Cloud-Assisted 360-Degree 3D Perception for Autonomous Vehicles Using V2X Communication and Hybrid Computing

Faisal Hawlader, François Robinet, Gamal Elghazaly, and Raphaël Frank
Interdisciplinary Centre for Security, Reliability and Trust (SnT), University of Luxembourg
29 Avenue J.F Kennedy, L-1855 Luxembourg
firstname.lastname@uni.lu

***Abstract*—A key challenge for autonomous driving lies in maintaining real-time situational awareness, particularly in complex urban settings. This study introduces an innovative cloud-driven solution for 360-degree perception in autonomous vehicles using vehicle-to-everything (V2X) communications. Our approach utilizes transformer-based models to fuse multi-camera sensor data into a comprehensive bird's-eye view (BEV) representation, enabling accurate 3D object detection. By offloading computationally intensive tasks to the cloud, the system achieves scalable processing while reducing latency. In addition, techniques such as feature vector clipping, compression, and quantization are applied to optimize data transmission, ensuring real-time performance. Experimental results show an over 79.2% reduction in end-to-end delay compared to onboard-only computing. These experiments validate the integration of AI and V2X technology to enhance autonomous vehicle perception.**

***Index Terms*—Autonomous Vehicles; 360-Degree Perception; Cloud Computing; V2X Communications**

I. Introduction

The development of fully autonomous vehicles has driven significant progress in automotive industry [1, 2], with the potential to transform driving experience by enhancing operational efficiency and safety [3]. At the core of autonomous driving are perception systems that enable real-time detection of surrounding objects [4, 5]. Real-time perception is critical for navigating complex environments and supporting decision making in motion planning [6]. Industry leaders such as Tesla [7], BMW and Mercedes-Benz face considerable challenges in processing the extensive sensor data (e.g., cameras, radar, and LiDAR) required for 3D object detection [8, 9]. Recent research has focused on addressing the stringent latency and accuracy requirements associated with autonomous perception tasks [10]. Perception models such as BEVFormer [11] have demonstrated high detection accuracy [12]. However, their computational demands often exceed the capabilities of vehicle hardware [10], resulting in increased latency and power consumption [13]. For instance, an industrial report published by Ford Motor indicated that future vehicles may need to allocate up to 47% of their energy to onboard computing [2]. This emphasizes the need for more efficient processing strategies.

To address onboard computing challenges, researchers have proposed partitioning perception models [14, 15] and offloading computationally intensive layers to the cloud [16, 17]. While this strategy reduces onboard computing burden, it also introduces intermediate feature vector transmission latency [18], which is problematic for real-time detection with strict latency requirements [14]. To address the challenge of efficiently transmitting large feature vectors from onboard to cloud processing, post-training quantization [19] and clipping [20] can be applied to layer activation feature vectors. By significantly reducing bandwidth requirements and transmission latency, these techniques enable real-time processing within hybrid computing environments [20]. Further compression can minimize the size of the feature vector [10], improving efficiency. However, quantization, clipping, and compression can negatively impact detection quality due to data loss [20]. Finding an optimal trade-off between end-to-end delay and detection quality remains a critical area of research.

To explore these trade-offs, we propose a BEVFormer based hybrid computing strategy integrating cooperative perception for 360-degree 3D detection. Cooperative perception enables vehicles and infrastructure to share sensor data via Vehicle-to-Everything (V2X), overcoming limitations such as occlusions and sensor range constraints [21]. By exchanging Cooperative Perception Messages (CPMs), which include information about detected objects, this method extends perception beyond onboard sensors [22]. In our approach, the vehicle performs lightweight feature extraction locally while offloading intensive computations to the cloud, combining local and cloud processing to improve real-time performance. Experimental results demonstrate over a 79.2% reduction in end-to-end delay compared to onboard-only computing, highlighting the effectiveness of distributing perception tasks between the vehicle and the cloud. Our contributions include:

- **360-degree 3D Perception**: We benchmark onboard computing, where tasks are processed locally, and hybrid computing, where intensive tasks offloaded to the cloud. Detection results are encoded into CPMs in both cases.
- **Hybrid Computing**: We introduce dynamic feature clipping, compression, and precision adjustments to reduce offloading latency while preserving detection quality.
- **Real-world Testing**: The system is tested in real-world scenarios with V2X integration, achieving low end-to-end delay, making it suitable for real-time perception.

The remainder of this paper is organized as follows: Section II reviews the relevant literature. Section III details our proposed method, including the on-board and cloud components, as well as the test route and communication technologies used. Section IV presents our experimental results, focusing on trade-offs between latency, accuracy. Finally, Section V summarizes our findings and outlines potential future research directions.

II. Related Work

Cooperative perception [21] has gained significant attention as a method to enhance the situational awareness of autonomous vehicles [1]. By enabling vehicles to share sensor data and computational resources [23], these systems can significantly improve object detection [9] and prediction in complex environments [24]. Several recent studies have explored vehicle-to-cloud (V2C) communication to offload perception tasks to remote servers [8, 10]. This approach leverages the higher computational power of the cloud to complement onboard processing [2, 25]. Early works in the field, such as [26, 27], focused on transmitting raw data to the cloud. However, these approaches suffered from bandwidth limitations and high transmission latency [28, 29], making them unsuitable for real-time applications [30]. To address these challenges, [18, 20] introduced feature-level offloading [4], where intermediate feature vectors are transmitted instead of raw data [27], significantly reducing bandwidth usage [31]. However, the size of these feature vectors can still be prohibitively large [14, 32], especially when generated by deep neural networks like ResNet101 [33] or BEVFormer [11].

