2025 IEEE International Conference on Systems, Man, and Cybernetics (SMC)
October 5-8, 2025. Vienna, Austria

2025 IEEE International Conference on Systems, Man, and Cybernetics (SMC) | 979-8-3315-3358-8/25/$31.00 ©2025 IEEE | DOI: 10.1109/SMC58881.2025.11343519

Distributed Processing of Deep Learning Models for Multi-Sensor Fusion

Robert Rauch
dept. of computers and informatics
Technical University of Kosice
Kosice, Slovakia
robert.rauch@tuke.sk

Marco Levorato
dept. of computer science
University of California, Irvine
Irvine, USA
levorato@uci.edu

Juraj Gazda
dept. computers and informatics
Technical University of Kosice
Kosice, Slovakia
juraj.gazda@tuke.sk

***Abstract*—Connected Autonomous Vehicles (CAVs) are equipped with an array of sensors generating substantial data streams whose real-time analysis often exceed the onboard computational capabilities. While offloading these computational tasks to edge servers is an established solution, such an approach is increasingly challenging as multiple sensor streams need to be fused and analyzed, and thus transferred over capacity-limited and volatile wireless channels. To address this challenge, in this paper we propose a "split computing" framework. Compared to existing solutions that focus on individual sensor streams (and most commonly cameras), our framework is designed for sensor fusion neural models, and specifically camera and LiDAR fusion for semantic segmentation. Our proposed framework optimizes data transfer by eliminating pooling indices in favor of sending LiDAR point cloud indices only to the final network block. We remark how sensor fusion is instrumental to guarantee robust operations in a broad range of conditions. By compressing the data to be transported over the channel, our approach reduces offloading latency, better utilizes CAV computational resources compared to full offloading schemes and decreases channel load in congested urban network scenarios. Our experimental results demonstrate up to 62.74% improvement in task execution latency for sensor fusion models, with at most 6.61% performance trade-off due to compression.**

***Index Terms*—Connected autonomous vehicles, LiDAR semantic segmentation, sensor fusion, split computing, task offloading**

I. Introduction

The rapid advancement of Deep Learning (DL) is transforming Intelligent Transportation Systems (ITS), with Connected Autonomous Vehicles (CAVs) representing one of the most promising applications. High-performance, low-latency perception remains a critical challenge for CAVs [1]. While DL models have emerged as state-of-the-art solutions for perception tasks [2], their complexity combined with massive data generation makes real-time on-CAV computing increasingly challenging [3].

This work was supported by The Slovak Research and Development Agency project no. APVV SK-CZ-RD-21-0028, APVV-23-0512, the Slovak Academy of Sciences project no. VEGA 1/0685/23, and the US National Science Foundation under grant CNS 2134567 and CCF 2140154.
Part of the Research results were obtained using the computational resources procured in the national project National competence centre for high performance computing (project code: 311070AKF2) funded by European Regional Development Fund, EU Structural Funds Informatization of society, Operational Program Integrated Infrastructure.

To address these limitations, Vehicular Edge Computing (VEC) enables CAVs to offload computationally expensive tasks to edge servers [4], supporting advanced capabilities in vehicle networking [5], localization [6], and collaborative perception [7]. However, complete offloading underutilizes CAVs' computational capabilities. Split computing [8] offers a more balanced approach for CV tasks, strategically dividing computing load between CAV and edge server, reducing latency in adverse channel conditions and increasing server throughput.

A key challenge in split computing is identifying optimal split points, as most generate extensive intermediate data [8]. Compression through autoencoder DL models [9] is well-researched, though such methods are inherently lossy. Despite existing work on split computing [10], a research gap exists for CAV-specific tasks like LiDAR semantic segmentation, which presents unique challenges compared to traditional image segmentation.

While camera-based segmentation approaches have proven successful [11], these single-modal methods face limitations from varying luminosity and adverse weather [12]. Multi-sensor fusion approaches offer more robust performance across environmental conditions [12], but haven't explored task offloading through split computing to enhance CV model execution speed.

In this paper, building upon the aforementioned split computing research, we explore split computing models for LiDAR and sensor fusion segmentation. According to [13], sensor fusion can be categorized as early fusion, deep fusion, or late fusion. We focus on early fusion, exploring its characteristics and optimization opportunities. While CNN encoder-decoder architectures effectively process sensor-fused data, their pooling indices create substantial overhead in split computing. We propose a modified architecture that preserves sparse LiDAR point cloud position information while minimizing data transfer by eliminating intermediate dependencies and transferring only essential indices to the final network block.

The main contributions of this paper can be summarized as follows:

- The first design of a split computing model for LiDAR

semantic segmentation task, featuring efficient reconstruction through strategic LiDAR point cloud position index transfer.
- A thorough analysis and evaluation of splitting strategies for early sensor fusion.
- An extensive simulation campaign based on the NuScenes dateset and wireless channel modeling using Hata path loss and Rayleigh fading demonstrating the advantages of split computing compared to *Edge Server Only* and *CAV Only* in system evaluation, while also comparing *LiDAR Only (Unpool)*, *LiDAR Only (Proposed)*, and *Early Fusion (Unpool)* approaches to evaluate our model modifications. Notably different from most contributions we consider a multi-vehicle system.

The remainder of this paper is organized as follows. Section II presents the system model. In Section III, we formulate the optimization problem. The proposed split computing approach for sensor fusion is detailed in Section IV. Section V provides experimental evaluation of our model. Finally, Section VI concludes the paper.

II. System Model

In this section, we describe our system model, showing the pipeline between CAVs and an edge server . We have a set of vehicles $\mathcal{V} = \{v_1, v_2, ..., v_{N_\mathcal{V}}\}$, each generating tasks $\mathcal{Z}_{t,v} = \{z_{1,v}, z_{2,v}, ..., z_{N_{\mathcal{Z}_{t,v}}}\}$ within time interval t. Each task has specific computational requirements and data size for offloading. A single base station with edge server serves these vehicles.

