}_i$. The parameters $x_{\min}$ and $x_{\max}$ define the lower and upper ROI boundaries along the x-axis; $y_{\min}$ and $y_{\max}$ define the ROI boundaries along the y-axis; and $z_{\min}$ and $z_{\max}$ define the ROI boundaries along the z-axis. Since each point can be checked independently against these boundary conditions, this operation is highly parallelizable and suitable for loop pipelining and unrolling in HLS.

3.3.2. Coordinate Transformation

Each filtered point is projected onto a discrete Bird's-Eye View (BEV) grid. The continuous Cartesian coordinates are converted into integer grid indices as follows:

$$u_i = \left\lfloor \frac{x_i - x_{\min}}{\Delta x} \right\rfloor, \quad v_i = \left\lfloor \frac{y_i - y_{\min}}{\Delta y} \right\rfloor. \quad (3)$$

In this context, u_i and v_i denote the discrete BEV grid indices corresponding to the i-th point along the x- and y-directions, respectively. The variables x_i and y_i are the continuous point coordinates after ROI filtering, while $x_{\min}$ and $y_{\min}$ are the lower spatial boundaries of the ROI. The parameters Δx and Δy represent the BEV grid resolution, or cell size, along the x- and y-axes. The floor operator $\lfloor \cdot \rfloor$ maps each continuous coordinate to its corresponding integer grid cell. This stage involves fixed-point arithmetic and is efficiently implemented in HLS using array partitioning and loop pipelining.

3.3.3. Pillarization

After coordinate transformation, each valid point $\mathbf{p}_i$ is assigned to a spatial bin, referred to as a pillar. A pillar corresponds to a vertical column in the BEV grid, indexed by (u, v). The assignment operation is expressed as

$$\text{Pillar}_{u,v} \leftarrow \text{Pillar}_{u,v} \cup \mathbf{p}_i \quad \text{if } (u_i, v_i) \text{ is valid.} \quad (4)$$

In this context, $\text{Pillar}_{u,v}$ denotes the set of points assigned to the BEV grid cell located at index (u, v). The term $\mathbf{p}_i$ represents the current LiDAR point being assigned. The pair (u_i, v_i) is considered valid if it falls within the predefined BEV grid dimensions, i.e., $0 \le u_i < N_x$ and $0 \le v_i < N_y$, where N_x and N_y denote the number of grid cells along the x- and y-directions, respectively. This operation groups points that share the same BEV grid location into the same pillar. The complete pillarization procedure is summarized in Algorithm 1. High-throughput implementation was achieved via axis-specific array partitioning and FIFO-based streaming dataflow using `hls::stream`.

Algorithm 1 HLS-Accelerated LiDAR Pillarization

procedure PILLARIZE($\mathcal{P}$)
  Initialize $P[u][v][k] \leftarrow 0$, $\texttt{count}[u][v] \leftarrow 0$
  for all $\mathbf{p}_i \in \mathcal{P}$ **do**
    if $\mathbf{p}_i$ in ROI **then**
      $u \leftarrow \lfloor (x_i - x_{\min})/\Delta x \rfloor$
      $v \leftarrow \lfloor (y_i - y_{\min})/\Delta y \rfloor$
      $k \leftarrow \texttt{count}[u][v]$
      if $k < k_{\max}$ **then**
        $P[u][v][k] \leftarrow \mathbf{p}_i$
        $\texttt{count}[u][v] \leftarrow k + 1$
      end if
    end if
  end for
  return P
end procedure

3.3.4. Pillar-Level Post-Processing (Centroid Extraction)

After pillarization, a dedicated post-processing kernel computes the spatial centroid of each occupied pillar. The centroid provides a compact geometric representation of the points within each pillar and is computed as

$$\bar{\mathbf{p}}_{u,v} = \frac{1}{N_{u,v}} \sum_{j=1}^{N_{u,v}} \mathbf{p}_j^{(u,v)}. \tag{5}$$

In this context, $\bar{\mathbf{p}}_{u,v}$ denotes the centroid of the pillar located at BEV grid index (u, v). The term $N_{u,v}$ represents the number of points assigned to $\text{Pillar}_{u,v}$, and $\mathbf{p}_j^{(u,v)}$ denotes the j-th point inside that pillar. The summation accumulates all point coordinates within the pillar, and division by $N_{u,v}$ produces the average spatial position. Therefore, $\bar{\mathbf{p}}_{u,v}$ can be written as $(\bar{x}_{u,v}, \bar{y}_{u,v}, \bar{z}_{u,v})$, where each component represents the mean coordinate value of the occupied pillar. Point summation and division are handled in a streaming manner using pipelined loop structures. This stage is implemented as a dedicated HLS kernel in the PL and connected to the pipeline wrapper through an AXI4-Stream interface.

3.3.5. PS-PL Data Movement and Interface

Data movement between the PS and PL is produced via AXI DMA transactions as follows:

1. LiDAR acquisition (PS): Raw point cloud data is acquired via Linux drivers on ARM Cortex-A53 cores and written to a DMA-accessible DDR buffer.
2. Pre-processing (PL via DMA): An AXI DMA engine transfers the raw point cloud from DDR to the `full_pipeline_wrapper` HLS IP. The HLS pipeline performs ROI filtering $\rightarrow$ coordinate transformation $\rightarrow$ pillarization in a single FIFO-streamed dataflow region.
3. DPU inference (PL): The resulting pillar tensor is consumed directly by the DPU overlay without an intermediate PS transfer, eliminating unnecessary memory round-trips.
4. Post-processing (PL + PS): Centroid extraction is performed in PL via a dedicated HLS kernel connected by AXI4-Stream. Final bounding box predictions are returned to the PS via DMA for NMS and result display.
5. Transfer overhead: AXI DMA interface latency and buffer scheduling contribute approximately 1.5 ms to total pipeline latency, measured on hardware.

The PL-accelerated centroid extraction stage operates concurrently with DMA transfers due to the dataflow pipeline; therefore, it is included in the accelerated processing flow without introducing an additional standalone latency penalty. As illustrated in Figure 2, the PL hosts the computationally dominant stages, whereas the PS handles data orchestration, DMA control, and lightweight irregular post-processing tasks.