Recent studies, such as [14], have proposed various methods for compressing feature vectors [16], including quantization and lossy compression [16, 34]. Although these methods reduce transmission data size, they often result in a degradation of detection accuracy [13]. Our work builds on these efforts by introducing a dynamic clipping mechanism [20] that minimizes unnecessary feature data while retaining key information for accurate 3D object detection. We also evaluate the effectiveness of lossless compression in combination with different floating-point precisions to achieve a balance between latency, bandwidth, and accuracy.

Additionally, while previous work has focused primarily on static environments or simulations, our system is tested in real-world driving scenarios, utilizing V2X communication. This enables us to account for network jitter and variability, providing a more robust evaluation of real-time capabilities and performance of the Cooperative Perception System (CPS).

III. Methodology

In this section, we outline the methodology for evaluating the proposed lightweight 360-degree 3D perception system. Multi-view (6x) camera images are processed using BEVFormer [11], a 3D object detection framework for autonomous driving. The model outputs 3D bounding boxes with object positions, orientations, and sizes in a Bird's-eye view (BEV) space, making it well suited for perception. Results are then encoded into CPMs and broadcast to nearby vehicles or infrastructure via V2X, standardized by The European Telecommunications Standards Institute (ETSI) [22].

A. Test Scenario & Route

The tests were carried out on public roads in the Kirchberg area of Luxembourg, which offers various road layouts and traffic conditions. This environment allowed us to evaluate perception scenarios in highly realistic settings. Communications between vehicles and infrastructure are handled by the YoGoKo Y-Box module, which supports the ITS-G5 and C-V2X technologies. For more information on the test vehicle, sensors, hardware, and software stack, refer to [35]. For this work, we setup two distinct scenarios, as shown in Figure 1:

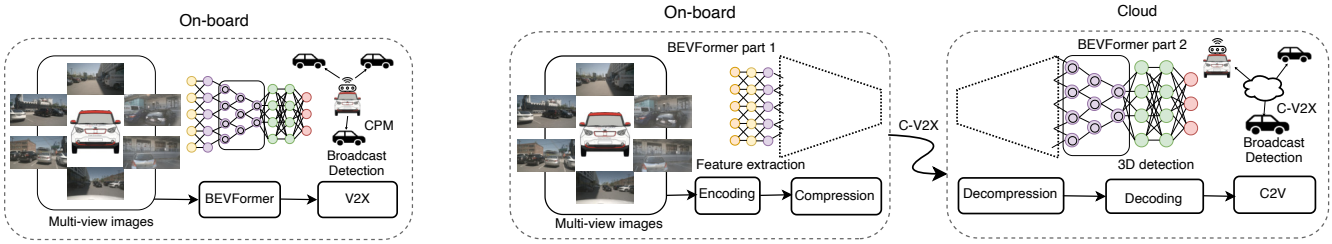
In the **Onboard computing scenario**, multi-view images are fed into the BEVFormer model, with all perception tasks performed locally. The detection results are then encoded into CPM and transmitted to nearby vehicle or infrastructure via ITS-G5. The experimental route spanned a distance of approximately 1.5 km[1]. Within this setup, we evaluated the transmission of CPMs to assess the reliability and performance of V2X communication in real world traffic. To ensure consistent measurements, we placed a stationary receiver at specific coordinates. This provided a fixed reference point for evaluating V2X communication quality as the transmitting vehicle moved along the designed test route.

In the **Hybrid computing scenario**, BEVFormer is split into two parts: Part-1 handles initial feature extraction with lightweight computations onboard, while Part-2 completes perception tasks in the cloud. Multi-view images are processed by BEVFormer Part-1 to extract feature vectors, which are then clipped, compressed, and sent to the cloud via C-V2X. In the cloud, BEVFormer Part-2 completes the remaining perception tasks, including 3D object detection. The detection results are encoded into CPMs and broadcast to nearby vehicles via C-V2X, enabling cooperative perception. The test route spans approximately 4km^2. We use the cellular mode of C-V2X to communicate with a cloud server located at the University of Luxembourg in the same area. Twelve commercial base stations, including 4G and 5G (non-standalone) sites, operate along the route on Low- (700 MHz) and Mid-band (3.6 GHz) frequencies. The area provides an average download throughput of 57 Mbps for 4G and 115 Mbps for 5G, with upload speeds ranging from 25 to 35 Mbps for both. We use UDP for data offloading to the cloud, as it offers the lowest end-to-end delay for offloading sensor data [36].

B. Hardware Configuration & Detection Model

The hardware configurations used in this study are outlined in Table I. The onboard setup utilizes a Jetson Orin, selected for its low power consumption and processing capabilities in embedded perception tasks. The cloud platform employs multiple Tesla V100 GPU nodes, designed to handle computationally intensive tasks. For more details on GPU node configurations, we refer the reader to [25].

[1]**Local test route:** http://g-o.lu/3/GsHC
[2]**Cloud test route:** http://g-o.lu/3/96TS



(a) Onboard computing scenario

(b) Hybrid computing scenario

Fig. 1: Experimental scenarios: In the Onboard Computing scenario, the BEVFormer model runs locally, transmitting detection results as CPMs over ITS-G5. In the Hybrid Computing scenario, a compressed feature vector is sent via C-V2X to the cloud for intensive BEVFormer processing, with detection results broadcast to nearby vehicles.

Platform	**Hardware Configuration**
Local (≈30W)	NVIDIA Jetson Orin 2048 CUDA Cores, 131.4 TOPS (INT8) 8-core ARM Cortex-A78AE
Cloud (≈3000W)	2x Intel Xeon Skylake CPUs (56 cores total) 4x NVIDIA Tesla V100 GPUs (16 GB each) 20480 CUDA Cores, 500 TFLOPS (FP16)

TABLE I: Hardware setups for the onboard and cloud computing platforms described in Section III-A

.