Each task involves a DL-based CV model composed of blocks $\mathcal{B} = \{b_1, b_2, ..., b_{N_\mathcal{B}}\}$. Each block is defined as $b = \{I_b, d_c^{\text{UL}}, N_b^{\text{channels}}\}$, where I_b represents computational requirements in Floating Point Operations (FLOPs), d_c^{UL} is the compressed output size, where output is compressed following methodology in [9], and N_b^{channels} is the original number of feature channels. Parameter $s \in \mathcal{B}$ represents the split point in our CV model.

A. Communication Model

The data rate between CAV $v \in \mathcal{V}$ and edge server is calculated as [14]:

$$r_{t,v} = N_t^{\text{bit}} \rho_v \frac{N^{\text{RB}}}{N_\mathcal{V}} R, \tag{1}$$

where N_t^{bit} is bits per symbol, ρ_v is the CAV's code rate, both determined by the modulation and coding scheme [15]. We use Rayleigh channel fading [16] and the Hata path loss model [17] for SINR calculations. N^{RB} denotes resource blocks, $N_\mathcal{V}$ is the number of vehicles, and R is the available symbol rate.

Following [18], we consider a quasi-static channel with static conditions during each offloading process.

The uplink and downlink latencies for task $z \in \mathcal{Z}_{t,v}$ are given by:

$$t_{z,c}^{\text{UL}} = \frac{d_c^{\text{UL}}}{r_{t,v}^{\text{UL}}} + t_{z,w}^{\text{UL}}, \tag{2}$$

$$t_z^{\text{DL}} = \frac{d_z^{\text{DL}}}{r_{t,v}^{\text{DL}}} + t_{z,w}^{\text{DL}}, \tag{3}$$

where d_c^{UL} and d_z^{DL} are uplink and downlink data sizes respectively, $r_{t,v}^{\text{UL}}$ and $r_{t,v}^{\text{DL}}$ are the respective data rates from Equation (1), and $t_{z,w}^{\text{UL}}$ and $t_{z,w}^{\text{DL}}$ represent the corresponding queuing delays [19].

B. Computing Model

Next, we present our computing pipeline. We define computational requirements (in FLOPs) for CAV $v \in \mathcal{V}$ and edge server as:

$$I_{v,z,s,c} = \sum_{b=1}^{s} I_b + I_c^{\text{encoder}}, \tag{4}$$

$$I_{z,s,c} = \sum_{b=s+1}^{N_\mathcal{B}} I_b + I_c^{\text{decoder}}, \tag{5}$$

where I_b is the computational load of block $b \in \mathcal{B}$, and s is our split point. I_c^{encoder} and I_c^{decoder} represent the computing load of encoder and decoder for compression c.

Execution latencies on CAV and edge server are:

$$t_{v,z,s,c}^{\text{COMP}} = \frac{I_{v,z,s,c}}{\eta_v} + t_{v,z,w}^{\text{COMP}}, \tag{6}$$

$$t_{z,s,c}^{\text{COMP}} = \frac{I_{z,s,c}}{\eta} + t_{z,w}^{\text{COMP}}, \tag{7}$$

where η_v and η denote computational capabilities in FLOPS, and $t_{v,z,w}^{\text{COMP}}$ and $t_{z,w}^{\text{COMP}}$ represent waiting times. We implement a FIFO queuing policy.

C. Total Latency and Performance Models

The total latency $t_{z,s}$ for a CV task is:

$$t_{z,s} = t_{z,c}^{\text{UL}} + t_z^{\text{DL}} + t_{v,z,s,c}^{\text{COMP}} + t_{z,s,c}^{\text{COMP}}. \tag{8}$$

For performance evaluation, we define several metrics. With segmentation classes $\mathcal{K} = \{k_1, k_2, ..., k_{N_\mathcal{K}}\}$, the IoU for class $k \in \mathcal{K}$ is [20]:

$$\mu_{k,s,c}^{\text{IoU}} = \frac{TP_{k,s,c}}{FP_{k,s,c} + TP_{k,s,c} + FN_{k,s,c}}, \tag{9}$$

where $TP_{k,s,c}$, $FP_{k,s,c}$, and $FN_{k,s,c}$ are true positive, false positive, and false negative points.

The mean IoU (mIoU) and Frequency Weighted mIoU (FWmIoU) [21] are:

$$\mu_{s,c}^{\text{mIoU}} = \frac{\sum_{k \in \mathcal{K}} \mu_{k,s,c}^{\text{IoU}}}{N_\mathcal{K}}, \tag{10}$$

$$\mu_{s,c}^{\text{FWmIoU}} = \frac{\sum_{k \in \mathcal{K}} (w_k \mu_{k,s,c}^{\text{IoU}})}{N_\mathcal{K}}, \tag{11}$$

where w_k is the weight based on class frequency.

Model accuracy is defined as:

$$\mu_{s,c}^{\text{Acc}} = \frac{N_{s,c}^{\text{correct}}}{N^{\text{points}}}, \tag{12}$$

where N^{correct} is number of correctly classified points and N^{points} is total number of points.

