Journal of Real-Time Image Processing (2025) 22:78
https://doi.org/10.1007/s11554-025-01643-9

RESEARCH

FPGA implementation of double-head SalsaNext: a CNN-based model for LiDAR point cloud segmentation

Muhammed Yasin Adiyaman[1] · **Faik Baskaya**[1]

Received: 22 December 2024 / Accepted: 6 February 2025 / Published online: 11 March 2025
© The Author(s) 2025

Abstract

This study details the adaptation and deployment of a customized SalsaNext model for semantic segmentation of LiDAR point clouds on edge devices, benchmarked using the SemanticKITTI and Waymo Open datasets. We introduce an innovative multi-dataset training framework designed specifically for range image-based segmentation models. Central to this approach is our double-head SalsaNext model, which features two output heads to facilitate simultaneous training and inference on the Waymo and SemanticKITTI datasets. Following training, the model is streamlined by removing the head dedicated to Waymo, resulting in a compact, single-headed version optimized for SemanticKITTI. This simplified model is then quantized to employ fixed-point arithmetic, significantly enhancing computational efficiency and enabling real-time operation on the Xilinx Kria KV260 board. The quantization process markedly reduces resource consumption while preserving competitive accuracy. Our deployment on this low-power, FPGA-based platform underscores the potential of energy-efficient systems for advanced 3D semantic segmentation, with promising applications in autonomous systems and robotics. Experimental results validate the effectiveness of our training schema and the success of the optimized implementation of the double-head model on resource-constrained hardware.

Keywords LIDAR · FPGA · Realtime · Semantic segmentation

Point cloud segmentation plays a vital role in vision-based applications such as autonomous driving and robotics. Over time, this field has shifted from traditional hand-crafted techniques to deep neural networks (DNNs), which deliver significantly better performance. This progress has been driven by advances in computational power, enabling the design of more sophisticated models.

However, deploying complex models on edge devices presents significant challenges due to limited computational resources and strict constraints on power consumption and latency. To address these limitations, DNN models must be optimized for hardware efficiency. One effective approach is quantization, which reduces the precision of model weights to lower bit resolutions. This significantly decreases computational requirements, improves power efficiency, and enhances inference speed, making it well-suited for edge deployment.

In this study, we adapted, customized, and trained the lightweight CNN-based SalsaNext model using our proposed multi-head training mechanism. Subsequently, we optimized and quantized the model to 8-bit fixed-point precision before deploying it on a Xilinx Kria KV260 FPGA, leveraging the deep learning processing unit (DPU) architecture within its programmable logic (PL).

The main contributions of this work are outlined below:

- **Real-time model for outdoor robotics:** Developed a robust LIDAR segmentation model optimized for real-time performance in outdoor robotics, ensuring stable and reliable operation across diverse environments.
- **Merging two of the largest LiDAR segmentation datasets:** Proposed a domain adjustment schema to merge two of the most common and largest datasets, SemanticKITTI and Open Waymo Dataset, for autonomous driving.
- **Multiple-head CNN model based on SalsaNext:** Proposed a multi-head SalsaNext model to train multiple LiDAR segmentation datasets with different character-

✉ Muhammed Yasin Adiyaman
yasin.adiyaman@std.bogazici.edu.tr

Faik Baskaya
faik.baskaya@bogazici.edu.tr

[1] Electrical and Electronics Engineering Department, Bogazici University, Bebek, 34342 Istanbul, Turkey

istics simultaneously without degrading the main head's performance.
- **State-of-the-art results on FPGA:** Achieved superior mean IoU and accuracy for point cloud segmentation on the SemanticKITTI dataset, surpassing other FPGA-based solutions.
- **Cost-effective FPGA deployment:** Demonstrated that advanced point cloud segmentation models can be effectively deployed on an affordable FPGA platform.
- **Open-source implementation:** The implementation will be released as an open-source Python-based GitHub repository, allowing the community to adapt the approach for other CNN-based models compatible with FPGA systems.

The paper is organized as follows: Section 1 reviews related work on point cloud segmentation for edge devices, providing context and background. Section 2 provides an overview of the model architecture, details the dataset domain adjustments, explains the training process, and also elaborates on the quantization and deployment process on the Kria KV260 board. Finally, Sect. 3 presents the experimental results from deploying the proposed model on a Xilinx FPGA card, showcasing its performance and effectiveness.

1 Related work

Point cloud segmentation for edge devices faces several critical challenges: managing unstructured data as part of input data representation; balancing complexity and stability in model architectures; achieving hardware efficiency for edge device optimization; and addressing domain shift during multi-dataset training.

1.1 Input data representation

Existing segmentation methods for point clouds typically fall into four categories based on how they structure the input data:

1. Point-set-based methods [1, 2],
2. Voxel-based methods [3],
3. Projection-based methods, such as those using range images or bird's-eye-view (BEV) representations [4–7], and
4. Hybrid approaches [8, 9].

Among these, projection-based techniques-especially those employing range images [5, 6]-are particularly attractive for edge deployment because they offer a compact, computationally efficient representation and can leverage mature 2D CNN architectures. Although BEV methods are a viable option, their extensive field of view in outdoor robotics often leads to a significant increase in input size, making them less ideal. In contrast, range images from single-sweep point clouds provide a more concise alternative, which is why our work focuses on range-image-based methods.

1.2 Model architectures

The literature identifies four primary model architectures for point cloud segmentation:

1. Graph-based models [10–12],
2. Transformer-based models [1, 2, 13],
3. 3D CNNs [14–16], and
4. 2D CNNs [4–7].

Transformer-based models-such as Point Transformer [1] and Range Former [13]-use attention mechanisms to achieve high performance; however, their complexity and reliance on heterogeneous operations can hinder their deployment on edge devices. Similarly, while graph-based approaches like PointNet [17, 18] and KPConv [10] are adept at handling unstructured data, their high computational cost or limited accuracy often makes them unsuitable for edge applications. In contrast, 2D CNNs, suitable for processing projected range images, offer a straightforward and repetitive architecture that is friendly to hardware implementations while still providing satisfactory performance. Consequently, our approach utilizes a 2D CNN-based model.