We use the BEVFormer with ResNet101 backbone [37], initialized from the FCOS3D checkpoint [12]. BEVFormer performs multi-view 3D object detection in several stages. It begins with a ResNet101 backbone that extracts features from multi-view camera inputs. These features are transformed into a BEV representation using a view transformer that fuses spatial information from different perspectives. The BEV representation is refined by a BEV encoder, which applies self-attention to improve the detection accuracy. Finally, the detection head outputs 3D bounding boxes, predicting object positions, orientations, and sizes. For more details on the BEVFormer, we refer the reader to [11]. In our hybrid setup, we split BEVFormer after the initial backbone layers, performing feature extraction onboard. The remaining backbone layers, view transformation, BEV encoding, and 3D detection are offloaded to the cloud for efficient processing.

C. Dataset and Evaluation

For this study, we use the **nuScenes** dataset [38], a large-scale dataset specifically created for autonomous driving research. The dataset consists of images resolution of 1600x900 from six cameras, five radars, and one LiDAR, providing full 360-degree coverage, perfectly aligning with our objective of achieving comprehensive 3D detection. The dataset also includes detailed annotations for 3D object detection, tracking, and segmentation across various classes such as vehicles, pedestrians, and cyclists. In this work, we use a BEVFormer model pre-trained on the NuScenes dataset without performing any additional training. We use the nuScenes dataset solely to evaluate the performance of the model after post-quantization, clipping, and compression. The evaluation aims to assess potential detection accuracy loss resulting from different quantization levels, as well as the effects of clipping and compression. To evaluate performance across all classes, we use the **NuScenes Detection Score (NDS)**, which offers a comprehensive assessment of detection tasks. The NDS is calculated using the following formula [11]:

$$\text{NDS} = \frac{1}{10}\left(5\text{mAP} + \sum_{\text{mTP}\in\mathbb{TP}} (1 - \min(1, \text{mTP}))\right)$$

This score integrates various aspects of model performance, including mean average precision (mAP) and mean true positive (mTP), providing a robust evaluation.

D. Lightweight Features Offloading: Hybrid Computing

In hybrid computing, multi-view images captured around the vehicle are first processed by the initial backbone layers onboard to extract features. These features are then dynamically clipped and compressed to reduce their size and optimize bandwidth before transmission to the cloud. In the cloud, the remaining backbone processing, view transformation, BEV encoding, and 3D object detection are completed. This division reduces the computational load on the vehicle, while resource intensive tasks are handled in the cloud. The complete processing workflow is detailed in Algorithm 1.

Feature Vector Extraction: The backbone network B on the vehicle processes the input images X and extracts feature vectors $F(x) \in \mathbb{R}^{C\times H\times W}$ at the split layer L_{split}, where C is the number of channels, and H and W represent the spatial dimensions. The selection of L_{split} balances onboard processing and reduces data offloaded to the cloud.

Feature Vector Clipping & Compression: The extracted features $F(x)$ at the split layer L_{split} undergo a clipping process with lower bound L_p and upper bound U_p.

$$F_{clip}(x) = \max\left(L_p, \min\left(F(x), U_p\right)\right)$$

The system dynamically chooses the bounds (L_p, U_p) in order to clip activations to their p_{lower} and p_{upper} percentiles. Based on our experimental analysis, we set p_{lower} and p_{upper} to 10th and 90th percentiles, respectively, to balance bandwidth reduction and detection accuracy. By clipping values outside this range, the system removes outliers that do not contribute

significantly to detection. This clipping function lowers the entropy of the activation distribution, making compression more efficient. The clipped features $F_{\text{clip}}(x)$ are compressed using a lossless zlib compression [39], further minimizing the data size for efficient transmission to the cloud.

Offloading to Cloud: After clipping and compression, the features are offloaded to the cloud via C-V2X while the vehicle follows the test route. Compressed $F_{\text{comp}}(x)$ are transmitted at regular intervals, defined by the offload frequency Δt, which adjusts according to network conditions to minimize latency [40]. The system monitors latency between transmission and cloud acknowledgment. If latency increases, indicating network congestion or slower speeds, the frequency is adjusted to prevent delays [41]. In the cloud, activations are decompressed and processed using a multi-GPU setup (Section III-A), accelerating detection compared to on-board hardware. The results are encoded into CPMs and transmitted to nearby vehicles and infrastructure, enhancing situational awareness through cooperative perception.

Algorithm 1 Lightweight Features Offloading: Hybrid

Given:
X: Input image, B: Backbone network, L_{split}: Split layer, $p_{\text{lower}}, p_{\text{upper}}$: Clipping percentiles, Δt: frequency

Onboard Hardware:
Initialize input X, backbone network B, and Layers L
while vehicle is driving **do**
  for each layer L in B **do**
    if $L = L_{\text{split}}$ **then**
      Compute clipping thresholds p_{lower}, p_{upper}:
        $L_p \leftarrow \text{Percentile}(F(x), p_{\text{lower}})$
        $U_p \leftarrow \text{Percentile}(F(x), p_{\text{upper}})$
      $F_{\text{clip}}(x) \leftarrow \max(L_p, \min(F(x), U_p))$
      $F_{\text{comp}}(x) \leftarrow \text{Compress}(F_{\text{clip}})$
      Transmit $F_{\text{comp}}(x)$ to the Cloud
  Wait Δt
On Cloud:
while receiving data from vehicle **do**
  $F_{\text{recv}}(x) \leftarrow \text{Decompress}(F_{\text{comp}}(x))$
  DetResults $\leftarrow$ BEVFormer.head$(F_{\text{recv}}(x))$
  CPM $\leftarrow$ Encode(DetResults)
  Broadcast CPM
Return: CPM containing 3D object detection results

E. CPM Encoding

The CPM encoding process packages detected objects and environmental data into a standardized format defined by ETSI [22], ensuring interoperability in CPS. As illustrated in Fig. 2, the message structure includes several containers such as the ITS-PDU header, management, and sensor information containers, which store the reference position, sensor ID, and metadata. The encoding process is managed by the YoGoKo Y-Box module [35], which supports both ITS-G5 and C-V2X technologies, enabling seamless V2X communication.