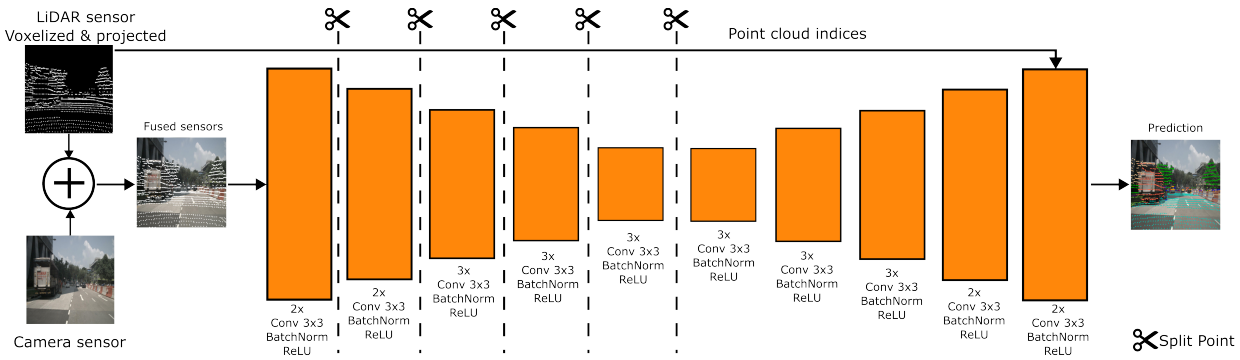


Fig. 1: Architectural overview of early sensor fusion implementation highlighting five potential split points along the encoder for split computing. The model comprises ten blocks with max pooling operations between encoder blocks and transpose convolution layers for upsampling in the decoder part of LiDAR segmentation model.

III. Problem Formulation

The overall goal of the system is to minimize the latency under task performance constraints. We formulate our optimization problem as follows:

$$\min_{\{s,c\}} \quad t_{z,s} \tag{13}$$

$$\text{s.t.} \quad s \in \mathcal{B} \tag{13a}$$

$$1 \leq c \leq N_s \tag{13b}$$

$$\mu^{\text{mIoU}}_{s,c} \geq \mu^{\min} \tag{13c}$$

$$\mu^{\min} \leq \mu^{\max} \tag{13d}$$

$$t_{z,s} \leq t^{\max} \tag{13e}$$

where constraint (13a) ensures the split point s is within $\mathcal{B}$. Constraint (13b) bounds feature channel compression between 1 and N_s. Constraint (13c) maintains performance above a minimum threshold using mIoU. The minimum threshold $\mu^{\min}$ must not exceed $\mu^{\max}$ per constraint 13d. Finally, constraint (13e) ensures total latency remains below $t^{\max}$.

IV. Split Computing for Sensor Fusion

To solve the optimization problem from the previous section, we leverage split computing to reduce data transmission overhead through intelligent task partitioning.

We focus on LiDAR semantic segmentation, introducing our complete pipeline in three parts: data pre-processing (Section IV-A), model architecture (Section IV-B), and our proposed split computing methodology (Section IV-C).

A. Data Preprocessing

Our LiDAR semantic segmentation pipeline fuses point cloud data with camera images as illustrated in Fig. 1.

The raw point cloud undergoes voxelization, discretizing 3D space into uniform cells. Each voxel is represented by seven features: 3D centroid coordinates, point density, and 3D eigenvalues characterizing local geometry. For detailed voxelization process, see [22].

We project the LiDAR point cloud to the image plane using camera intrinsic and extrinsic parameters similar to [23], removing points outside the camera's field of view.

Finally, we fuse image data with LiDAR data along the channel dimension, combining three channels from the camera with seven channels from the voxelized LiDAR representation. Where LiDAR points are absent, tensor values are set to zero. The resulting ten-channel tensor maintains image dimensions and is resized to 224×224 resolution.

B. Model Architecture

Convolutional Neural Networks (CNNs), particularly encoder-decoder architectures, are widely adopted for processing sensor-fused data [24], [25]. These architectures typically transfer information from encoder to decoder such as pooling indices or skip connections [26]. However, as demonstrated in our results (Section V), pooling indices generate substantial data volumes, making intermediate data offloading impractical.

While some studies have explored architectures without pooling indices for semantic image segmentation [27], this approach poses unique challenges when used on LiDAR data. The sparse nature of LiDAR point clouds, combined with the loss of precise spatial information during downsampling operations, makes it difficult for models to accurately reconstruct point positions in the decoder without position information. Unlike image data where interpolation approximates missing spatial information, LiDAR point clouds require precise position reconstruction. To address this, we propose a modified architecture that eliminates the need for intermediate decoder data transfer by employing transpose convolutions for upsampling operations. The transpose convolutions learn to predict spatial relationships, replacing the traditional role of pooling indices. This allows the model to learn effective upsampling strategies for data reconstruction: to maintain precise point localization while minimizing data transfer, we only pass point cloud position indices to the final network block, as illustrated in Fig. 1. This approach is efficient as the sparse nature of LiDAR data results in minimal position

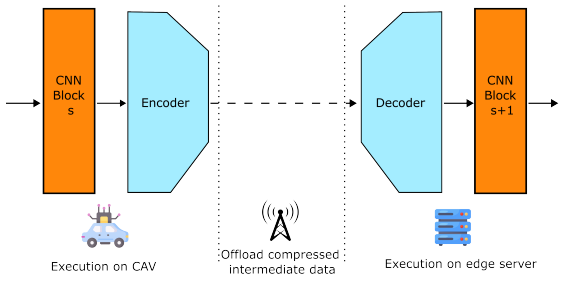


Fig. 2: Split computing architecture illustrating split point between consecutive blocks s and $s+1$, where $s \in \mathcal{B}$. Intermediate data from block s is compressed by the encoder, offloaded to the server, and reconstructed by the decoder.

indices, requiring significantly less data transfer compared to full point cloud information or pooling indices.

C. Split Computing

As illustrated in Fig. 1, we introduce potential split points after each encoder block, representing offloading points to the edge server. Following [8], we split between blocks rather than within blocks for optimal latency benefits. We limit splits to the encoder portion of architecture shown in Fig. 1 since decoder splits would increase data transmission to the CAV. For each split point $s \in \mathcal{B}$, features are compressed, transmitted, decompressed, and processing continues from block $s+1$. Architecture of a split point is depicted in Fig. 2.