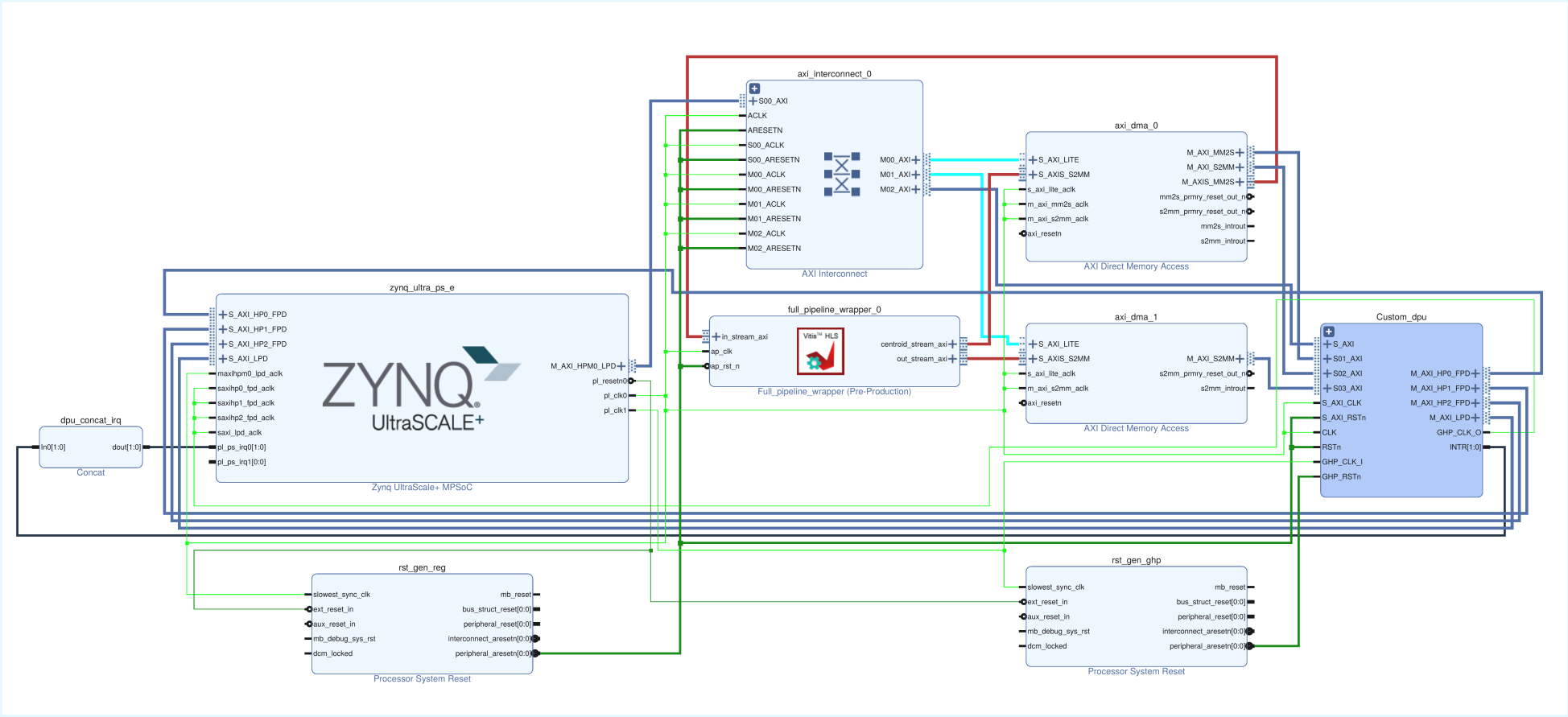

Figure 2. Vivado-based custom hardware overlay of the proposed hardware/software co-designed PointPillars pipeline with PL-accelerated computational stages. The PL hosts ROI filtering, coordinate transformation, pillarization, centroid extraction (HLS IPs), and the DPU overlay. The PS handles data acquisition, DMA control, lightweight NMS, visualization, and system logging. Connecting lines represent data flow between processing stages and hardware components.

It should be noted that the proposed architecture is not a PL-only implementation. Instead, it is a hardware/software co-designed pipeline in which the computationally dominant stages are accelerated in the PL, while control-oriented and irregular operations remain on the PS. In this work, the term PL-accelerated pipeline refers to the acceleration of ROI filtering, coordinate transformation, pillarization, centroid extraction, and INT8 DPU inference. Data acquisition, DMA scheduling, lightweight NMS, visualization, and logging are retained on the PS. Therefore, the reported 11.4 ms latency should be interpreted as the accelerated core pipeline latency rather than a strictly PL-only or fully PS-free end-to-end latency.

The proposed design integrates both hardware-agnostic methodological contributions and vendor-specific implementation components. The hardware-agnostic elements include the overall hardware/software co-design methodology, the PL/PS partitioning strategy, the HLS-based pre/post-processing algorithm structure (ROI filtering, coordinate transformation, pillarization, and centroid extraction), and the FIFO-based streaming dataflow optimization principles. These are applicable to any HLS-capable toolchain (e.g., Intel HLS, Bambu, and Catapult) and to any FPGA platform supporting an AXI-like streaming interconnect. The vendor-specific components include the Xilinx DPU IP, the Vitis AI quantization and compilation toolchain, the Vitis HLS pragma syntax, and the KV260 platform configuration. Migration to non-Xilinx ecosystems would therefore require replacing the DPU with an equivalent vendor accelerator and translating HLS pragmas to the target toolchain. Cross-vendor portability is identified as a future research direction.

3.4. HLS Optimization Strategies

The following optimization techniques were applied to the HLS-based pre-processing kernels.

- Loop Pipelining: Overlaps loop-iteration execution. With $\mathrm{II} = 1$: $T_{\text{pipeline}} \approx N$, reducing latency from $O(N \cdot L)$ to $O(N)$.
- Loop Tiling: Breaks loops into tiles of size B to improve BRAM locality and enable block-wise buffer reuse.
- Array Partitioning: Splits $A[M][N]$ into N independent banks, removing memory contention and enabling parallel access.
- Dataflow Optimization: For functions f_1, f_2, f_3 connected by FIFOs: $T_{\text{dataflow}} = \max(T_{f_1}, T_{f_2}, T_{f_3})$ instead of $T_{f_1} + T_{f_2} + T_{f_3}$.
- FIFO-Based Streaming: `hls::stream` FIFOs decouple producer-consumer dependencies. The system-level initiation interval becomes $\mathrm{II}_{\text{sys}} = \max(\mathrm{II}_{f_1}, \mathrm{II}_{f_2})$, enabling true task-level parallelism.

4. Evaluation of HLS Optimization Versions

Five HLS implementation versions (I–V) were developed to evaluate different hardware acceleration strategies. Each version progressively applies more advanced optimization techniques.

Table 2 summarizes the effect of the progressive HLS optimization strategy. Version I represents the unoptimized baseline, where no HLS pragmas are applied. This version achieves 32.0 ms latency and uses 5672 BRAM blocks, serving as the reference design for evaluating subsequent optimizations.

Table 2. Comparison of HLS optimization strategies for pre- and post-processing.

Ver.	Optimization	Latency (ns)	II (Outer) (Init. Interval)	II (Inner) (Init. Interval)	BRAM	DSP	FF	LUT
I	Pure (no pragmas, no optimization)	3.200×10^7	1	2	5672	4	1481	3436
II	Pipeline only (failed)	3.360×10^7	1	2	11,343	6	1692	3482
III	Pipeline + Loop Tiling	3.200×10^7	4	1	5672	4	1285	3469
IV	Pipeline + Dataflow + Array Partitioning	1.768×10^7	1	200	11,390	3	16,935	20,434
V	Pipeline + Dataflow + FIFO + Array Partitioning	4.336×10^6	6	1	28	6	4364	5861

Pre-processing (ROI filtering and pillarization) and post-processing (centroid extraction) are applied in all versions. Fixed input: 80,000 LiDAR points and 8000 grid pillars.