IV. Experiments and Results

In this section, we evaluate the performance of our proposed perception system designed for autonomous driving.

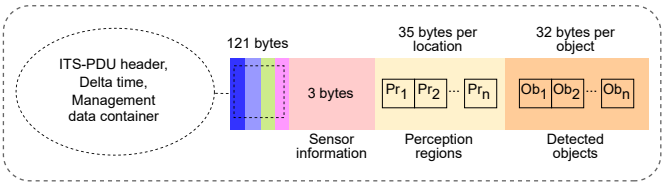


Fig. 2: A basic overview of different containers included in the CPM message format as defined by the ETSI standard [22].

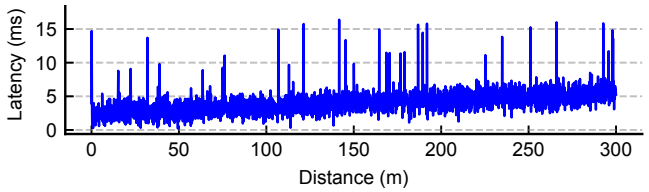


Fig. 3: CPM transmission latency versus distance between a moving vehicle (25 km/h Avg.) and a stationary receiver at fixed coordinates (longitude: 6.161993, latitude: 49.626478).

The system generates and transmits CPMs while integrating 3D object detection to enhance environmental awareness. By offloading sensor data processing to the cloud and enabling communication with nearby vehicles, the system supports cooperative perception, allowing more accurate and timely decision making in complex driving scenarios. Repeating each experiment five times for statistical robustness, we evaluated its effectiveness using the scenarios detailed in Section III-A.

A. Onboard computing and CPM transmission

To establish a baseline, we performed inference tests on the onboard platform, as detailed in Section III-B, using the NuScenes dataset described in III-C. These tests yielded an average inference time of 673 ms for the default model prior to optimization. This far exceeds the typical latency threshold for real-time perception in autonomous driving (e.g., less than 100 ms) [10]. Although the model achieved an NDS of 0.52, onboard processing consumed over 65% (±4) of the hardware resources, as monitored through the nvidia-smi GPU tracking. These results highlight the limitations of onboard computing, particularly regarding resource usage and time constraints.

Model optimization using TensorRT: To address resource usage and time constraints, we employ TensorRT for model optimization [5]. TensorRT enhances performance by applying precision calibration (FP32, FP16, or FP8) and optimizing the computational complexity of the model. The experimental results in Table II show how TensorRT optimization significantly reduces inference times without major degradation in detection. By reducing the precision from FP32 to FP16 and FP8, we observe substantial improvements in inference.

For instance, inference time drops from 486 ms (FP32) to 194 ms (FP8), with only a marginal decrease in NDS, from 0.52 to 0.51, indicating that the performance in terms of detection is largely preserved even with reduced precision.

These results align with previous studies, which demonstrated that quantization effectively maintains high detection accuracy while significantly reducing inference time [24, 30].

Quantization	Inference (ms)	NDS	CPM (ms)	End-to-end delay (ms)
FP32	486	0.52	5.9 ($\pm$1.8)	491.9 ($\pm$2.7)
FP16	257	0.52	5.7 ($\pm$1.7)	262.7 ($\pm$2.3)
FP8	194	0.51	4.9 ($\pm$1.6)	198.9 ($\pm$2.3)

TABLE II: Performance metrics for the BEVFormer model with ResNet101 backbone, evaluated using TensorRT optimization at different quantization levels (FP32, FP16, FP8) on the onboard vehicle platform, as detailed in Section III-B. The table includes inference time and CPM transmission latency, which together form the end-to-end delay. Standard deviations ($\pm$) are provided to reflect measurement variability.

CPM Transmission & V2X Communication: Upon detecting surrounding objects, the 3D detection results are encoded into a CPM and broadcast to nearby vehicles via V2X direct communication. The objective of this CPM transmission test was to measure the end-to-end latency in a cooperative perception scenario. Key parameters for this evaluation are summarized in Table III.

Parameter Name	Value
Transmission Power (Tx)	23 dBm
Energy threshold	-85 dBm
Channel bandwidth / carrier frequency	10 MHz / 5.9 GHz
Radio Configuration	Single Channel (CCH)
Data rate	7 Mbps
Number of CPM Transmitted / loss ratio	6000 / 0.09

TABLE III: Important network parameters for V2X.

A static receiver node was placed at fixed coordinates, as described in the caption of Figure 3. The transmitting node, located in the vehicle, as detailed in Section III-A. The results of these tests, shown in Figure 3, illustrate how CPMs transmission latency varies with the distance between the two communicating nodes. The results indicate a slight linear increase in transmission latency as the distance grows, likely due to propagation delays and intermittent network congestion. The average transmission latency was 4.10 ms, with a maximum of 18.41 ms and a standard deviation of 1.61 ms. Notably, packet loss increased significantly when the distance exceeded 300 m, highlighting sensitivity to longer distances. These observations are consistent with previous simulation-based research [30].