1.3 Edge device optimization

Optimizing LiDAR segmentation models for edge devices necessitates careful consideration of speed and power efficiency. FPGA-based implementations have become a focal point because their PL units can serve as specialized computing cores, leading to enhanced performance and reduced energy consumption. Both traditional handcrafted algorithms and modern DNN-based methods have been successfully accelerated on FPGAs. For example, a non-AI approach [19] introduced a real-time FPGA-based ground segmentation technique for LiDAR point clouds using channel-based methods, angular data repair, and a cross-eight-way flood-fill algorithm, achieving significant improvements over CPU and GPU implementations. Later, a parallel processing strategy further refined depth ground segmentation (RDG) on solid-state LiDARs, resulting in a 30.9× speedup compared to CPU-based methods [20]. Despite their speed and power efficiency, these methods often struggle to generalize to the complexities of natural environments compared to AI-based models.

In the domain of AI-based methods, most edge implementations utilize range images processed by quantized

2D CNN models [21–24]. Two widely adopted CNN accelerators for edge applications are NVDLA [25] and DPU [26]. For example, the NVDLA-based method described in [21] introduces a lightweight CNN with just 1.9 million parameters. This approach creates a more compact model by splitting the processing of contextual and spatial features into separate pathways, which are subsequently merged using a feature fusion module. The model, after quantization to INT8 and deployment on a ZCU104 board using the NVDLA architecture, achieved 46.4% mIoU on the SemanticKITTI dataset, with performance improvements in speed (56 fps) and power efficiency (46 times better than GPUs).

On the DPU front, [22] demonstrated a hardware-software co-design implementation of the SqueezeSegV3 [6] model on a VCK190 board, achieving substantial real-time performance and energy efficiency improvements across various customizations of the model compared to GPU-based implementations. Similarly, [23] deployed a lightweight CNN model on the ZCU102 board featuring a customized ResNet34 backbone [27] and an ASPP head [28], which reached 10 fps and an efficiency of 42.5 GOP/W on 2 DPU cores. Additionally, [24] applied channel pruning to the SalsaNext model [5] before training and later quantized the model to INT8, demonstrating high inference rates (up to 97 fps on the VCK5000 board) and improved power efficiency across various Xilinx platforms.

Beyond the training schema (discussed in the next subsection), our FPGA mapping process closely follows conventional DPU-based implementations by employing similar DPU architectures. However, we introduce additional customizations designed to reduce quantization error, as detailed in the Methodology section.

1.4 Domain adaptability

A common challenge with models trained on a single dataset is their instability when faced with domain shifts across different environments. Although multi-dataset training is a potential remedy, significant domain discrepancies-such as those between the Waymo and KITTI datasets-can still adversely affect performance. Previous methods addressing domain shifts [29–32] often suffer from limited generalizability in real-world settings [29, 30] or depend heavily on supervised techniques [31, 32]. To address these issues, our work introduces a simple yet effective double-head training scheme with domain adjustment, specifically designed for range-image-based architectures across multiple LiDAR datasets. This multi-head approach also provides the flexibility to accommodate varying dataset annotations and differing classification requirements.

2 Methodology

In FPGA-based systems, selecting an appropriate model requires a careful balance between size, complexity, and stability. The primary trade-off is between the model's size-measured in terms of parameters and multiply-accumulate (MAC) operations-and its speed or power efficiency. Although larger models tend to yield higher accuracy, they also incur greater power consumption and increased latency. For efficient hardware mapping, the model must therefore remain both compact and simple.

Given our objective to perform inference using a DPU embedded in the PL-which supports only standard operations such as convolution, dense layers, batch normalization, and ReLU activations-we chose a lightweight CNN-based model, SalsaNext [5], specifically designed to meet these constraints. We further customized the model for multi-dataset training and optimized it for edge deployment.

As illustrated in Fig. 1 and detailed in Table 1, the model follows a fully convolutional design. Its encoder features three residual context blocks followed by four residual downsampling blocks that reduce the spatial dimensions by a factor of 16 and an additional residual block. The decoder comprises four convolutional blocks interleaved with bilinear upsampling layers, ending with a final convolutional layer. Notably, 78% of the model's parameters reside in the feature extractor backbone, while the remaining parameters are allocated to the head. Moreover, the model utilizes atrous convolution to achieve a shallower architecture with a larger receptive field, all without increasing the overall parameter count. In total, the model contains approximately 6.7 million parameters.

To enhance the stability and generalization of the model, we proposed a novel training scheme for multi-dataset training for range-image-based LIDAR semantic segmentation models and customized the base model by adding extra heads for each additional dataset (with different labeling conventions). The complete edge implementation pipeline is shown in Fig. 2.

There are three major datasets commonly used for LiDAR segmentation tasks: SemanticKITTI, nuScenes-lidarseg, and the Waymo Open Dataset, as summarized in Table 2. Due to its low number of channels and horizontal resolution, the nuScenes dataset was not suitable for training the model on projected range images. Consequently, we focused on the SemanticKITTI and Waymo datasets.

However, a significant domain shift exists between these datasets. The Waymo dataset provides range images with intensity values, whereas SemanticKITTI offers point cloud data with intensity values, and the distributions of these intensity values differ considerably. Additionally,