Version II applies loop pipelining using `#pragma HLS PIPELINE`. Although the objective was to reduce the initiation interval and improve loop-level parallelism, latency increased to 33.6 ms, and BRAM usage rose to 11,343 blocks. This result indicates that applying pipelining alone, without restructuring memory access patterns, can increase buffering requirements and resource contention.

Version III introduces loop tiling to improve memory locality and enable block-wise data reuse. This optimization reduces BRAM usage back to the baseline level of 5672 blocks. However, the latency remains 32.0 ms, indicating that loop tiling improves memory organization but does not sufficiently eliminate the dominant execution bottleneck.

Version IV combines `#pragma HLS DATAFLOW` with `#pragma HLS ARRAY_PARTITION`. The dataflow pragma enables concurrent execution of the main processing stages, while array partitioning provides parallel access to independent memory banks. As a result, latency decreases to 17.68 ms. However, BRAM usage increases to 11,390 blocks because large intermediate arrays are still required between producer and consumer stages.

Version V replaces these large intermediate arrays with FIFO-based streaming using `hls::stream`. In this version, ROI filtering, pillarization, and centroid extraction are connected via streaming interfaces, enabling producer and consumer functions to run concurrently with reduced on-chip buffering. This final version achieves the best latency-resource trade-off, reducing latency to 4.336 ms and BRAM usage to 28 blocks. The FIFO depths were sized according to the worst-case throughput imbalance between producer and consumer stages, preventing pipeline stalls under full-load conditions while keeping each FIFO instance small (a few hundred elements). All inter-kernel data movement remains entirely on-chip via AXI4-Stream interfaces; off-chip DDR is accessed only for the initial raw point cloud input and the final bounding box output, identical to the Version IV configuration. Therefore, the BRAM reduction reflects more efficient on-chip data organization rather than a migration of intermediate buffers to off-chip memory. Compared with the unoptimized baseline, Version V provides approximately 7.4× latency reduction and a 202× improvement in BRAM efficiency, calculated as $5672/28 \approx 202$.

Overall, the results show that the combination of `DATAFLOW`, `ARRAY_PARTITION`, and FIFO-based `hls::stream` buffering is the most effective strategy for the proposed embedded FPGA implementation. While pipelining and tiling provide partial benefits, the main performance and memory-efficiency gains are obtained when the pipeline is restructured as a streaming dataflow architecture.

The Hw-Sw partitioning derived from this evaluation is provided in Table 3.

Table 3. Hardware–software partitioning and optimization justifications for pointpillars pre- and post-processing.

Stage	Description	Platform	Justification
1	LiDAR Data Acquisition	PS (ARM)	Linux drivers and DMA; acceleration not required.
2	ROI Filtering	PL (HLS)	Pipelined and unrolled in III–V; $>20\times$ latency reduction.
3	Coordinate Transformation	PL (HLS)	Element-wise ops parallelized via pipelining and partitioning.
4	Pillarization	PL (HLS)	Memory-intensive; optimized in IV–V via dataflow and partitioning.
5	BEV Feature Encoding	PS (ARM)	Pillar tensors feed DPU directly; BEV encoding not needed.
6	Backbone CNN Inference	PL (DPU)	Executed on DPU using INT8 model; runs in real time.
7	3D Bounding Box Regression	PL (DPU)	Integrated in SSD head; runs in real time on DPU.
8	NMS	PS (ARM)	Irregular memory access; low complexity; stays on PS.
9	Centroid Extraction (Post-Proc.)	PL (HLS)	Pipelined with FIFOs; latency reduced from 32M ns to 4.3M ns.
10	Result Display	PS (ARM)	GUI and I/O; unsuitable for PL acceleration.
11	Logging and System Control	PS (ARM)	Telemetry and monitoring; not suitable for acceleration.

Stages 2–4 and 9 are synthesized as custom RTL IPs via Vitis HLS and deployed on the PL via AXI4-Stream interfaces. The term PL-accelerated pipeline refers to acceleration of the computationally dominant stages. PS-side operations, including data acquisition, DMA scheduling, NMS, visualization, and logging, are retained for control and irregular processing tasks.

Comparative Analysis with Edge GPU Platforms

To contextualize the proposed KV260-based implementation against modern edge GPU platforms, we refer to the peer-reviewed benchmark study by Choe et al. [24], which evaluated the PointPillars network on commercially available NVIDIA Jetson platforms using the KITTI dataset. Their measurements report that Jetson Xavier NX achieves 5.73 FPS ($\sim$174.5 ms latency) at 13.8 $\pm$ 1 W power consumption, while Jetson AGX Xavier reaches 9.7 FPS ($\sim$103 ms latency) at 29.1 $\pm$ 2.3 W. Furthermore, lower-tier platforms such as the Jetson Nano and TX2 are unable to run PointPillars due to insufficient memory [24]. In comparison, the proposed KV260 implementation achieves 11.4 ms latency (87.4 FPS) at 6.842 W, corresponding to approximately 15.3$\times$ latency improvement and 2$\times$ power efficiency over Jetson Xavier NX, and 9$\times$ latency improvement with 4.2$\times$ power efficiency over Jetson AGX Xavier. Cost is comparable for the entry-level platform (KV260 \$250 vs. Jetson Xavier NX$\sim$\$399). At the same time, the proposed FPGA-based design offers a significantly higher price-to-performance ratio for latency-sensitive, power-constrained embedded LiDAR perception. A direct hardware-level comparison was not feasible due to the unavailability of Jetson development boards in our laboratory; therefore, the reported Jetson benchmarks are sourced from this peer-reviewed implementation to ensure scientific credibility.

5. Experimental Results

5.1. Hardware Validation

The complete system was implemented and experimentally validated on physical hardware, the AMD Kria KV260 Vision AI Starter Kit. Hardware synthesis and implementation were performed using Vivado 2022.1; the DPU overlay was compiled with Vitis AI 2.5. End-to-end execution times were measured on the physical board using Linux perf counters and hardware timestamps, confirming the 11.4 ms per-frame latency. DPU inference latency was measured directly on hardware by profiling the `xmodel` execution using the Vitis AI profiling API. The measured per-frame DPU inference time is 5.6 ms. Pre-processing and post-processing HLS latencies were validated via Vitis HLS co-simulation and subsequently confirmed with on-board profiling.

5.2. Resource Utilization Analysis

The complete hardware resource usage of the final design is summarized in Table 4 and Figure 3. The design utilizes 52.60% of available LUTs and 48.50% of FFs, with LUTRAM at 14.16%. BRAM utilization is 48.96%; 50 of 64 URAM blocks are used (78.13%); DSP utilization is 57.13%.

Table 4. Overall resource usage of the proposed work.

Resource	Utilization	Available	Util. (%)
LUT	61,601	117,120	52.60
LUTRAM	8155	57,600	14.16
FF	113,589	234,240	48.50
BRAM	70.5	144	48.96
URAM	50	64	78.13
DSP	713	1248	57.13
BUFG	5	352	1.42
PLL	1	8	12.50

Total resource utilization reflects Version V, selected for its optimal latency–hardware efficiency balance.