Although TensorRT optimizations and model quantization have significantly reduced end-to-end delay, as shown in Table II, the system still falls short of the 100 ms target required for real-time perception. For example, quantization to FP8 results in an end-to-end delay of 198.9ms, which is almost double the desired threshold. Consequently, the end-to-end delay (onboard inference plus transmission) restricts CPM transmission rates to below 5Hz, highlighting the need for more efficient data processing to achieve real-time perception.

B. Cloud Computing and Lightweight Features Sharing

Offloading intensive perception tasks to the cloud, while keeping lighter tasks onboard, reduces local computation but adds transmission latency. Techniques like post-training quantization, compression, and clipping help minimize bandwidth usage and transmission time. This section explores the feasibility of lightweight feature sharing over networks, focusing on split layer selection, accuracy retention, and end-to-end delay.

Layer Partitioning and Feature Extraction: Determining the optimal partition layer in cloud processing is crucial, as it affects both the onboard feature extraction time and the size of transmitted features. In BEVFormer, partitioning earlier backbone layers (e.g., layer 1) minimizes onboard computation but requires transmitting larger feature vectors to the cloud. Conversely, deeper partitioning reduces feature vector size but increases onboard inference time. Figure 4 illustrates the trade-off between inference latency and feature vector size between split points and quantization levels.

- **Feature extraction time**: As more backbone layers are executed locally, feature extraction time increases, significantly impacting the real time feasibility. A split at Layer 5 in FP32 yields a latency of 55 ms, acceptable for real-time use, but deeper splits quickly exceed the 100 ms threshold (e.g., 115 ms at Layer 10). Lower precisions like FP16 (20 ms at Layer 5) and FP8 (9 ms at Layer 5) reduce latency, but the benefits diminish with deeper layers, where even FP8 exceeds 140 ms by Layer 30. This highlights the need for optimal split points and further optimizations to meet real-time constraints.
- **Feature Vector Size:** Shallow splits generate large feature vectors, making real-time transmission challenging. For example, FP32 at Layer 1 produces 52 MB, requiring a throughput of 520 Mbps at 10 Hz, far exceeding typical V2X bandwidth. Even FP16 (43 MB, 430 Mbps) and FP8 (31 MB, 310 Mbps) remain too large. Deeper splits, however, reduce vector sizes: FP32 at Layer 25 requires 5 MB (50 Mbps), and FP8 only 3 MB (30 Mbps), which are more suitable for real-time transmission. Nevertheless, deeper splits increase onboard processing time, requiring a trade-off between transmission latency and feature size.

To reduce activation sizes beyond what quantization can offer, we also rely on dynamic clipping and compression.

Feature Transmission and Compression: Dynamic clipping and zlib compression were applied to reduce feature vector sizes and transmission latency across quantization levels. These techniques reduced feature sizes by approximately 97% for FP32, 90% for FP16, and 80% for FP8, significantly improving transmission efficiency across all layers. The larger reductions for FP32 are expected, as higher precision data contains more entropy, making it more compressible compared to FP16 and FP8. We focus on layers 1 to 5 because deeper splits yielded minimal improvements in transmission latency while increasing feature extraction time. As shown in Figure 5, FP8 exhibited the lowest latency, with medians of 52 ms at

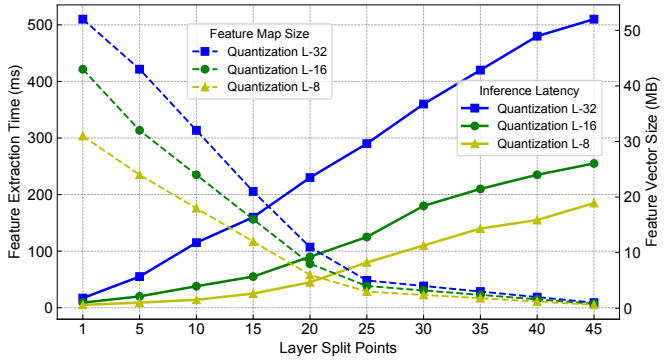


Fig. 4: Feature vector size and extraction time vs. split depth. Solid lines show feature extraction time (left y-axis), while dashed lines indicate size (right y-axis). Lower quantization (FP16, FP8) reduce both extraction time and feature size.

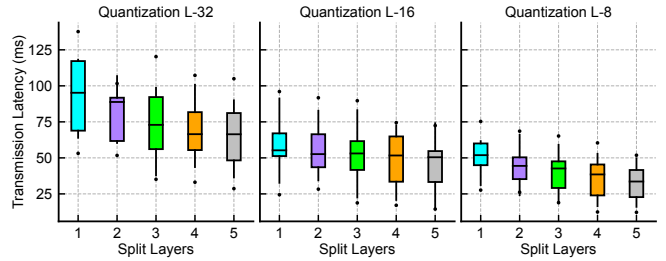


Fig. 5: Transmission latency of feature vectors from vehicle to cloud across five split layers for FP32, FP16, and FP8 over a 5G network using C-V2X. FP8 demonstrates the lowest and most stable latency, suitable for real-time transmission. FP32 exhibits the highest latency and variability, especially at earlier split layers due to larger feature size.

Layer 1 and 35 ms at Layer 5, meeting the threshold required for real-time perception systems.