Algorithm 1 Finding Optimal Compression Rate

Require: Initial channels c^{init}, Channel increment Δc, Performance improvement threshold τ, Minimum performance threshold μ^{min}, Maximum channels N_s

1: $c \leftarrow null$
2: $c' \leftarrow c^{\text{init}}$
3: $\mu^{\text{cur}} \leftarrow -\infty$
4: $\mu^{\text{prev}} \leftarrow 0$
5: **while** $c' < N_s$ **do**
6: Train encoder and decoder model with c' channels
7: $\mu^{\text{prev}} \leftarrow \mu^{\text{cur}}$
8: $\mu^{\text{cur}} \leftarrow$ performance of the model
9: **if** $\mu^{\text{cur}} > \mu^{\text{min}}$ **and** $\mu^{\text{cur}} - \mu^{\text{prev}} < \tau$ **then**
10: $c \leftarrow c'$
11: **break**
12: **end if**
13: $c' \leftarrow c' + \Delta c$
14: **end while**
15: **return** c as optimal number of channels

To determine the optimal channel compression c for split compression, we employ the elbow method [28], formalized in Algorithm 1. This method identifies the point where performance improvements diminish with increased channel size, indicating that additional data offloading would yield diminishing returns. The algorithm requires several parameters: initial channel size c^{init}, channel increment value Δc, performance improvement threshold τ, and minimum performance threshold μ^{min}.

The algorithm begins by initializing variables (lines 1-4) and enters an iterative process (line 5). The loop continues while current channels c' are below maximum channels N_s (i.e., number of channels from the block $s \in \mathcal{B}$). Within each iteration, we train the split compression model's encoder and decoder components using convolutional layers to compress features from N_s to c channels, following an approach similar to [9] (line 6). After training, we update performance metrics (lines 7-8). We then check for two conditions: if current performance μ^{cur} exceeds the minimum threshold μ^{min} (ensuring constraint 13c is satisfied), and if performance improvement falls below threshold τ, indicating we've reached diminishing returns (line 9). If both conditions are met, we set our channel compression c to c' and break the loop (lines 10-11). Otherwise, we increment c' by Δc (line 13). Finally, we return c (line 15). If no configuration satisfies both conditions throughout the process, the algorithm returns null, indicating that compression should not be applied at this split point.

V. Results and Evaluation

In this section, we evaluate: i) our proposed model modification for split computing, ii) compression for split computing, and iii) latency improvements from adjusting simulation parameters. Section V-A describes the model configuration and simulation setup, followed by baselines in V-B. We present evaluations of our LiDAR semantic segmentation model modifications in V-C and channel compression in V-D. Finally, we analyze split computing performance by varying resource blocks n^{RB} and CAV computational capability η_v.

A. Model & Simulation Setup

We evaluate LiDAR semantic segmentation on the NuScenes dataset [29], restructuring it into 560 training, 140 validation, and 150 test scenes (using the original validation set). This restructuring supports our primary objective of evaluating split computing effectiveness across different split strategies rather than optimizing absolute segmentation performance. Each scene contains 20-second keyframes at 2 Hz with multi-sensor data. Our early fusion model uses LiDAR and camera sensors, with 17 segmentation classes merged from the original 32 following NuScenes mapping[1], excluding the first class (non-essential objects) from metrics.

Models are trained for 50 epochs using cross entropy loss, Adam optimizer (lr=0.001), and checkpoint saving based on validation mIoU improvements. Our approach processes LiDAR data projected onto a single camera view but can extend to multiple views through either independent 2D convolutions or simultaneous 3D convolutions.

For compression networks at split points, we add a reconstruction objective where features are compressed through an encoder-decoder network. The reconstruction loss compares

[1]https://www.nuscenes.org/lidar-segmentation

TABLE I: Simulation setup

Parameter	Value
Number of resource blocks (N^{RB})	4000
Symbol rate (R)	2×10^9
Computational power of CAV v (η_v)	1.5×10^{12}
Computational power of edge server b (η)	75×10^{12}
Number of tasks per t ($N_{Z_{t,v}}$)	50
Number of CAVs ($N_{\mathcal{V}}$)	10
Number of simulation steps	3,000
Number of simulations	10

decoder output to original features using mean squared error, optimized with Adam (lr=0.001).

Each simulation is averaged over 10 runs of 3000 steps for statistical significance. Key parameters are shown in Table I.

Our wireless channel implementation calculates SINR using base station transmit power, Hata path loss (based on CAV-to-base station distance), and thermal noise [30]. SINR values are adjusted by Rayleigh channel fading, calculated using Doppler frequency that accounts for CAV velocity and transmission frequency. We sample Rayleigh fading at 1 kHz to capture temporal signal variations.

B. Baselines

Our evaluation uses two baseline sets: one assessing model modifications for split computing, another evaluating split computing and compression strategies.

To isolate the impact of our architectural modifications for split computing, we compare against different variants of the base segmentation model:

- *LiDAR Only (Unpool)*: Original architecture using unpooling indices in the decoder, processing only LiDAR data.
- *LiDAR Only (Proposed)*: Our modified architecture replacing unpooling with transpose convolutions, forwarding LiDAR point cloud indices only to the final decoder block.
- *Early Fusion (Unpool)*: Multi-modal approach fusing LiDAR and camera data early in the network, using original unpooling operations throughout the decoder.

Since no prior work exists for split computing in sensor fusion, we evaluate against the following computational distribution strategies:

- *Edge Server Only*: Centralized approach where LiDAR and camera data are directly offloaded to the edge server (full offloading).
- *CAV Only*: Fully local approach where all processing, including early fusion, is performed on the CAV (no offloading).