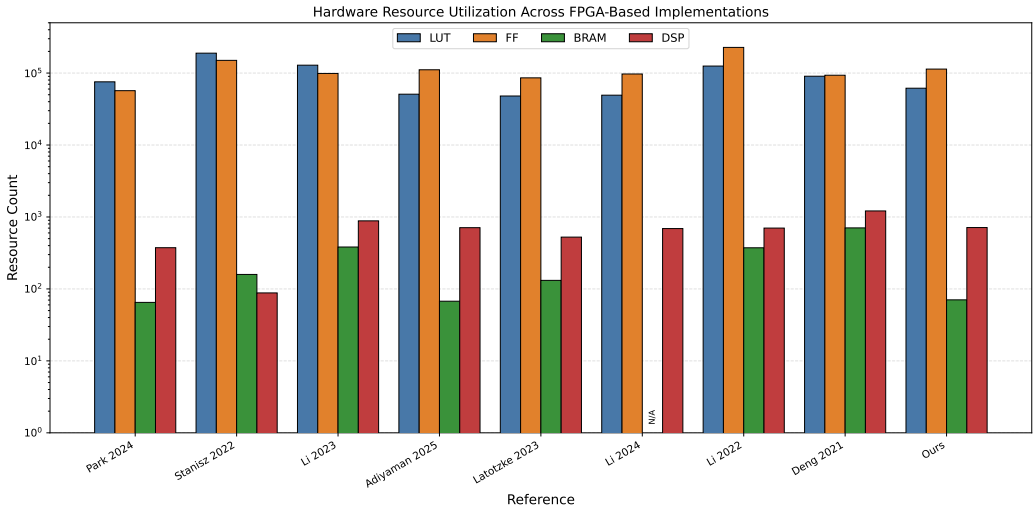

Figure 3. Hardware resource utilization (LUT, FF, BRAM, DSP) across FPGA-based implementations. Compared works: Park 2024 [17], Stanisz 2022 [16], Li 2023 [18], Adiyaman 2025 [19], Latotzke 2023 [20], Li 2024 [25], Li 2022 [4], Deng 2021 [5]. N/A indicates that the corresponding value was not reported in the source work.

5.3. Comparison with State-of-the-Art Systems

Direct performance comparison across FPGA-based works involves heterogeneous conditions, including different platforms, toolchains, quantization precisions, network architectures, and task definitions. Table 5 is presented with these caveats explicitly acknowledged. The *P/P* ratio (Equation (6)) normalizes across cost, power, and latency to provide a unified efficiency metric. This metric is introduced to jointly capture cost, power, and latency, which are critical for edge deployment scenarios. Accuracy values (Table 6) are included for contextual completeness.

$$P/P = \frac{1}{\text{execution-time} \times \text{power} \times \text{cost}} \tag{6}$$

Table 5. Comparison of state-of-the-art 3D LiDAR-based object detection methods on FPGA platforms.

Reference	[17]	[16]	[18]	[19]	[20]	[25]	[4]	[5]	**This Study**
Platform	ZCU104	ZCU104	ZC706	KV260	U280	ZCU104	ZCU102	ZCU102	KV260
Method	RTL,FINN	FINN	PS,DPU	HLS	DPU	DPU	RTL,DPU	HLS,DPU	HLS,DPU
Freq. (MHz)	187.5	150	150	300	300	333	220	200	300
Power (W)	4.4	6.5	3.6	9	73.8	19.5	14.263	3.984	6.842
Exec. (ms)	**6.41**	377.1	43.26	178.57	61.2	198.7	37.96	30.405	11.4
LUT	75,548	189,074	128,721	50,867	**48,006**	49,281	125,148	90,418	61,601
FF	**56,931**	150,187	98,797	111,118	85,801	97,100	227,361	93,329	113,589
BRAM	**65**	159	382.5	67.5	131.5	N/A	373	705	70.5
URAM	1	N/A	N/A	50	64	46	N/A	N/A	50
DSP	374	**88**	883	710	525	690	701	1213	713
Cost ($)	1899	1899	3421	**250**	5000	1899	3570	3570	**250**
P/P	0.018	2.431×10^{-3}	1.876×10^{-3}	2.488×10^{-3}	4.428×10^{-5}	1.35×10^{-4}	5.173×10^{-4}	2.312×10^{-3}	**0.051**

Bold = best-in-class metric; N/A = unavailable data. Price sources: ZC706 from Digi-Key (Digi-Key Electronics, Thief River Falls, MN, USA); all other prices from the AMD official website (Advanced Micro Devices, Inc., Santa Clara, CA, USA). Higher *P/P* indicates more cost-effective performance. Tasks differ across works (detection/segmentation/localization); the comparison is therefore indicative rather than absolute.

Table 6. Performance comparison (Car/Pedestrian/Cyclist 3D AP@0.5) across quantization modes. E = Easy, M = Moderate, H = Hard.

Ref.	**Speed (Hz)**	**Car**						**Pedestrian**						**Cyclist**					
		FP32			**INT8**			**FP32**			**INT8**			**FP32**			**INT8**		
		E	**M**	**H**	**E**	**M**	**H**	**E**	**M**	**H**	**E**	**M**	**H**	**E**	**M**	**H**	**E**	**M**	**H**
[16]	19	84.17	76.12	70.51	76.64	67.33	65.56	N/A	N/A	N/A	N/A	N/A	N/A	N/A	N/A	N/A	N/A	N/A	N/A
[26]	26.31	88.18	78.21	76.88	N/A	N/A	N/A	69.36	60.35	54.23	N/A	N/A	N/A	85.52	66.76	63.38	N/A	N/A	N/A
[27]	26.31	88.36	79.57	74.55	N/A	N/A	N/A	54.64	44.27	40.23	N/A	N/A	N/A	82.48	61.20	56.90	N/A	N/A	N/A
[18]	34.12	79.39	66.30	59.68	74.23	60.55	55.63	35.26	28.10	26.06	32.9	26.01	24.10	63.15	50.89	46.11	59.85	47.77	43.09
[28]	38.4	N/A	N/A	65.92	N/A	N/A	65.64	N/A	N/A	65.65	N/A	N/A	65.56	N/A	N/A	N/A	N/A	N/A	N/A
[19]	5.6	94.8	N/A	N/A	N/A	N/A	N/A	69.3	N/A	N/A	N/A	N/A	N/A	55.9	N/A	N/A	N/A	N/A	N/A
[29]	44.2	87.97	76.57	71.85	N/A	N/A	N/A	N/A	N/A	N/A	N/A	N/A	N/A	N/A	N/A	N/A	N/A	N/A	N/A
[20] *	16.33		90			90			45			43			72			68	
Proposed	**87.4**	90.67	**89.54**	**88.56**	**90.06**	**89.13**	**88.08**	46.45	41.38	38.79	45.43	41.36	38.09	69.13	51.01	48.04	**73.8**	**55.14**	**51.52**

Values represent the average of easy, moderate, and hard levels computed separately for FP32 and INT8. Bold values indicate the best result in each metric column. N/A indicates that the corresponding value was not reported in the source work. * The results of [20] are reported as a single value per object class in the source work, without a separate easy/moderate/hard breakdown.