FP16 provided a balanced latency, with medians of 67 ms at Layer 1 and 45 ms at Layer 5, making it suitable for applications that can tolerate moderate delays. In contrast, FP32 exhibited the highest latency, reaching 90 ms at Layer 1 and 70 ms at Layer 5, and was more sensitive to network jitter, as indicated by larger outliers. Despite providing higher precision, the elevated latency of FP32 renders it impractical for time-sensitive decisions without further optimization. FP8, with its minimal jitter and consistently low latency, emerges as the most viable option for cloud-based 3D object detection. As network demands increase with the adoption of 5G and beyond, the scalability of this approach, particularly with FP8, position it well for future real-time perception systems. While trade-offs between quantization levels exist, FP8 is ideal for latency critical environments, whereas FP16 and FP32 are more suitable for scenarios that prioritize higher data quality.

End-to-end Delay: The results in Table IV summarize the end-to-end delay and the impact of split layers and quantization levels. For FP32 quantization at split Layer 1, the total end-to-end delay is 128.7 ms, with local processing time (backbone and compression) contributing 27.9 ms, and transmission latency (V2C and C2V) adding 77.4 ms. In contrast, FP8 at the same split layer shows a significantly reduced total delay of 73.8 ms, mainly due to the lower local processing time of 13.3 ms and a reduced transmission latency of 43.4 ms. Lower quantization levels, such as FP8, reduce both computational burden and transmission latency, though they introduce a slight trade-off in accuracy, with the NDS for FP8 at split Layer 1 being 0.50 compared to 0.52 for FP32. As the split Layer deepens (e.g., Layer 5), onboard processing time increases due to the more complex feature extraction. For FP32, the total delay at split Layer 5 increases to 137.7 ms, with 60.5 ms for local processing time and 62.1 ms for transmission latency. In comparison, FP8 achieves a lower end-to-end delay of 61.9 ms at the same split Layer, primarily due to its reduced local processing time of 12.7 ms and transmission latency of 36.3 ms. This reduction in delay for FP8 comes with only a slight decrease in accuracy, as the NDS drops marginally to 0.45. Cloud processing times (decompression and head) remain low across all quantization levels due to the powerful cloud hardware, with decompression times ranging from 1.4 to 2.6 ms, depending on the quantization level. This consistent cloud performance ensures that most of the delay comes from local processing time and transmission latency, highlighting the importance of selecting the optimal split point and quantization level for real-time systems. Transmission latencies exhibit greater variability compared to local or cloud latencies, with the largest deviations observed at split layer 1 for FP32 ($\pm$4.00 ms for V2C) and FP8 ($\pm$2.80 ms for V2C), reflecting the impact of fluctuating network conditions on the transmission process. Although lower quantization levels like FP8 reduce total delay, they introduce a slight degradation in accuracy. Optimizing split layer, quantization, compression, and clipping based on real-time constraints and network bandwidth enhances performance and reliability in cloud-based cooperative perception.

End-to-end Delay vs. Accuracy Trade-off: Figure 6 illustrates the trade-off between end-to-end delay and NDS across different split layers and quantization levels. This suggests that while FP32 maintains accuracy, it is highly sensitive to the increased computational load of deeper layers, making it less practical for real-time systems with strict latency constraints. Conversely, FP16 and FP8 provide more favorable trade-offs. FP16 at Layer 5 reduces latency to 77.2 ms, with a modest NDS of 0.45. FP8 offers the lowest delay, achieving 61.9 ms at Layer 5 while maintaining an acceptable NDS of 0.43. Smaller feature vectors also reduce the risk of network-induced latency and packet loss, enhancing robustness in real-world deployments. Intermediate splits, such as Layer 3 with FP16, provide a balanced trade-off between end-to-end delay and accuracy. Layer 3 with FP16 achieves a delay of 88.7 ms and an NDS of 0.47, making it a practical solution for perception systems that require both timely responses and reasonably accurate detection, a visual demonstration of the result is shown in Figure 7. In scenarios where accuracy is the primary