C. Model Evaluation

Table II demonstrates the necessity of our proposed modifications for split computing, showing averaged data sizes with an average input of 0.685 MB. We compare data transfer requirements between traditional pooling indices and our approach using LiDAR point cloud indices.

TABLE II: Data transfer sizes for different splits

Split	Size with Pooling Indices (MB)	Size with LiDAR Point Cloud Indices (MB)
Split 1	9.63	3.23
Split 2	11.24	1.63
Split 3	12.04	0.82
Split 4	12.44	0.42
Split 5	12.34	0.12

Input size: 0.685 MB

Traditional pooling indices make offloading impractical, requiring over 10 MB in most splits. Our approach using transpose convolutions and LiDAR point cloud indices significantly reduces data transferachieving up to 99.03% reduction. Notably, splits 4 and 5 require even less data than the original input (0.685 MB), making them promising candidates. These requirements can be further reduced through compression techniques explored in subsequent sections.

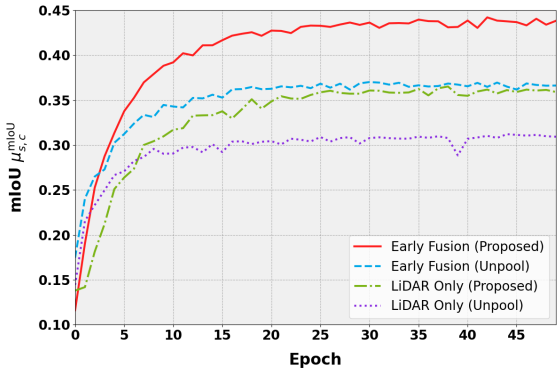


Fig. 3: Model performance (mIoU) $\mu_{s,c}^{\text{mIoU}}$ during 50 training epochs: comparison between our proposed approach (*Early Fusion (Proposed)*) and baseline approaches.

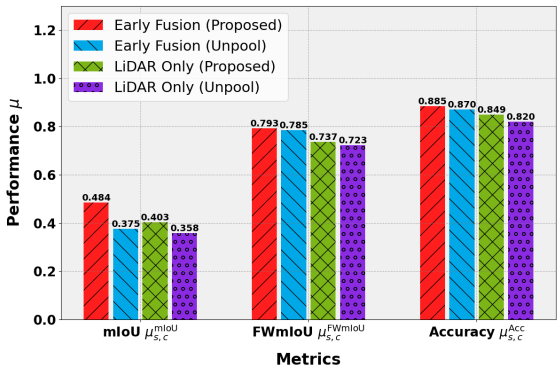


Fig. 4: Model performance across different metrics on test set: comparison between our proposed approach (*Early Fusion (Proposed)*) and baseline approaches.

Now, we evaluate performance of our proposed approach (*Early Fusion (Proposed)*) against the baselines. As shown in Fig. 3, our proposed approach achieves the highest performance during training on the validation set, outperforming all

baselines by a significant margin. Early fusion with unpool operations (*Early Fusion (Unpool)*) achieves the second-best performance, closely followed by our proposed architecture using LiDAR data only (*LiDAR Only (Proposed)*). The baseline using LiDAR only with unpool operations (*LiDAR Only (Unpool)*) shows the lowest performance.

Notably, our proposed architectural modifications enable the LiDAR-only approach to achieve comparable performance to early fusion with unpool operations, demonstrating the effectiveness of our proposed changes. Furthermore, our complete proposed approach, combining early fusion with architectural modifications, achieves up to 41.6% improvement in mIoU compared to the baseline methods on the validation set.

In Fig. 4, we present the evaluation results on the test set. Our proposed approach (*Early Fusion (Proposed)*) not only maintains its superior performance but achieves even higher results compared to training, suggesting better generalization capabilities. Specifically, the proposed approach demonstrates a significant improvement in performance, achieving up to 35.2% better mIoU $\mu_{s,c}^{\text{mIoU}}$ compared to the baselines. This improvement is consistent across all metrics, with notable gains in FWmIoU and accuracy as well.

D. Channel Compression Evaluation

In this section, we evaluate compression strategies for split computing, focusing specifically on the intermediate data between blocks of our LiDAR semantic segmentation model. We exclude the LiDAR point cloud data from compression as it is inherently sparse, averaging only 0.02 MB in size.

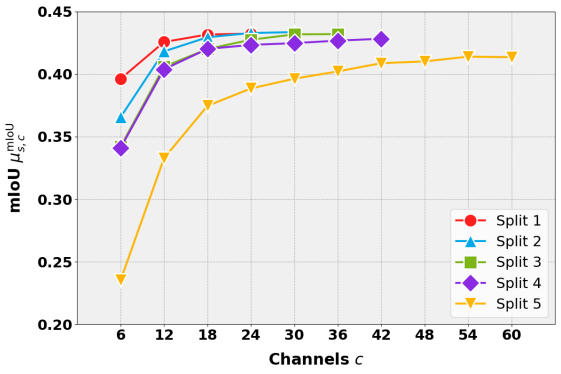


Fig. 5: Feature channel compression c optimization across different splits using Algorithm 1.

Fig. 5 illustrates the feature channel compression c applied to different splits following Algorithm 1. For our compression algorithm, we use mIoU as the performance metric with a minimum performance threshold $\mu^{\min} = 0.3978$, set to 90% of the validation mIoU. We use validation mIoU to reflect realistic conditions, terminating when performance change falls below $\tau = 0.001$.