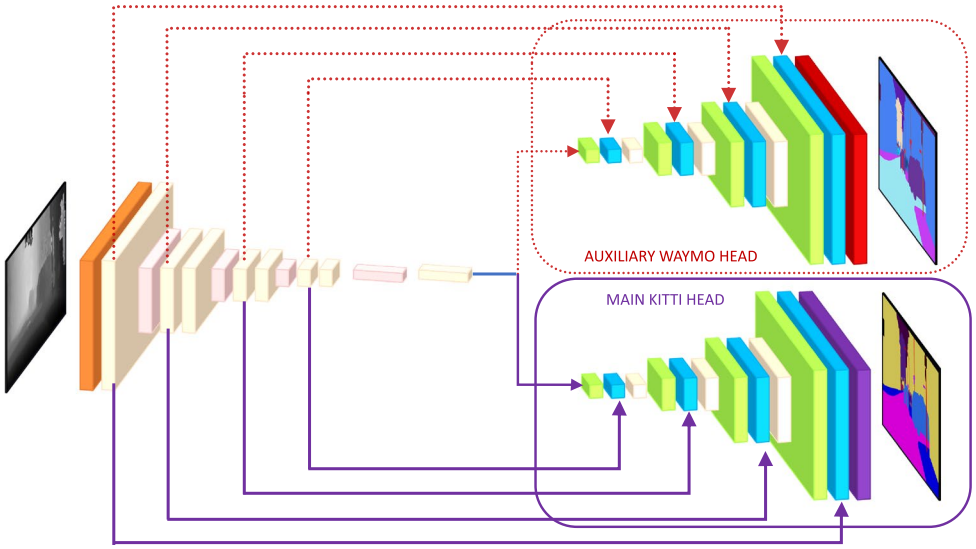

Fig. 1 SalsaNext model with an additional head for multi-dataset training. The layer types are color-coded as pink: average pooling with stride=2, orange: ResNet context blocks with stride=1, yellow: ResNet blocks with stride=1, blue: simple concatenate, green: bilinear upsampling with a convolutional layer, purple: last convolutional layer with 20 channels, and red: last convolutional layer with 23 channels

Table 1 The proposed model's layer counts

Layer	Conv2d	ReLU6	BatchNorm2d	Bilinear Upsample	Pooling
Count	55	54	46	4	4

although there are overlapping classes in their annotations, they cannot be fully mapped to each other. Thus, domain adaptation is required for both inputs and labels. Therefore, the following adjustments were applied to address the domain shift between Waymo and SemanticKITTI:

- **Class mapping:** The Waymo dataset contains 23 classes, and the Waymo head was designed to output the same number of classes. In contrast, SemanticKITTI includes 34 classes, which were mapped to 20 classes as per [5]. The number of channels in the KITTI head was adjusted accordingly.
- **Range image regeneration:** Waymo range images were regenerated to match SemanticKITTI characteristics. x, y, and z coordinates were extracted alongside the range image. The regenerated range image was aligned to SemanticKITTI sensor specifications by setting the height from the ground to 1.75 m, rearranging the coordi-

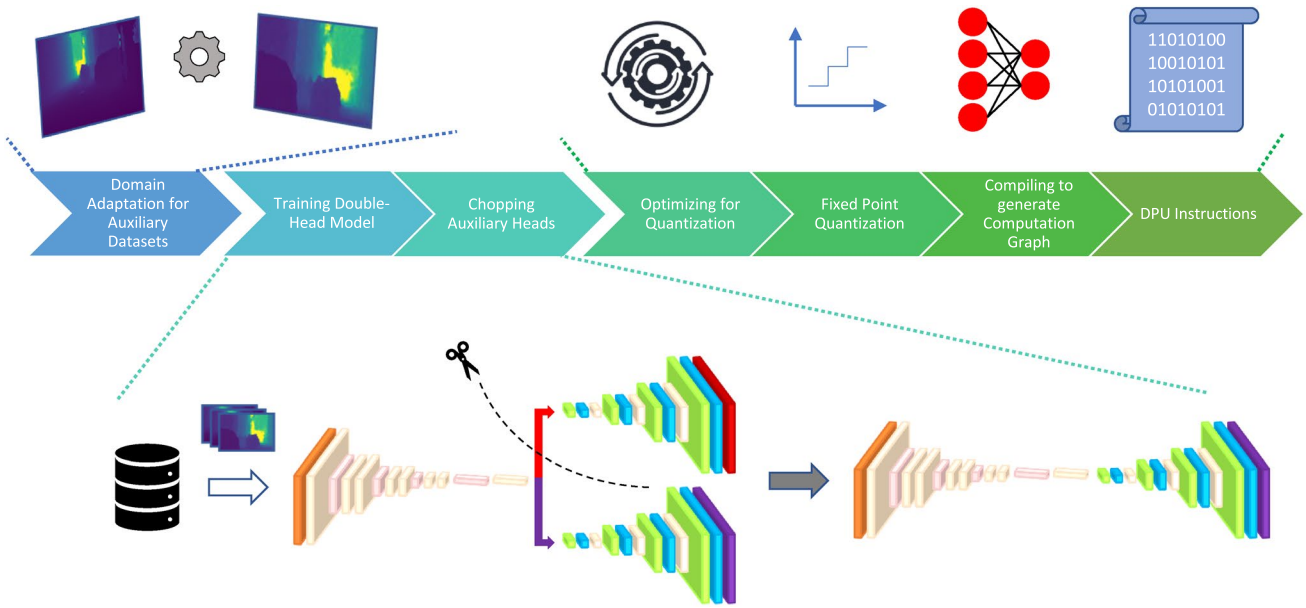

Fig. 2 The proposed edge implementation pipeline

Table 2 Comparison of key features across major autonomous driving LiDAR datasets

Feature	nuScenes	SemanticKITTI	Waymo Open Dataset
Sensor	Velodyne VLP-32C	Velodyne HDL-64E	Custom 64-beam
Vertical Resolution	32	64	64
Horizontal Resolution	1080	2048	2650
Vertical FOV	-30° to +10°	−24.3° to +2°	−18° to +2.3°
Range	100 m	120 m	75 m
# of Train Sweeps	28,130	19,130	23,691
# of Valid Sweeps	6,019	4,071	5,976
# of Test Sweeps	6,008	20,351	2,982
# of Semantic Classes	32	28	23

nate frame, and adjusting the field-of-view (FoV) angles to match SemanticKITTI (−25° to 3°).
- **Intensity mapping:** Intensity values from Waymo, which can reach thousands, were clipped and mapped to the range [0, 1] to ensure a similar cumulative distribution function (CDF) as SemanticKITTI.