Q Level	Split Layer	Onboard Processing Time (ms) Backbone	Onboard Processing Time (ms) Compression	Transmission latency (ms) V2C	Transmission latency (ms) C2V	Cloud Processing Time (ms) Decompression	Cloud Processing Time (ms) Head	End-to-end Delay (ms)	NDS	Bandwidth Usage (Mbps)
32	1	17.2 (±2.10)	10.7 (±1.85)	65.8 (±4.00)	11.6 (±1.15)	2.6 (±0.60)	20.8 (±1.35)	128.7 (±4.20)	0.52	10.5
	2	22.3 (±1.75)	8.6 (±1.55)	58.0 (±3.90)	9.8 (±0.95)	2.4 (±0.55)	18.4 (±1.25)	119.6 (±3.90)	0.50	8.4
	3	30.5 (±1.90)	7.3 (±1.50)	48.9 (±3.75)	8.4 (±0.85)	2.5 (±0.50)	15.9 (±1.30)	113.5 (±3.80)	0.48	6.8
	4	39.8 (±1.65)	6.4 (±1.30)	54.5 (±3.50)	7.0 (±0.75)	2.3 (±0.45)	14.7 (±1.10)	124.7 (±3.60)	0.47	5.9
	5	55.4 (±1.45)	5.1 (±1.10)	56.3 (±3.20)	5.8 (±0.70)	2.5 (±0.40)	12.6 (±0.95)	137.7 (±3.40)	0.46	5.4
16	1	9.3 (±1.70)	9.1 (±1.50)	57.6 (±3.10)	12.7 (±0.95)	2.1 (±0.50)	18.2 (±1.30)	109.0 (±3.90)	0.51	9.0
	2	11.7 (±1.50)	7.3 (±1.30)	39.3 (±2.95)	7.6 (±0.85)	2.2 (±0.45)	16.6 (±1.20)	84.7 (±3.70)	0.49	6.6
	3	15.3 (±1.35)	6.2 (±1.20)	44.1 (±2.80)	6.6 (±0.80)	2.1 (±0.40)	14.3 (±1.05)	88.7 (±3.50)	0.47	5.6
	4	18.5 (±1.25)	5.2 (±1.05)	42.3 (±2.65)	8.6 (±0.75)	2.0 (±0.35)	13.4 (±0.95)	90.0 (±3.25)	0.46	4.6
	5	20.4 (±1.10)	4.3 (±0.95)	31.2 (±2.50)	7.1 (±0.70)	2.0 (±0.30)	12.2 (±0.85)	77.2 (±3.05)	0.45	4.3
8	1	5.1 (±1.45)	8.2 (±1.45)	33.6 (±2.80)	9.8 (±0.90)	1.6 (±0.50)	15.5 (±1.25)	73.8 (±3.90)	0.47	8.4
	2	6.2 (±1.25)	6.7 (±1.30)	40.4 (±2.60)	8.1 (±0.85)	1.6 (±0.45)	14.1 (±1.15)	77.1 (±3.60)	0.46	6.9
	3	7.3 (±1.10)	5.7 (±1.10)	44.3 (±2.40)	6.3 (±0.80)	1.5 (±0.40)	12.6 (±1.00)	77.7 (±3.50)	0.44	5.5
	4	8.4 (±1.05)	4.7 (±1.00)	33.4 (±2.20)	5.6 (±0.75)	1.5 (±0.35)	11.9 (±0.90)	65.5 (±3.25)	0.43	4.7
	5	9.1 (±0.90)	3.6 (±0.90)	29.3 (±2.00)	7.0 (±0.70)	1.4 (±0.30)	11.0 (±0.85)	61.9 (±3.00)	0.43	4.1

TABLE IV: Performance metric for various split layers and quantization levels, including onboard processing time, V2C and C2V transmission latency, cloud processing time, and total end-to-end delay. The standard deviations reflect variability. The last column shows bandwidth utilization for offloading feature vectors from vehicle to cloud.

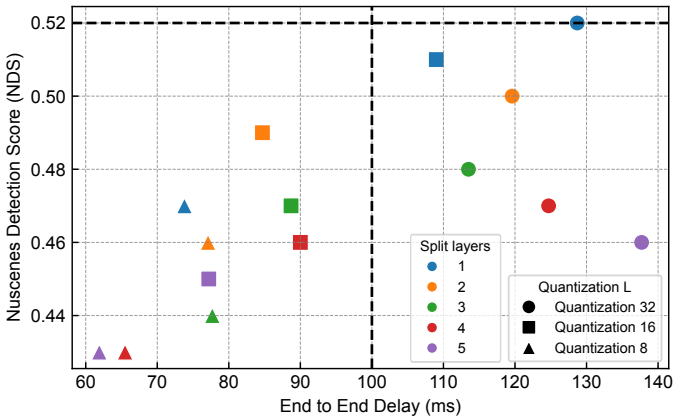


Fig. 6: End-to-end delay vs. detection accuracy (NDS) across split layers and quantization levels. Earlier layers result in higher accuracy but increased delay, while intermediate layers provide a balance between delay and accuracy.

concern, FP32 at shallower layers remains the best choice, although it comes at the expense of higher delay.

V. Conclusion and Future Work

This study evaluates cloud-based 3D object detection using a BEVFormer based model integrated with V2X communication. We propose a hybrid computing strategy that leverages cooperative perception for 360-degree detection. The approach offloads intensive computations to the cloud while maintaining lightweight feature extraction onboard, enabling real-time perception. Experimental results demonstrate that dynamic clipping, compression, and 5G-enabled C-V2X transmission significantly optimize latency and bandwidth utilization. For instance, offloading FP32 feature vectors at 10Hz from Layer 1 reduced bandwidth usage by over 95%, from 520 Mbps to 10.5 Mbps. We also investigate the trade-offs between end-to-end delay and detection quality across various split layers and quantization levels. Additionally, TensorRT optimizations were applied to further enhance inference speed. Our results demonstrate that while FP32 offers the highest accuracy, its substantial end-to-end delay renders it impractical for real-time applications. In contrast, FP8 achieves significantly lower latency with reasonable accuracy, making it suitable for latency-sensitive scenarios. FP16 provides a balanced trade-off between accuracy and latency, fitting applications that require both timely responses and adequate detection performance. This study underscores the importance of selecting appropriate split layers and quantization levels based on operational requirements. Shallower splits with FP32 are optimal for accuracy-focused tasks, whereas deeper splits with FP8 cater to applications with strict latency constraints.

One promising direction for future work is to refine the quantization through the exploration of advanced mixed-precision strategies. This involves dynamically adjusting the layers between FP32, FP16, and FP8 based on real-time requirements. Such an approach could further improve both latency and detection quality compared to the static quantization employed in this study. Additionally, evaluating the system with emerging wireless technologies, such as 6G or satellite-based communication, could reveal significant performance enhancements, especially in non-urban or remote areas.

Acknowledgments

This work is supported by the Fonds National de la Recherche of Luxembourg (FNR), under AFR grant agreement No 17020780 and project acronym *ACDC*.

References

[1] S. Sonko, E. A. Etukudoh, K. I. Ibekwe, V. I. Ilojianya, and C. D. Daudu, "A comprehensive review of embedded systems in autonomous vehicles: Trends, challenges, and future directions," *World Journal of Advanced Research and Reviews*, vol. 21, no. 1, pp. 2009–2020, 2024.