Table III demonstrates the impact of our channel compression strategy. For each split point, we compare the original number of channels N_s and their corresponding data size

TABLE III: Channel compression impact on data transfer sizes

Split	Orig. Ch. N_s	Orig. Size (MB)	Comp. Ch. c	Comp. Size (MB)	Gain (%)
Split 1	64	3.23	24	1.22	62.23
Split 2	128	1.63	30	0.39	76.07
Split 3	256	0.82	36	0.13	84.15
Split 4	512	0.42	42	0.07	83.33
Split 5	512	0.12	60	0.03	75.00

with our compressed configuration using optimized channel counts c. The compression achieves significant reduction in data transfer requirements, with gains ranging from 62.23% to 84.15%. Notably, all splits show substantial improvement, with *Split 3* achieving the highest compression ratio, reducing the data transfer size from 0.82 MB to just 0.13 MB. These compressions are achieved while maintaining the model performance above our threshold, as shown in Fig. 5.

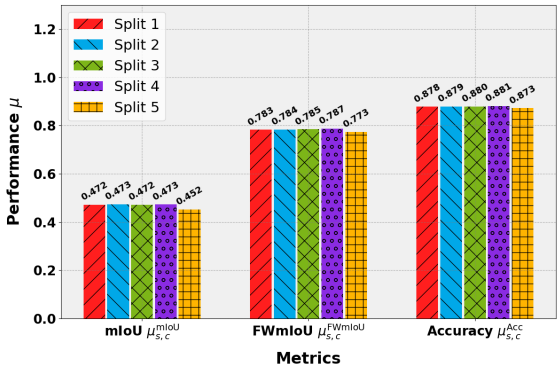


Fig. 6: Evaluation metrics μ comparison across different LiDAR semantic segmentation model split points, demonstrating the impact of split location on model performance.

Performance of different split configurations with compression c, evaluated across multiple metrics μ on the test set, is shown in Fig. 6. Results align with validation patterns, where *Split 1*-*Split 4* demonstrate consistent performance while *Split 5* shows slight degradation. *Split 1*-*Split 4* maintain strong performance with only 2.27% to 2.48% loss compared to the uncompressed model, while *Split 5* shows a more notable 6.61% performance loss. This trend is consistent across all metrics, demonstrating robust performance through *Split 1*-*Split 4*, with *Split 5* offering additional compression when higher compression ratios are prioritized.

Simulations with parameters from Table I across different strategies (*Edge Server Only*, *CAV Only*, and *Split 1*-*Split 5*) yield latencies shown in Figure 7. We exclude download times as they are typically several magnitudes lower than other latencies (ns versus ms). Later splits require more CAV computation but less offloading time, consistent with Table II. *Split 1* shows the highest total latency $t_{z,s}$ at 20.44ms, while *Split 3* achieves the lowest at 8.56ms, suggesting up to 50.46% latency improvement compared to baselines. *Split 1*'s

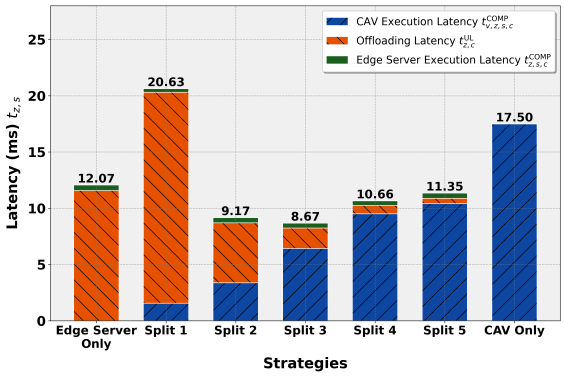


Fig. 7: Comparison of Processing Latencies Across Different Strategies.

offloading latency could be reduced by adjusting channel size c through the τ parameter in Algorithm 1, though this would reduce performance μ. The following section analyzes how key simulation parameters affect different strategies.

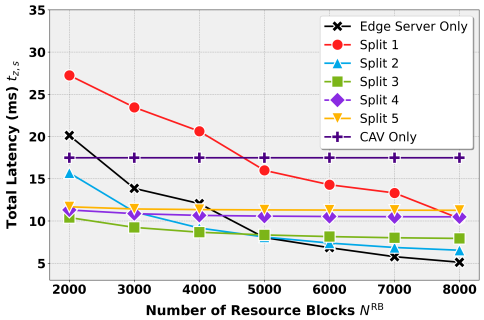
E. Split Computing Evaluation

In this section, we evaluate split computing for the sensor fusion model across varying resource blocks N^{RB} and two CAV computational capability configurations: $\eta_v = 1.5\times10^{12}$ and $\eta_v = 2.5 \times 10^{12}$.

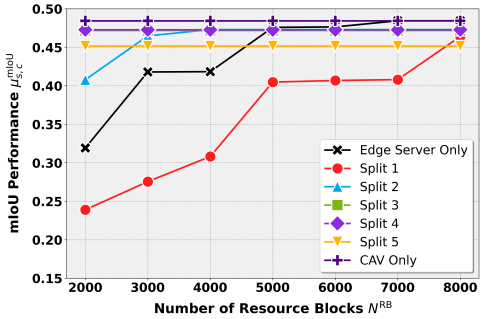
Fig. 8 shows system behavior with CAV computational power $\eta_v = 1.5\times10^{12}$ across resource blocks N^{RB} from 2000 to 8000. *Split 1* exhibits highest sensitivity to N^{RB} changes, aligning with Table III and Fig. 7, showing largest data sizes and offloading latency $t^{\text{UL}}_{z,c}$. *Split 2* and *Split 3* show moderate sensitivity, while *Split 4* and *Split 5* remain relatively stable. Split computing reduces latency by up to 62.64%, though the *Edge Server Only* approach outperforms all splits after resource blocks $N^{\text{RB}} = 6000$.

The relationship between mIoU performance and dropped tasks is evident in Fig. 8b and Fig. 8c, respectively. As the number of dropped tasks decreases, mIoU performance improves, with this effect being most pronounced in *Split 1* due to its higher total latency $t_{z,s}$.