On the other hand, to adapt the base *SalsaNext* model for FPGA deployment, the following modifications were implemented:

- **Layer reordering:** Batch normalization layers were reordered with activation layers to optimize the fusion mechanism, applied by Vitis-AI tools (conv+bn+act).
- **Activation replacement:** If the network feature values exceed the fixed-point range, the related layers are mapped to the CPU. To avoid this, we applied ReLU6 as an activation function throughout the entire network.
- **Upsampling adjustments:** Since pixel shuffle is not supported by DPU, pixel shuffle upsampling was substituted with bilinear upsampling, followed by convolution, batch normalization, and activation layers.
- **Block splitting:** The model was divided into encoder and decoder components to separate the feature extractor backbone and facilitate the multi-head architecture.
- **Dual training:** Two separate decoder heads were trained: one for the KITTI dataset and another for Waymo.
- **Model simplification:** After training, the Waymo-specific head was removed to create a compact, single-head model.
- **Activation conversion:** ReLU6 activations were converted to HardTanh to address quantization issues in Vitis-AI conversion tools, ensuring accurate representation without accuracy loss.
- **Loss function:** The overall loss was computed as a weighted combination of losses from the two heads, with a ratio of 4:1. Each head's loss comprised a class-balanced cross-entropy loss and a Lovász loss [33], following the approach in [5]. For the cross-entropy loss, the weight assigned to the `unlabeled` class was adjusted to `min(class_weights)/50`, enhancing accuracy and awareness of unlabeled regions.

After training, we employed 8-bit fixed-point quantization and compiled the model for specific DPU configurations (`DPUZCXDG-ISA1-b4096` and `DPUZCXDG-ISA1-b3136`). The quantized model was deployed and tested on the Kria KV260 board for both configurations.

Figure 3 shows the DPU processing pipeline. For efficient data transfer between DDR and PL (or DPU), 3 strategies are used: First, data are transferred via AXI parallel interface. Secondly, there is a dedicated on-chip memory inside PL made of URAMs and BRAMs. This reduces the need for DDR access. Thirdly, Direct Memory Access (DMA) is used to directly access DDR without CPU intervention.

The inference process begins when the DPU fetches instructions from its dedicated instruction memory, which may reside in on-chip buffers or be accessed via the AXI interface from external DDR memory. These instructions include operation codes for various neural network tasks such as convolution, activation, and pooling. Once the instructions are fetched, the DPU's control unit decodes them to determine the specific operations to be performed. This decoding phase directs the subsequent actions by routing instructions to the appropriate compute units within the DPU.

In parallel with instruction decoding, the DPU initiates data transfers to prepare the operands for computation. Data read operations-managed via DMA and the AXI interface-load the required input feature maps and filter weights from DDR or on-chip memory (such as URAMs and BRAMs) into the processing units. With the data available, the dedicated MAC units perform the convolution operations, calculating dot products between input data and weights to produce intermediate feature maps. Once computed, these results are written back to memory, either to local caches or directly to DDR, ensuring that the outputs are available for subsequent processing stages like activation functions or pooling. This continuous cycle of reading, processing, and writing ensures that the DPU maintains a steady flow of data

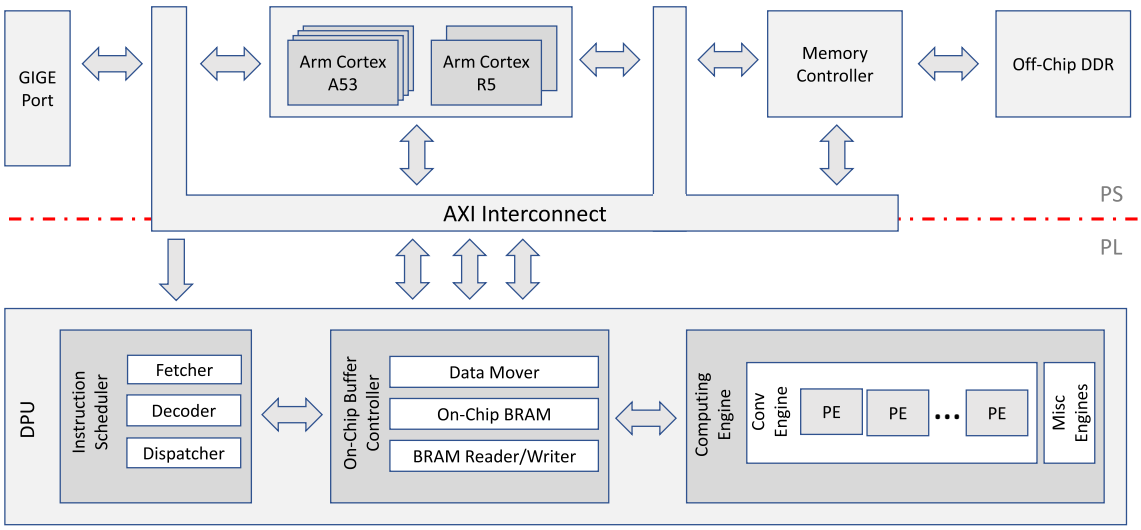

Fig. 3 DPU processing pipeline

through its highly optimized pipeline, thereby maximizing throughput and minimizing latency.

3 Experiments

For qualitative evaluation, we compared the performance of our float model on the SemanticKITTI and Waymo Open datasets, using only the KITTI-specific head for inference in both cases. Figure 4 shows the performance of our proposed models on samples from the SemanticKITTI and domain-adapted Waymo validation datasets. In the first SemanticKITTI example, our models-unlike the original model-successfully infer unlabeled regions too, enhancing their overall reliability. Moreover, the double-head model outperforms both the single-head version and the original model on the Waymo sample. For example, while the original model misclassifies pedestrians as other vehicles, cars, or even buildings, our single-head model correctly labels about half of these instances, and the double-head model nearly classifies all of them correctly.

For quantitative comparison, firstly, we evaluated our models' performance against the original model on three datasets: SemanticKITTI, Waymo, and domain-adjusted Waymo datasets. The three compared models include the original SalsaNext [5], and our customized single-head and double-head versions of SalsaNext. As shown in Table 3, the double-head SalsaNext achieves the best performance on the domain-adjusted Waymo dataset. Specifically, the performance improvement due to domain adjustment and double-head training is 12.25% in the mIoU metric and 26.34% in accuracy (or precision) compared to the original model.

Table 4 shows the comparison of classwise IoU scores on SemanticKITTI validation dataset with the baseline float models. According to the baseline, SalsaNext-based models exhibit the best performance in terms of mIoU and accuracy. Our single-head and double-head models achieve approximate results with the original implementation with a small degradation in mIoU (−0.04% for single-head and −0.22% for double-head models).