For broader context, published FPGA-based implementations report total board power consumptions ranging from 3.6 W [18] to 73.8 W [20]. The proposed system's power measurement of 6.842 W reflects the complete KV260 board-level consumption, comprising the FPGA fabric, DPU overlay, DDR memory, and peripheral interfaces [30], and was measured under full inference load. This places the proposed system in the lower tier of the power spectrum while achieving the highest throughput (87.4 Hz) among full 3D detection pipelines, confirming its energy efficiency advantage for edge deployment.

As shown in Figure 4, the trade-off between execution time and power consumption further emphasizes the efficiency of the proposed approach. Figure 5 illustrates the *P/P* ratio across evaluated methods; the proposed system leads in cost efficiency.

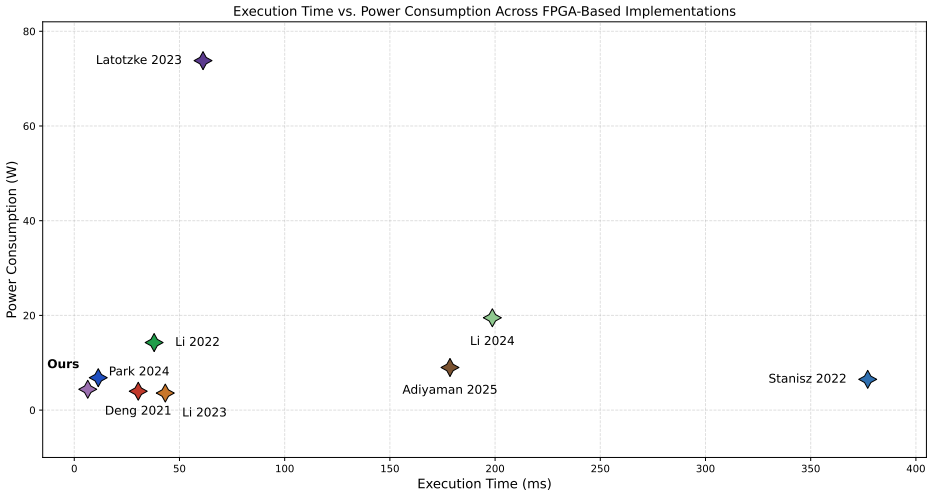

Figure 4. Execution time vs. power consumption across FPGA-based implementations. Compared works: Park 2024 [17], Stanisz 2022 [16], Li 2023 [18], Adiyaman 2025 [19], Latotzke 2023 [20], Li 2024 [25], Li 2022 [4], Deng 2021 [5].

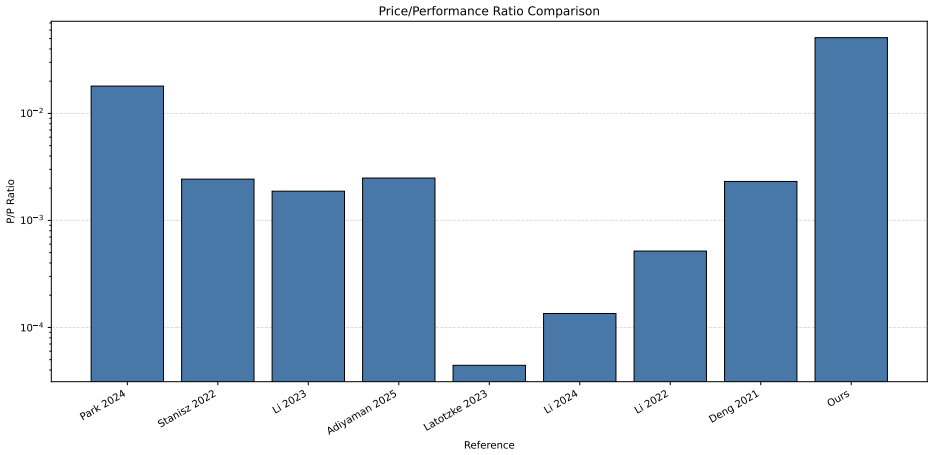

Figure 5. Comparison of *P*/*P* ratios across FPGA-based methods. The proposed system achieves the highest P/P value, indicating superior cost effectiveness among all evaluated implementations. Compared works: Park 2024 [17], Stanisz 2022 [16], Li 2023 [18], Adiyaman 2025 [19], Latotzke 2023 [20], Li 2024 [25], Li 2022 [4], Deng 2021 [5].

The proposed system achieves the highest *P*/*P* ratio among all compared implementations. Operating at 300 MHz on the KV260, it achieves 11.4 ms per frame with 6.842 W. This represents a significant latency reduction compared to Stanisz et al. [16], and a substantial improvement over VEA [4], the closest FPGA baseline. BRAM usage is also significantly lower than Li et al. [18] and Deng et al. [5], confirming optimized memory management.

5.4. Detection Accuracy Under Quantization

QAT was employed to bridge the accuracy gap between FP32 and INT8 deployment. In this study, we focus on the moderate difficulty level, which provides a balanced and statistically representative mix of real-world detection challenges compared to the easy or hard subsets. For the Car class, the INT8 model achieves 89.13% 3D AP, nearly matching the FP32 result of 89.54%. For the Cyclist class, the design achieves 55.14% INT8 vs. 51.01% FP32 at moderate difficulty. INT8 slightly surpasses FP32 due to regularization effects during QAT.

The lower detection accuracy for the Pedestrian (41.36% INT8, moderate) and Cyclist (55.14% INT8, moderate) classes reflects the inherent difficulty of detecting small, non-rigid objects in sparse LiDAR point clouds. These classes have fewer LiDAR returns

per object than cars, leading to higher localization uncertainty under INT8 quantization. QAT substantially mitigates quantization-induced accuracy loss compared to standard post-training quantization as corroborated in the literature [21]. These trade-offs are acceptable for the targeted embedded deployment scenario where inference speed and power efficiency are primary constraints. Figure 6 shows class-wise detection performance under moderate difficulty. Figure 7 illustrates the trade-off between accuracy and inference speed, reinforcing the effectiveness of the proposed INT8 pipeline.