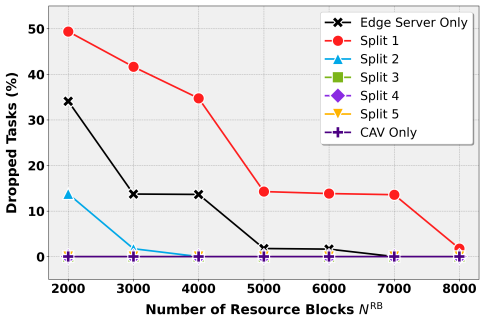
Fig. 9 shows similar patterns to Fig. 8, but with increased CAV computational capability $\eta_v = 2.5 \times 10^{12}$. While the trends remain consistent, *CAV Only* total latency $t_{z,s}$ improves significantly due to higher computational capability. *Split 3*, *Split 4*, and *Split 5* show the most pronounced improvements, as they have the highest CAV computational latency $C^{\text{COMP}}_{v,z,s,c}$ as shown in Fig. 7. This enhanced computational capability enables *Split 3*, *Split 4*, and *Split 5* to outperform *Edge Server Only* at $N^{\text{RB}} = 6000$, while *Split 2* and *Split 3* still achieve better performance at $N^{\text{RB}} = 7000$ and comparable results at $N^{\text{RB}} = 8000$. Furthermore, at lower number of resource blocks N^{RB}, split computing outperforms baselines by up to 62.74%. For conciseness, Fig. 9 presents only latency results, as the trends for model performance and task drop rate remain largely similar to those illustrated in Fig. 8.



(a) Total latency $t_{z,s}$ analysis.



(b) mIoU performance evaluation $\mu^{\text{mIoU}}_{s,c}$.



(c) Task drop rate analysis.

Fig. 8: System behavior analysis across different resource block configurations with CAV computational capability $\eta_v = 1.5 \times 10^{12}$

VI. Conclusions

In this paper, we propose a novel approach to sensor fusion for LiDAR semantic segmentation using split computing. Our method leverages an encoder-decoder model architecture by utilizing LiDAR point cloud indices instead of transmitting other data, such as pooling indices or intermediate feature maps through skip connections, to the decoder. This approach achieves better model performance while reducing the data that needs to be offloaded to the edge server. Our evaluation demonstrates that the proposed split computing approach frequently outperforms *Edge Server Only* and *CAV Only* strategies, although its effectiveness is highly dependent on environmental conditions. Future work should focus on developing strategy selection algorithms to determine the

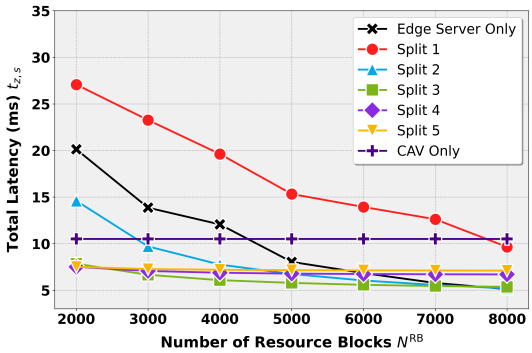


Fig. 9: Total latency analysis across different resource block configurations with CAV computational capability $\eta_v = 2.5 \times 10^{12}$.

appropriate split based on these conditions.

ACKNOWLEDGMENT

Icons used in this work were made by Tomas Knop, smashingstocks, rcherem and freepik from www.flaticon.com. The authors used artificial intelligence to assist with grammar, style, and language editing in this manuscript. All substantive content, analysis, and conclusions remain the author's original work.

REFERENCES

[1] Y. Lu, H. Ma, E. Smart, and H. Yu, "Real-time performance-focused localization techniques for autonomous vehicle: A review," *IEEE Transactions on Intelligent Transportation Systems*, vol. 23, no. 7, pp. 6082–6100, 2021.

[2] J. Janai, F. Güney, A. Behl, A. Geiger, *et al.*, "Computer vision for autonomous vehicles: Problems, datasets and state of the art," *Foundations and Trends® in Computer Graphics and Vision*, vol. 12, no. 1–3, pp. 1–308, 2020.

[3] A. Ndikumana, K. K. Nguyen, and M. Cheriet, "Age of processing-aware offloading decision for autonomous vehicles in 5g open ran environment," *IEEE Transactions on Mobile Computing*, 2023.

[4] M. Ahmed, S. Raza, M. A. Mirza, A. Aziz, M. A. Khan, W. U. Khan, J. Li, and Z. Han, "A survey on vehicular task offloading: Classification, issues, and challenges," *Journal of King Saud University-Computer and Information Sciences*, vol. 34, no. 7, pp. 4135–4162, 2022.

[5] H. Taslimasa, S. Dadkhah, E. C. P. Neto, P. Xiong, S. Ray, and A. A. Ghorbani, "Security issues in internet of vehicles (iov): A comprehensive survey," *Internet of Things*, vol. 22, p. 100809, 2023.

[6] J. Laconte, A. Kasmi, R. Aufrère, M. Vaidis, and R. Chapuis, "A survey of localization methods for autonomous vehicles in highway scenarios," *Sensors*, vol. 22, no. 1, p. 247, 2021.

[7] Y. Han, H. Zhang, H. Li, Y. Jin, C. Lang, and Y. Li, "Collaborative perception in autonomous driving: Methods, datasets, and challenges," *IEEE Intelligent Transportation Systems Magazine*, 2023.

[8] Y. Kang, J. Hauswald, C. Gao, A. Rovinski, T. Mudge, J. Mars, and L. Tang, "Neurosurgeon: Collaborative intelligence between the cloud and mobile edge," *ACM SIGARCH Computer Architecture News*, vol. 45, no. 1, pp. 615–629, 2017.