To evaluate the impact of different DPU architectures, the model was implemented with two configurations: B4096 and B3136, with their resource utilizations detailed in Table 5. Test results reveal that the B4096 configuration exhibits superior performance in terms of speed and accuracy. Specifically, the B4096 configuration achieves 5.6 frames per second (fps), whereas the B3136 configuration reaches only 3.85 fps, reflecting a 32% reduction in speed. This difference arises because the DPU-B4096 architecture can handle greater data throughput compared to DPU-B3136. Detailed results for both configurations are summarized in Table 6.

To further evaluate DPU-B4096 architecture, we measured DDR read and write data rates and plotted them in Fig. 5. As shown from the figure, thanks to direct memory access and parallel AXI interface, for a single port, the maximum read rate exceeds 3GB/s and the maximum write rate exceeds 1.5GB/s.

The latencies for real-time performance are given in Table 7. In terms of data collection, since we cannot reach a 64-channel LIDAR sensor, we read data from the image file on disk which takes 3 ms. However, we include sensor readout delays in experimental results. Most of the LIDAR sensors support Gigabit Ethernet (GigE) with a theoretical speed of 125MB/s. Our card also supports GigE. However, due to protocol overhead and other delays, the speed is

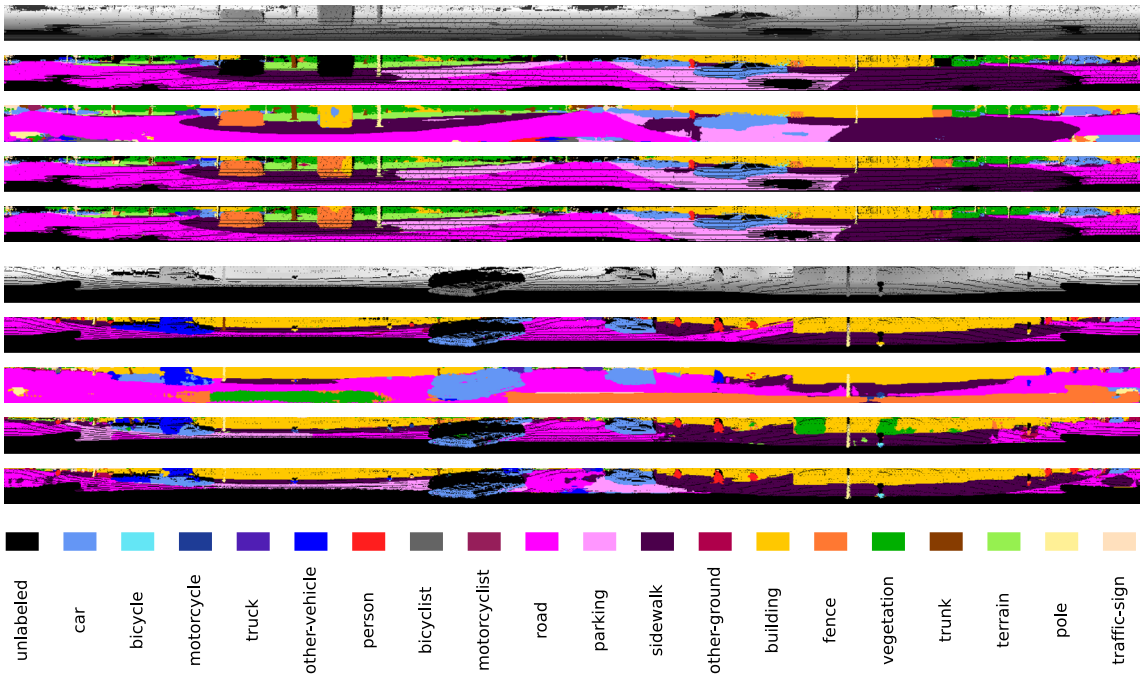

Fig. 4 First half: Inference results for a sample from the SemanticKITTI dataset. From top to bottom: 1) Range image 2) Label 3) Inference result from original model [5] 4) Inference result from our single head model 5) Inference result from Kitti head of our double headed model. Second half: Inference results for a sample from the Waymo dataset after applying domain adjustment. From top to bottom: 6) Range image 7) Label 8) Inference result from original model [5] 9) Inference result from our single head model 10) Inference result from Kitti head of our double headed model

Table 3 Performance comparison of original SalsaNext [5] with our customized single-head and double-head versions on SemanticKITTI, Waymo, and domain-adjusted Waymo datasets

Method	SemanticKITTI		Waymo		Waymo_DA	
	mIoU	Accuracy	mIoU	Accuracy	mIoU	Accuracy
SalsaNext-orig [5]	**61.01**	90.14	9.93	30.16	14.02	43.8
Single-Head [ours]	60.97	90.48	12.5	**31.83**	15.21	46.31
Double-Head [ours]	60.79	**90.71**	**13.2**	31.26	**22.18**	**56.5**

Bold values indicate the best performance for each metric across each dataset

generally between 80–100 MB/s. We assumed the transfer speed be 80MB/s. Besides we assumed 12 bytes of data packet for each point, and 131k points per sweep. Therefore, we calculated 19.67 ms of transfer delay. We included this transfer delay in our calculations. The rest of the latencies are based on the real-time measurements on the board, including preprocessing, DPU inference, and post-processing. According to these results, overall latency reaches 259 ms on the board.

Table 8 provides a quantitative comparison of our model against baseline segmentation models implemented on FPGA, as well as CNN-based semantic segmentation models executed on FPGA platforms. Our quantized model achieves the highest mIoU, with a total power consumption of 9W for the entire board. Although our inference frame rate is low, it remains within the operational range of most of the commercial LiDAR sensors (5–15 Hz).

In Table 8 SalsaNext-based models [5, 24] (including our models too) exhibit the best performance in terms of mIoU and accuracy. It is worth noting that there is a small, but important difference between our study and [5, 24], Since we changed the activation function from LeakyRELU to RELU6, the numbers during inference cannot explode, which makes the model suitable for fixed-point representation where the scale is limited. Without *LeakyReLU* $\longrightarrow$ *ReLU*6 conversion we could not quantize our model: Either the tools gave an error, or the generated model graph is partitioned between CPU and DPU which reduces the inference speed and increases the power consumption. Another SalsaNext implementation in [24] achieved to convert the model with LeakyReLU because they applied QAT beforehand.