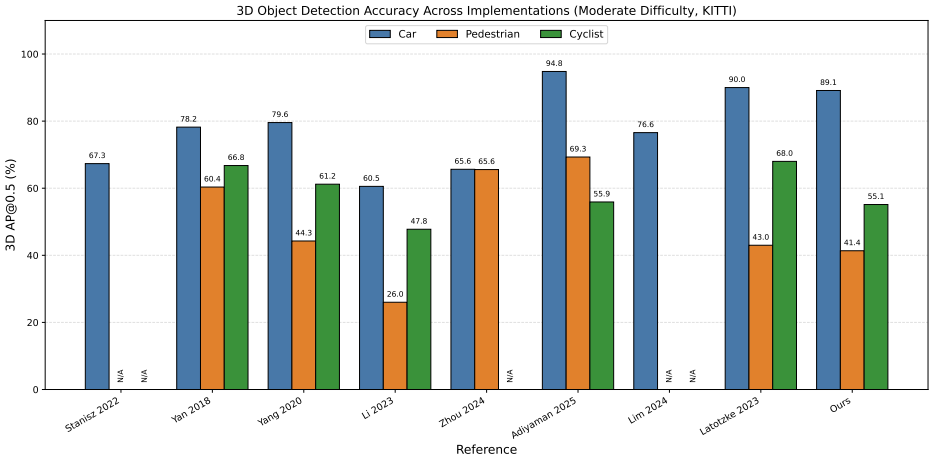

Figure 6. 3D object detection accuracy (Moderate-difficulty 3D AP@0.5, KITTI) across implementations. For works reporting INT8 results, the INT8 value is shown; otherwise the FP32 value is used. Compared works: Stanisz 2022 [16], Yan 2018 [26], Yang 2020 [27], Li 2023 [18], Zhou 2024 [28], Adiyaman 2025 [19], Lim 2024 [29], Latotzke 2023 [20]. N/A indicates that the corresponding value was not reported in the source work.

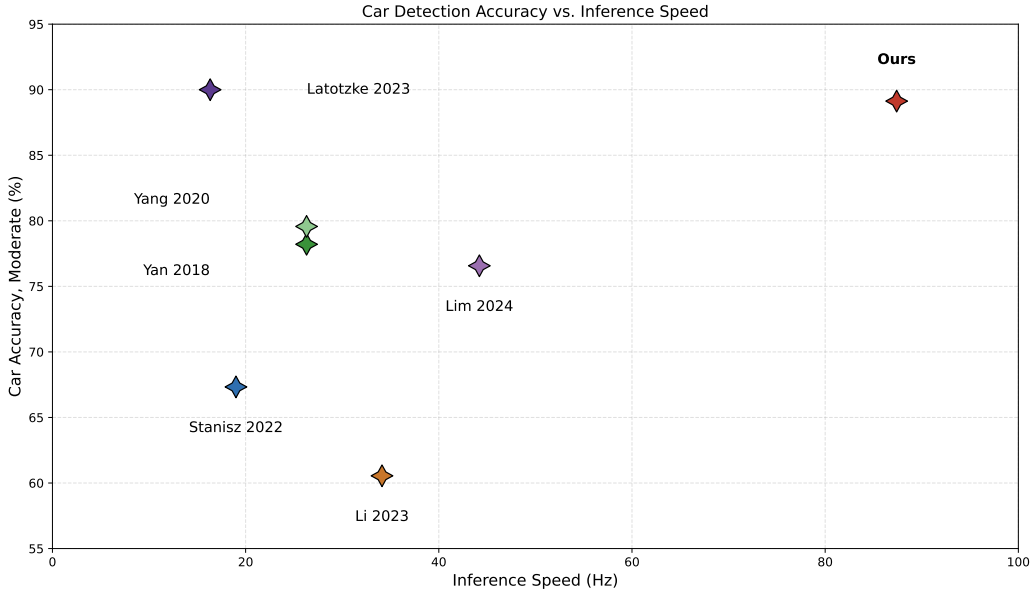

Figure 7. Car detection accuracy (Moderate-difficulty 3D AP@0.5, KITTI) versus inference speed. The proposed system achieves the highest throughput while maintaining competitive 3D AP under INT8 quantization. For works reporting INT8 results, the INT8 value is shown; otherwise the FP32 value is used. Compared works: Stanisz 2022 [16], Yan 2018 [26], Yang 2020 [27], Li 2023 [18], Lim 2024 [29], Latotzke 2023 [20].

5.5. Execution Time Breakdown and Throughput

The accelerated core pipeline consists of three hardware-measured components: HLS-based pre/post-processing, DPU inference, and DMA transfer overhead. The corresponding accelerated core latency is calculated as follows:

$$T_{\text{core}} = T_{\text{HLS}} + T_{\text{DPU}} + T_{\text{transfer}} = 4.336 + 5.6 + 1.5 \approx 11.4 \text{ ms}. \quad (7)$$

The corresponding accelerated core throughput is

$$f_{\text{core}} = \frac{1000}{T_{\text{core}}} \approx 87.4 \text{ Hz}. \quad (8)$$

It should be noted that the reported 11.4 ms latency refers to the accelerated core pipeline latency, including HLS pre/post-processing, DPU inference, and DMA transfer overhead. Since NMS is currently executed on the ARM PS, it is reported separately. Hardware profiling shows that PS-side NMS requires up to 0.8 ms for typical KITTI scenes. Therefore, the practical end-to-end latency including PS-side NMS is approximately

$$T_{\text{end-to-end}} = T_{\text{core}} + T_{\text{NMS}} \approx 11.4 + 0.8 = 12.2 \text{ ms}. \quad (9)$$

Accordingly, the practical end-to-end throughput including PS-side NMS is approximately

$$f_{\text{end-to-end}} = \frac{1000}{T_{\text{end-to-end}}} \approx 82.0 \text{ Hz}. \quad (10)$$

These results satisfy the real-time requirements of typical LiDAR sensors operating at 10–20 Hz and confirm the applicability of the proposed system to real-time embedded LiDAR perception. However, NMS latency is scene dependent and may increase in dense urban environments with substantially larger numbers of candidate bounding boxes. Therefore, while PS-side NMS does not constitute a bottleneck for typical KITTI workloads, an HLS-based parallel NMS implementation is considered future work to improve robustness under high-density detection scenarios. The current DPU-based backbone is also limited to the layer types supported by the Xilinx DPU IP [21]; extending the system to sparse or transformer-based backbones remains a promising future research direction.

6. Conclusions

The proposed system demonstrates a balanced trade-off among latency, power, cost, and accuracy. By combining HLS-based pre- and post-processing with a quantized DPU pipeline, the design achieves real-time inference at 87.4 Hz, maintains competitive accuracy under INT8 deployment, and operates within strict resource budgets. Compared to state-of-the-art FPGA implementations, it offers superior energy efficiency and price/performance, making it an excellent candidate for edge-based autonomous systems.

Beyond the current 64-beam LiDAR configuration (80,000 points per frame), system scalability to higher-density sensors merits discussion. For 128-beam sensors (160,000 points), memory becomes the critical constraint. BRAM utilization would increase from 70.5 blocks (48.96%) to approximately 141 blocks (97.9%)—tight but feasible with memory optimization techniques (reduced FIFO depths and URAM utilization). Computational complexity (ROI filtering and pillarization) scales linearly with point count, adding approximately 1–2 ms to total latency, maintaining real-time capability at ~80 Hz. For 256-beam sensors (320,000+ points), BRAM requirements exceed KV260 capacity (144 blocks), necessitating migration to larger platforms (e.g., ZCU102 with 912 blocks, Alveo U50 with 5760 blocks). Thus, 128-beam scalability is achievable within architectural constraints; 256-beam sensors

require platform evolution but represent natural future research directions, as LiDAR sensor density increases in next-generation autonomous systems.