[9] J. Shao and J. Zhang, "Bottlenet++: An end-to-end approach for feature compression in device-edge co-inference systems," in *2020 IEEE International Conference on Communications Workshops (ICC Workshops)*, pp. 1–6, IEEE, 2020.

[10] Y. Matsubara, M. Levorato, and F. Restuccia, "Split computing and early exiting for deep learning applications: Survey and research challenges," *ACM Computing Surveys*, vol. 55, no. 5, pp. 1–30, 2022.

[11] W. Wang, J. Dai, Z. Chen, Z. Huang, Z. Li, X. Zhu, X. Hu, T. Lu, L. Lu, H. Li, *et al.*, "Internimage: Exploring large-scale vision foundation models with deformable convolutions," in *Proceedings of the IEEE/CVF conference on computer vision and pattern recognition*, pp. 14408–14419, 2023.

[12] Y. Zhang, A. Carballo, H. Yang, and K. Takeda, "Perception and sensing for autonomous vehicles under adverse weather conditions: A survey," *ISPRS Journal of Photogrammetry and Remote Sensing*, vol. 196, pp. 146–177, 2023.

[13] K. Huang, B. Shi, X. Li, X. Li, S. Huang, and Y. Li, "Multi-modal sensor fusion for auto driving perception: A survey," *arXiv preprint arXiv:2202.02703*, 2022.

[14] J. Plachy, Z. Becvar, E. C. Strinati, and N. di Pietro, "Dynamic allocation of computing and communication resources in multi-access edge computing for mobile users," *IEEE Transactions on Network and Service Management*, vol. 18, no. 2, pp. 2089–2106, 2021.

[15] 3GPP, "Nr; physical channels and modulation," *3rd Generation Partnership Project (3GPP), Technical Specification (TS) 38.211*, vol. 9, 2018.

[16] E. Björnson and L. Sanguinetti, "Rayleigh fading modeling and channel hardening for reconfigurable intelligent surfaces," *IEEE Wireless Communications Letters*, vol. 10, no. 4, pp. 830–834, 2020.

[17] I. Mohamed, "Path-loss estimation for wireless cellular networks using okumura/hata model," *Science Journal of Circuits, Systems and Signal Processing*, vol. 7, no. 1, pp. 20–27, 2018.

[18] A. Bozorgchenani, S. Maghsudi, D. Tarchi, and E. Hossain, "Computation offloading in heterogeneous vehicular edge networks: On-line and off-policy bandit solutions," *IEEE Transactions on Mobile Computing*, vol. 21, no. 12, pp. 4233–4248, 2021.

[19] J. Zhou, D. Tian, Z. Sheng, X. Duan, and X. Shen, "Distributed task offloading optimization with queueing dynamics in multiagent mobile-edge computing networks," *IEEE Internet of Things Journal*, vol. 8, no. 15, pp. 12311–12328, 2021.

[20] M. A. Rahman and Y. Wang, "Optimizing intersection-over-union in deep neural networks for image segmentation," in *International symposium on visual computing*, pp. 234–244, Springer, 2016.

[21] J. Li, H. Dai, H. Han, and Y. Ding, "Mseg3d: Multi-modal 3d semantic segmentation for autonomous driving," in *Proceedings of the IEEE/CVF conference on computer vision and pattern recognition*, pp. 21694–21704, 2023.

[22] H. Aljumaily, D. F. Laefer, D. Cuadra, and M. Velasco, "Point cloud voxel classification of aerial urban lidar using voxel attributes and random forest approach," *International Journal of Applied Earth Observation and Geoinformation*, vol. 118, p. 103208, 2023.

[23] X. Wang, K. Li, and A. Chehri, "Multi-sensor fusion technology for 3d object detection in autonomous driving: A review," *IEEE Transactions on Intelligent Transportation Systems*, 2023.

[24] Q. Tang, J. Liang, and F. Zhu, "A comparative review on multi-modal sensors fusion based on deep learning," *Signal Processing*, p. 109165, 2023.

[25] M. Tan, Z. Zhuang, S. Chen, R. Li, K. Jia, Q. Wang, and Y. Li, "Epmf: Efficient perception-aware multi-sensor fusion for 3d semantic segmentation," *IEEE Transactions on Pattern Analysis and Machine Intelligence*, 2024.

[26] H. Kumaraswamy, M. U. Kumari, B. R. Reddy, T. Baitha, *et al.*, "Implementation of object detection for autonomous vehicles by lidar and camera fusion," in *2024 IEEE International Conference on Interdisciplinary Approaches in Technology and Management for Social Innovation (IATMSI)*, vol. 2, pp. 1–6, IEEE, 2024.

[27] Z. Li, A. Zhou, J. Pu, and J. Yu, "Multi-modal neural feature fusion for automatic driving through perception-aware path planning," *IEEE Access*, vol. 9, pp. 142782–142794, 2021.

[28] P. Bholowalia and A. Kumar, "Ebk-means: A clustering technique based on elbow method and k-means in wsn," *International Journal of Computer Applications*, vol. 105, no. 9, 2014.

[29] K. Genova, X. Yin, A. Kundu, C. Pantofaru, F. Cole, A. Sud, B. Brewington, B. Shucker, and T. Funkhouser, "Learning 3d semantic segmentation with only 2d image supervision," in *2021 International Conference on 3D Vision (3DV)*, pp. 361–372, IEEE, 2021.

[30] I. Hadj-Kacem, S. B. Jemaa, H. Braham, and A. M. Alam, "Sinr prediction in presence of correlated shadowing in cellular networks," *IEEE Transactions on Wireless Communications*, vol. 21, no. 10, pp. 8744–8756, 2022.