Table 4 Comparison of class-wise IoU scores of float models on SemanticKITTI validation dataset

Method	Mean IoU	Car	Bicycle	Motor-cycle	Truck	Other-vehicle	Person	Bicy-clist	Motor-cyclist	Road	Parking	Side-walk	Other-ground	Build-ing	Fence	Vegeta-tion	Trunk	Terrain	Pole	Traffic-sign
SSGV3-K7 [22]	43.8	74.5	22.3	40.9	23.1	16.3	39.7	49.8	0	92.4	39.7	79.0	0.4	78.5	31.2	80.2	46.3	71.9	34.8	33.3
ResNet-GCM [21]	47.9	87.3	19.4	27.6	46.6	28.3	43.5	57.3	**1.6**	91.2	34.1	75.2	2.1	81.3	36.9	78.0	49.4	73.1	41.5	36.3
ResNet-NLA [23]	56.4	92.2	37.5	42.5	72.4	37.5	63.2	75.4	0	92.0	34.2	77.7	8.1	85.0	45.3	84.3	58.7	72.0	50.3	44.1
Sal-saNext-orig [5]	**61.0**	94.8	**48.3**	46.4	**79.1**	44.1	**73.2**	**82.9**	0.0	**94.9**	43.0	**81.8**	4.8	**87.0**	53.7	83.4	**67.0**	67.1	**60.2**	47.5
Proposed Work (SH)	60.9	94.3	39.5	51.3	75.3	**47.0**	71.4	79.6	0.0	94.2	**45.0**	81.3	**8.8**	86.9	**54.8**	84.3	65.7	71.4	58.0	49.7
Proposed Work (DH)	60.8	**94.8**	42.7	**55.9**	73.7	45.4	69.3	81.0	0.2	94.1	44.9	81.2	2.4	85.7	51.4	**85.4**	64.8	**74.1**	58.4	**49.7**

Bold values indicate the best class-wise mIoU for each class

Table 5 Resource utilization for 2 different DPU architectures

Resource	Available	DPU-B3136		DPU-B4096	
		Utilization	Utilization (%)	Utilization	Utilization (%)
LUT	117,120	45,640	38.97	50,867	43.43
LUTRAM	57,600	5171	8.98	7063	12.26
FF	234,240	79,642	34	98,797	42.18
BRAM	144	45.5	31.6	67.5	46.88
URAM	64	50	78.13	50	78.13
DSP	1248	566	45.35	710	56.89

Table 6 Performance results of the proposed double-head model after 8-bit quantization, on SemanticKITTI validation dataset, for different DPU configurations

Architecture	Mean IoU	Accuracy	Speed (fps)
DPU-B3136	54.28	87.78	3.85
DPU-B4096	56.92	88.31	5.6

However, before fine-tuning (QAT) their model achieves 58.8 mIoU while after QAT the quantized model achieves 54.2 mIoU. This means an overall 4.6% quantization error in mIoU, whereas our model experiences a lower (3.9%) drop even without QAT, thanks to ReLU6 conversion. We tried the Vitis quantization library to test our model performance with QAT, however, the training process was too slow, and we could not continue on that. However, we could apply QAT using the Pytorch quantization library (work with a similar principle as Vitis tools) to quantize our model to INT8 which resulted in 58% mIoU, meaning an even lower 2.9% quantization error in mIoU when applying QAT.

Table 7 Latencies of inference phases

Operation	Fetch	Pre-processing	Inference	Post-processing
Processor	CPU	CPU	DPU	CPU
Delay (ms)	19.67	50.60	175.44	12.99

4 Conclusion

In this work, we deployed a real-time point cloud segmentation model on the Kria KV260, a cost-effective FPGA platform, utilizing the lightweight CNN-based *SalsaNext* model. The model was trained on two commonly used large-scale LiDAR semantic segmentation datasets, namely

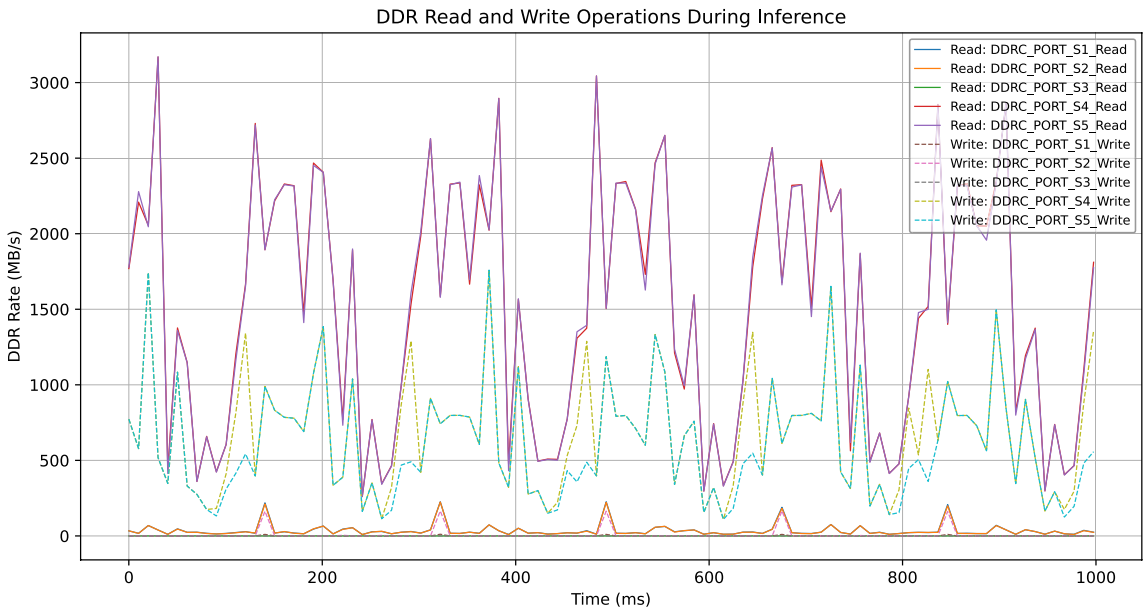

Fig. 5 DDR read and write rates during inference

Table 8 Comparison with the baseline of FPGA-based LIDAR segmentation models on SemanticKITTI dataset