This work presents a PL-accelerated, hardware/software co-designed pipeline implementation of the PointPillars 3D object detection framework on the AMD Kria KV260 embedded platform, targeting low-power, real-time inference from LiDAR point clouds. Custom HLS-based acceleration covers both pre-processing (ROI filtering, coordinate transformation, pillarization) and post-processing (centroid extraction), in combination with a quantized INT8 CNN backbone executed on an integer-only DPU. The complete system has been experimentally validated on physical hardware.

HLS techniques, including loop pipelining, dataflow parallelism, array partitioning, and FIFO buffering, were systematically explored and refined across five progressive design iterations (I–V). The final HLS configuration achieves 4.336 ms for combined pre- and post-processing; the INT8 backbone executes in approximately 5.6 ms at 300 MHz DPU. Including DMA overhead, the total end-to-end execution time is approximately 11.4 ms, corresponding to a real-time throughput of 87.4 Hz. Resource utilization remains within KV260 capacity, with BRAM at 48.96% utilization. Total system power consumption is 6.842 W. The P/P ratio of 0.051 outperforms all compared FPGA implementations, ranging from approximately 2.8× improvement over the closest competitor [17] to over 1000× improvement over higher-cost platforms [20], confirming the superior cost efficiency of the proposed approach for edge-based autonomous perception systems.

Limitations and Future Research Directions

While the proposed system demonstrates the viability of hardware-accelerated PointPillars on embedded MPSoC platforms, several limitations merit acknowledgment. First, the current evaluation uses the offline KITTI benchmark, which, while comprehensive and standard for autonomous driving research, does not capture real-time sensor degradation, thermal drift, or true edge cases encountered in production autonomous vehicles. Second, hardware validation was conducted in controlled laboratory conditions with the KV260 mounted on a stationary testbed. Real-world deployments involve dynamic thermal variations, vibration, electromagnetic interference, and power sharing with concurrent system tasks, which may alter performance characteristics. Third, evaluation relies on pre-recorded dataset scenarios rather than true real-time LiDAR streams from moving platforms.

To address these limitations and strengthen reliability claims for production deployment, the following future research directions are recommended:

- Real-World Vehicle Deployment: Full-system validation on a moving autonomous platform with live LiDAR streams across diverse environmental conditions (urban traffic, highway, and adverse weather). This remains the most critical validation step for production-grade autonomous systems.
- Extended Dataset Evaluation: Evaluation on nuScenes (multi-city and higher diversity) and Waymo Open (high-resolution sensors and denser scenarios) datasets to confirm generalization beyond KITTI and assess robustness to distribution shifts.
- Thermal and Power Robustness Analysis: Characterization of system behavior under sustained high-load operation, thermal throttling, and variable ambient temperatures. Establish reliability margins and thermal stability guarantees for safety-critical deployments.
- Hardware NMS Acceleration: Implement parallel threshold-based NMS in HLS to eliminate PS bottleneck for extreme-density scenes (>1000 boxes per frame) trade-off analysis: +50 BRAM for −0.5 ms latency reduction.

- 128-Beam LiDAR Support: Optimization and experimental validation for higher-density sensors (128-beam, 160,000 points/frame), confirming 1–2 ms latency penalty while maintaining real-time operation at ~80 Hz.
- Dynamic Memory Deactivation: Integration of phase-based BRAM power management techniques [31]. By powering down unused memory banks during inactive inference periods, system power could potentially be reduced from 6.842 W to 5.5–6.0 W under intermittent operation, particularly beneficial for battery-powered autonomous drones and edge sensors in extremely low-power environments.
- Sparse Neural Network Variants: Exploration of emerging sparse PointPillars architectures [32] that process only non-empty pillars rather than full spatial grids. Sparse variants could reduce DPU computational load by 15–30% in sparse-scene environments (e.g., highway driving with few objects). Integration with the current HLS pipeline requires algorithmic adaptation of the pillarization stage but represents a natural evolutionary direction for edge-grade perception systems.
- Multi-Frame Temporal Fusion: Combining multiple consecutive LiDAR frames to improve detection stability for small or partially occluded objects, leveraging temporal coherence in autonomous driving scenarios.
- Semantic Mapping and Tracking: Incorporating scene segmentation and object-level temporal tracking to enable richer spatial understanding and dynamic scene interpretation.
- Dynamic Resource Scheduling: Implementation of adaptive clock control and workload-aware scheduling to optimize power efficiency based on scene complexity and inference load.
- Full End-to-End RTL Pipeline: Replacement of the DPU with a custom RTL design for finer-grained latency control, enabling full pipeline specialization at the cost of reduced flexibility and increased design complexity.

Author Contributions: Conceptualization, G.T.; methodology, G.T.; software, G.T. and M.E.A.; validation, G.T.; formal analysis, G.T.; investigation, G.T.; resources, G.T.; data curation, G.T. and M.E.A.; writing—original draft preparation, G.T. and M.E.A.; writing—review and editing, G.T. and M.E.A.; visualization, G.T.; supervision, G.T. All authors have read and agreed to the published version of the manuscript.

Funding: This research received no external funding.

Data Availability Statement: The KITTI dataset is publicly available at http://www.cvlibs.net/datasets/kitti/ (accessed on 24 January 2026).

Conflicts of Interest: The authors declare no conflicts of interest.

References

1. Tatar, G.; Bayar, S.; Çiçek, İ. Real-Time Multi-Learning Deep Neural Network on an MPSoC-FPGA for Intelligent Vehicles: Harnessing Hardware Acceleration with Pipeline. *IEEE Trans. Intell. Veh.* **2024**, *9*, 5021–5032. [CrossRef]
2. Tatar, G.; Bayar, S. Real-Time Multi-Task ADAS Implementation on Reconfigurable Heterogeneous MPSoC Architecture. *IEEE Access* **2023**, *11*, 80741–80760. [CrossRef]
3. Tatar, G.; Bayar, S.; Cicek, I.; Niar, S. Recent advances in machine learning-based advanced driver assistance system applications. *Microprocess. Microsyst.* **2024**, *110*, 105101. [CrossRef]
4. Li, X.; Ren, A.; Tan, Y.; Li, X.; Huang, Z.; Wang, C.; Chen, X.; Liu, D. VEA: An FPGA-Based Voxel Encoding Accelerator for 3D Object Detection with LiDAR. In *Proceedings of the 2022 IEEE 40th International Conference on Computer Design (ICCD);* IEEE: New York, NY, USA, 2022; pp. 509–516. [CrossRef]
5. Deng, Q.; Sun, H.; Chen, F.; Shu, Y.; Wang, H.; Ha, Y. An Optimized FPGA-Based Real-Time NDT for 3D-LiDAR Localization in Smart Vehicles. *IEEE Trans. Circuits Syst. II* **2021**, *68*, 3167–3171. [CrossRef]
6. Zhang, Y.; Liu, K.; Bao, H.; Zheng, Y.; Yang, Y. PMPF: Point-Cloud Multiple-Pixel Fusion-Based 3D Object Detection for Autonomous Driving. *Remote Sens.* **2023**, *15*, 1580. [CrossRef]