Method	GOPs	mIoU	Accuracy	Power (W)	Speed (fps)	Freq. (M)	Platform	Efficiency (GOPs/W)	Infer/E2E Latencies (ms)
ResNet-NLA [23]	71.4	56.4/–	–	16.8 (all)	10.0	300	ZCU102	42.5	100/–
SSGV3-K7 [22]	–	45/43.8	87/85.5	74 (peak)	4.8	–	VCK190	–	208/–
ResNet-GCM [21]	–	47.9/46.4	–	**0.71/5.43** (idle/infer)	**56.0**	250	ZCU104	–	**18**/–
SalsaNext-pruned [24]	33.3	58.8/54.2	–	–	6.4	300	KV260	–	–/–
Proposed Work (DH)	125.7	**60.8/56.9**	**90.7/88.3**	5.18/9 (idle/infer)	5.6	300	KV260	**78.2**	175/259

Bold values indicate the best values

mIoU, Accuracy, Power Efficiency, and Latency metrics are given for both float and quantized models. For DPU-based implementations, results for single-thread, single-DPU configuration are noted except [23] which has 2 DPU cores

SemanticKITTI and the Open Waymo Dataset. To mitigate performance degradation caused by domain shifts between datasets, we proposed a domain adjustment schema, which is particularly beneficial for 2D range image-based models. Although the domain shift between these datasets is significant, our approach-incorporating proper weighting on loss functions and a double-headed model-achieved a +12.25% increase in the mean Intersection over Union (mIoU) metric on the Waymo validation dataset when cross-inference was applied to the KITTI head, owing to the domain adjustment and double-head training schema.

After training, the model was simplified by removing the Waymo head, resulting in a single compact model optimized for edge deployment. The compact model was further refined and quantized to 8 bits for efficient hardware execution. On the SemanticKITTI validation dataset, our implementation outperformed baseline models, achieving a mIoU of 56.9% and a frame rate of 5.6 fps. This represents only a 3.8% reduction in mIoU compared to its floating-point counterpart. Additionally, the quantized model achieved an overall accuracy of 88.3%, with a minor 2.4% reduction compared to the floating-point version.

Despite the relatively limited computational resources of the Kria KV260, our optimized approach delivered superior performance compared to other FPGA-based models on the SemanticKITTI validation dataset, achieving the highest reported mIoU and accuracy metrics. Qualitative analyses further highlight the model's consistent performance across both datasets, emphasizing its robustness and adaptability.

Acknowledgements The numerical calculations presented in this study were partially conducted at TÜBİTAK ULAKBİM, High Performance and Grid Computing Center, and the National Center for High Performance Computing of Turkey (UHeM), under grant number 1020162024. Additionally, supplementary cloud computing resources were provided by `vagon.io`.

Funding Open access funding provided by the Scientific and Technological Research Council of Türkiye (TÜBİTAK).

Data Availability The source code and additional resources related to this paper will be available on GitHub at: https://github.com/dymnmysn/hydrasalsa.

Open Access This article is licensed under a Creative Commons Attribution 4.0 International License, which permits use, sharing, adaptation, distribution and reproduction in any medium or format, as long as you give appropriate credit to the original author(s) and the source, provide a link to the Creative Commons licence, and indicate if changes were made. The images or other third party material in this article are included in the article's Creative Commons licence, unless indicated otherwise in a credit line to the material. If material is not included in the article's Creative Commons licence and your intended use is not permitted by statutory regulation or exceeds the permitted use, you will need to obtain permission directly from the copyright holder. To view a copy of this licence, visit http://creativecommons.org/licenses/by/4.0/.

References

1. Wu, X., Jiang, L., Wang, P.-S., Liu, Z., Liu, X., Qiao, Y., Ouyang, W., He, T., Zhao, H.: Point transformer v3: Simpler faster stronger. In: Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition (CVPR), pp. 4840–4851 (2024)
2. Lai, X., Chen, Y., Lu, F., Liu, J., Jia, J.: Spherical transformer for lidar-based 3d recognition. In: 2023 IEEE/CVF Conference on Computer Vision and Pattern Recognition (CVPR), pp. 17545–17555 (2023). https://doi.org/10.1109/CVPR52729.2023.01683
3. Zhou, Y., Tuzel, O.: VoxelNet: end-to-end learning for point cloud based 3D object detection (2017). arxiv:1711.06396
4. Aksoy, E.E., Baci, S., Cavdar, S.: Salsanet: Fast road and vehicle segmentation in lidar point clouds for autonomous driving. In: 2020 IEEE Intelligent Vehicles Symposium (IV), pp. 926–932 (2020). https://doi.org/10.1109/IV47402.2020.9304694
5. Cortinhal, T., Tzelepis, G., Erdal Aksoy, E.: Salsanext: Fast, uncertainty-aware semantic segmentation of lidar point clouds. In: Advances in Visual Computing: 15th International Symposium, ISVC 2020, San Diego, CA, USA, October 5-7, 2020, Proceedings, Part II, pp. 207–222. Springer, Berlin, Heidelberg (2020). https://doi.org/10.1007/978-3-030-64559-5_16
6. Xu, C., Wu, B., Wang, Z., Zhan, W., Vajda, P., Keutzer, K., Tomizuka, M.: Squeezesegv3: Spatially-adaptive convolution for efficient point-cloud segmentation. In: Vedaldi, A., Bischof, H., Brox, T., Frahm, J.-M. (eds.) Computer Vision - ECCV 2020, pp. 1–19. Springer, Cham (2020)
7. Milioto, A., Vizzo, I., Behley, J., Stachniss, C.: Rangenet ++: Fast and accurate lidar semantic segmentation. In: 2019 IEEE/

RSJ International Conference on Intelligent Robots and Systems (IROS), pp. 4213–4220 (2019). https://doi.org/10.1109/IROS40897.2019.8967762
8. Liu, Y., Chen, R., Li, X., Kong, L., Yang, Y., Xia, Z., Bai, Y., Zhu, X., Ma, Y., Li, Y., Qiao, Y., Hou, Y.: UniSeg: a unified multi-modal LiDAR segmentation network and the OpenPCSeg Codebase. In: 2023 IEEE/CVF International Conference on Computer Vision (ICCV), pp. 21605–21616. IEEE Computer Society, Los Alamitos, CA, USA (2023). https://doi.org/10.1109/ICCV51070.2023.01980
9. Xu, X., Kong, L., Shuai, H., Liu, Q.: FRNet: Frustum-range networks for scalable LiDAR segmentation (2024). arxiv:2312.04484
10. Thomas, H., Qi, C., Deschaud, J.-E., Marcotegui, B., Goulette, F., Guibas, L.J.: Kpconv: flexible and deformable convolution for point clouds. In: 2019 IEEE/CVF International Conference on Computer Vision (ICCV), pp. 6410–6419 (2019)
11. Bai, Y., Li, G., Yang, C., Li, Y., Xiao, Q., Li, Z.: Differential graph convolution network for point cloud understanding. Neurocomputing **597**, 127940 (2024). https://doi.org/10.1016/j.neucom.2024.127940
12. Qi, C.R., Yi, L., Su, H., Guibas, L.J.: PointNet++: deep hierarchical feature learning on point sets in a metric space (2017). arxiv:1706.02413
13. Kong, L., Liu, Y., Chen, R., Ma, Y., Zhu, X., Li, Y., Hou, Y., Qiao, Y., Liu, Z.: Rethinking range view representation for lidar segmentation. In: 2023 IEEE/CVF International Conference on Computer Vision (ICCV), pp. 228–240 (2023). https://doi.org/10.1109/ICCV51070.2023.00028
14. Kleesiek, J., Urban, G., Hubert, A., Schwarz, D., Maier-Hein, K., Bendszus, M., Biller, A.: Deep MRI brain extraction: a 3D convolutional neural network for skull stripping. NeuroImage **129**, 460–469 (2016). https://doi.org/10.1016/j.neuroimage.2016.01.024
15. Çiçek, Abdulkadir, A., Lienkamp, S.S., Brox, T., Ronneberger, O.: 3D U-Net: learning dense volumetric segmentation from sparse annotation (2016). arxiv:1606.06650
16. Zhou, Y., Tuzel, O.: VoxelNet: end-to-end learning for point cloud based 3D object detection (2017). arXiv:1711.06396
17. Qi, C.R., Su, H., Mo, K., Guibas, L.J.: PointNet: Deep learning on point sets for 3D classification and segmentation (2017). arXiv:1612.00593
18. Bai, L., Lyu, Y., Xu, X., Huang, X.: Pointnet on FPGA for real-time lidar point cloud processing. In: 2020 IEEE International Symposium on Circuits and Systems (ISCAS), pp. 1–5 (2020). https://doi.org/10.1109/ISCAS45731.2020.9180841
19. Zhang, X., Huang, Z., Antony, G.G., Jachimczyk, W., Huang, X.: Stream-based ground segmentation for real-time LiDAR point cloud processing on FPGA (2024). arXiv:2408.10410
20. Zhang, X., Huang, Z., Antony, G.G., Jachimczyk, W., Huang, X.: Parallel processing of point cloud ground segmentation for mechanical and solid-state lidars. CoRR arXiv:2408.10404 (2024) https://doi.org/10.48550/ARXIV.2408.10404
21. Xie, X., Bai, L., Huang, X.: Real-time lidar point cloud semantic segmentation for autonomous driving. Electronics **11**(1), 11 (2021)
22. Delgado, P.P.F.: Real-time implementation of 3D LiDAR point cloud semantic segmentation in an FPGA (2022)
23. Bai, L., Zhao, Y., Huang, X.: A near sensor edge computing system for point cloud semantic segmentation. In: 2022 IEEE International Symposium on Circuits and Systems (ISCAS), pp. 1818–1822 (2022). https://doi.org/10.1109/ISCAS48785.2022.9937678
24. Xilinx, I.: Vitis AI Model Zoo: Reference Guide. (2024). Accessed: 2024-12-14. https://xilinx.github.io/Vitis-AI/3.0/html/docs/reference/ModelZoo_VAI3.0_Github_web.htm
25. NVDLA: Open Source Deep Learning Inference Accelerator. https://nvdla.org. Accessed: 2025-02-02
26. Xilinx Deep Learning Processor Unit (DPU). https://docs.amd.com/r/3.3-English/pg338-dpu. Accessed: 2025-02-02
27. He, K., Zhang, X., Ren, S., Sun, J.: Deep residual learning for image recognition (2015). arXiv:1512.03385
28. Chen, L.-C., Papandreou, G., Schroff, F., Adam, H.: Rethinking atrous convolution for semantic image segmentation (2017). arXiv:1706.05587
29. Yi, L., Gong, B., Funkhouser, T.: Complete & Label: a domain adaptation approach to semantic segmentation of LiDAR point clouds (2021). arXiv:2007.08488
30. Wu, B., Zhou, X., Zhao, S., Yue, X., Keutzer, K.: SqueezeSegV2: Improved model structure and unsupervised domain adaptation for road-object segmentation from a LiDAR point cloud (2018). arXiv:1809.08495
31. Saleh, K., Abobakr, A., Attia, M., Iskander, J., Nahavandi, D., Hossny, M.: Domain adaptation for vehicle detection from bird's eye view LiDAR point cloud data (2019). arXiv:1905.08955
32. Qin, C., You, H., Wang, L., Kuo, C.-C.J., Fu, Y.: Pointdan: A multi-scale 3D domain adaption network for point cloud representation. In: Wallach, H., Larochelle, H., Beygelzimer, A., Alché-Buc, F., Fox, E., Garnett, R. (eds.) Advances in Neural Information Processing Systems, vol. 32. Curran Associates, Inc., Red Hook (2019)
33. Berman, M., Triki, A.R., Blaschko, M.B.: The Lovász-Softmax loss: A tractable surrogate for the optimization of the intersection-over-union measure in neural networks (2018). arXiv:1705.08790

Publisher's Note Springer Nature remains neutral with regard to jurisdictional claims in published maps and institutional affiliations.