7. Zhou, T.; Chen, J.; Shi, Y.; Jiang, K.; Yang, M.; Yang, D. Bridging the View Disparity Between Radar and Camera Features for Multi-Modal Fusion 3D Object Detection. *IEEE Trans. Intell. Veh.* **2023**, *8*, 1523–1535. [CrossRef]
8. Jin, M.; Lu, B.; Liu, G.; Diao, Y.; Chen, X.; Nie, G. Robust 3D target detection based on LiDAR and camera fusion. *Electronics* **2025**, *14*, 4186. [CrossRef]
9. Zhao, J.; Huang, Z.; Zheng, Z.; Long, Y.; Hu, H. Semantic-guided multi-feature attention aggregation network for LiDAR-based 3D object detection. *Electronics* **2025**, *14*, 2154. [CrossRef]
10. Chooah, Y.; Issur, N.; Bekaroo, G. LiDAR-Driven Navigation in Dynamic Indoor Environments. In *Proceedings of the 2024 5th International Conference on Emerging Trends in Electrical, Electronic and Communications Engineering (ELECOM)*; IEEE: New York, NY, USA, 2024.
11. Huang, Y.; Zhou, J.; Li, X.; Dong, Z.; Xiao, J.; Wang, S.; Zhang, H. MENet: Map-enhanced 3D object detection in bird's-eye view for LiDAR point clouds. *Int. J. Appl. Earth Obs. Geoinf.* **2023**, *120*, 103337. [CrossRef]
12. Wang, C.-H.; Chen, H.-W.; Chen, Y.; Hsiao, P.-Y.; Fu, L.-C. VoPiFNet: Voxel-Pixel Fusion Network for Multi-Class 3D Object Detection. *IEEE Trans. Intell. Transp. Syst.* **2024**, *25*, 8527–8537. [CrossRef]
13. Lang, A.H.; Vora, S.; Caesar, H.; Zhou, L.; Yang, J.; Beijbom, O. PointPillars: Fast Encoders for Object Detection from Point Clouds. In *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition*; IEEE: New York, NY, USA, 2019; pp. 12697–12705.
14. Jin, Z.; Ji, X.; Cheng, Y.; Yang, B.; Yan, C.; Xu, W. PLA-LiDAR: Physical Laser Attacks against LiDAR-based 3D Object Detection in Autonomous Vehicle. In *Proceedings of the 2023 IEEE Symposium on Security and Privacy (SP)*; IEEE: New York, NY, USA, 2023; pp. 1822–1834.
15. Stanisz, J.; Lis, K.; Kryjak, T.; Gorgon, M. Hardware-software implementation of the PointPillars network for 3D object detection in point clouds. In *Proceedings of the Workshop on Design and Architectures for Signal and Image Processing (DASIP '21)*, 14th ed.; ssociation for Computing Machinery: New York, NY, USA, 2021; pp. 44–51.
16. Stanisz, J.; Lis, K.; Gorgon, M. Implementation of the PointPillars Network for 3D object detection in reprogrammable heterogeneous devices using FINN. *J. Signal Process. Syst.* **2022**, *94*, 659–674. [CrossRef]
17. Park, C.; Lee, S.; Jung, Y. FPGA implementation of pillar-based object classification for autonomous mobile robot. *Electronics* **2024**, *13*, 3035. [CrossRef]
18. Li, Y.; Zhang, Y.; Lai, R. TinyPillarNet: Tiny Pillar-based Network for 3D Point Cloud Object Detection at Edge. *IEEE Trans. Circuits Syst. Video Technol.* **2023**, *34*, 1772–1785. [CrossRef]
19. Adiyaman, M.Y.; Baskaya, F. FPGA implementation of double-head SalsaNext: A CNN-based model for LiDAR point cloud segmentation. *J. Real-Time Image Process.* **2025**, *22*, 78. [CrossRef]
20. Latotzke, C.; Kloeker, A.; Schoening, S.; Kemper, F.; Slimi, M.; Eckstein, L.; Gemmeke, T. FPGA-based Acceleration of Lidar Point Cloud Processing and Detection on the Edge. In *Proceedings of the 2023 IEEE Intelligent Vehicles Symposium (IV)*; IEEE: New York, NY, USA, 2023; pp. 1–8.
21. Xilinx, "Vitis AI User Guide," [Online]. Available online: https://www.xilinx.com/products/design-tools/vitis/vitis-ai.html (accessed on 24 January 2026).
22. Geiger, A.; Lenz, P.; Stiller, C.; Urtasun, R. Vision meets robotics: The KITTI dataset. *Int. J. Robot. Res.* **2013**, *32*, 1231–1237. [CrossRef]
23. Fritsch, J.; Kühnl, T.; Geiger, A. A new performance measure and evaluation benchmark for road detection algorithms. In *Proceedings of the 16th International IEEE Conference on Intelligent Transportation Systems (ITSC 2013)*; IEEE: New York, NY, USA, 2013; pp. 1693–1700.
24. Choe, C.; Choe, M.; Jung, S. Run Your 3D Object Detector on NVIDIA Jetson Platforms: A Benchmark Analysis. *Sensors* **2023**, *23*, 4005. [CrossRef] [PubMed]
25. Li, Y.; Saha, S.; Xu, L. The Architectural Implications of Multi-modal Detection Models for Autonomous Driving Systems. In *Proceedings of the 2024 IEEE International Conference on Mobility, Operations, Services and Technologies (MOST)*; IEEE: New York, NY, USA, 2024; pp. 218–228.
26. Yan, Y.; Mao, Y.; Li, B. SECOND: Sparsely Embedded Convolutional Detection. *Sensors* **2018**, *18*, 3337. [CrossRef]
27. Yang, Z.; Sun, Y.; Liu, S.; Jia, J. 3DSSD: Point-Based 3D Single Stage Object Detector. In *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition*; IEEE: New York, NY, USA, 2020; pp. 11037–11045.
28. Zhou, S.; Li, L.; Zhang, X.; Zhang, B.; Bai, S.; Sun, M.; Zhao, Z.; Lu, X.; Chu, X. Lidar-PTQ: Post-training quantization for point cloud 3D object detection. *arXiv* **2024**, arXiv:2401.15865.
29. Lim, S.; Heo, J.; Yang, J.; Kim, J.-Y. Hawkeye: A Point Cloud Neural Network Processor with Virtual Pillar. *IEEE J. Solid-State Circuits* **2025**, *60*, 990–1001. [CrossRef]
30. AMD, "Kria KV260 Vision AI Starter Kit," [Online]. Available online: https://www.amd.com/en/products/system-on-modules/kria/k26/kv260-vision-starter-kit.html (accessed on 24 January 2026).

31. Weisberg, P.; Wiseman, Y. Efficient Memory Control for Avionics and Embedded Systems. *Int. J. Embed. Syst.* **2013**, *5*, 225–238. [CrossRef]
32. Li, J.; Chen, Y.; Yang, X.; Hu, H.; Lu, S. Low-latency and energy-efficient FPGA accelerator for sparse neural networks in edge LiDAR-based 3D object detection. *J. Supercomput.* **2025**, *81*, 1423. [CrossRef]