[2] J. H. Gawron, G. A. Keoleian, R. D. De Kleine, T. J. Wallington, and H. C. Kim, "Life cycle assessment of connected and automated vehicles: sensing and computing subsystem and vehicle level effects," *Environmental science & technology*, vol. 52, no. 5, p. 3249, 2018.

[3] V. Bhardwaj, "Ai-enabled autonomous driving: Enhancing safety and efficiency through predictive analytics," *International Journal of Scientific Research and Management (IJSRM)*, vol. 12, no. 02, p. 1076, 2024.

[4] J.-S. Lee and T. Ebrahimi, "Perceptual video compression: A survey," *IEEE Journal of selected topics in signal processing*, vol. 6, no. 6, 2012.

[5] T. Pham, B.-J. Maghoumi, J. Truong, and M. Park, "Nvautonet: Fast and accurate 360° 3d visual perception for self driving," in *2024 IEEE/CVF Winter Conference on Applications of Computer Vision (WACV)*, 2024.

[6] S.-W. Kim, B. Qin, Z. J. Chong, X. Shen, W. Liu, M. H. Ang, E. Frazzoli, and D. Rus, "Multivehicle cooperative driving using cooperative

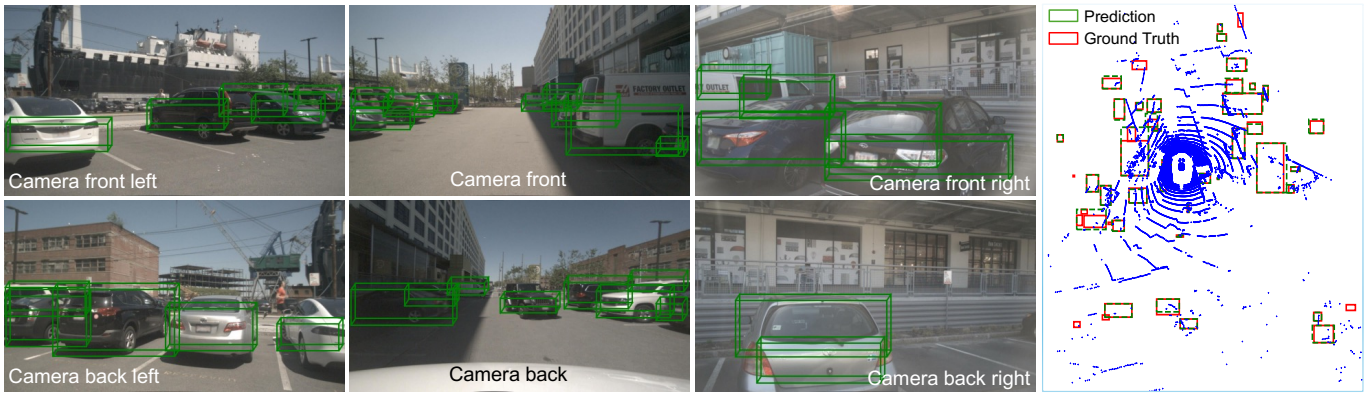


Fig. 7: Qualitative results at split layer 3 with FP16 on the nuScenes validation set (SceneID: n008-2018-05-21-11-06-59-0400), an NDS score of 0.47. We show 3D bounding box predictions in multi-view (6x) camera images and the 360-degree BEV.

perception: Design and experimental validation," *IEEE Transactions on Intelligent Transportation Systems*, vol. 16, no. 2, pp. 663–680, 2015.

[7] E. Marti, M. A. De Miguel, F. Garcia, and J. Perez, "A review of sensor technologies for perception in automated driving," *IEEE Intelligent Transportation Systems Magazine*, vol. 11, no. 4, pp. 94–108, 2019.

[8] A. Yaqoob, T. Bi, and G.-M. Muntean, "A survey on adaptive 360 video streaming: Solutions, challenges and opportunities," *IEEE Communications Surveys & Tutorials*, vol. 22, no. 4, pp. 2801–2838, 2020.

[9] Z. Wang, C. Li, and X. Yang, "Distillbev: Boosting multi-camera 3d object detection with cross-modal knowledge distillation," in *Proceedings of the IEEE/CVF International Conference on Computer Vision*, 2023.

[10] F. Hawlader, F. Robinet, and R. Frank, "Leveraging the edge and cloud for v2x-based real-time object detection in autonomous driving," in *Computer Communications*, vol. 213, 2024, pp. 372–381.

[11] Z. Li, E. Wang, C. Sima, T. Lu, Y. Qiao, and J. Dai, "Bevformer: Learning bird's-eye-view representation from multi-camera images via spatiotemporal transformers," *arXiv preprint arXiv:2203.17270*, 2022.

[12] C. Yang, Y. Chen, H. Tian, G. Huang, H. Li, Y. Qiao, L. Lu, J. Zhou, and J. Dai, "Bevformer v2: Adapting modern image backbones to bird's-eye-view recognition via perspective supervision," *ArXiv*, 2022.

[13] S. Gyawali, S. Xu, Y. Qian, and R. Q. Hu, "Challenges and solutions for cellular based v2x communications," *IEEE Communications Surveys & Tutorials*, vol. 23, no. 1, pp. 222–255, 2020.

[14] R. Mehta and R. Shorey, "Deepsplit: Dynamic splitting of collaborative edge-cloud convolutional neural networks," in *International Conference on COMmunication Systems & NETworkS (COMSNETS)*. IEEE, 2020.

[15] F. Hawlader, F. Robinet, and R. Frank, "Poster: Lightweight features sharing for real-time object detection in cooperative driving," in *2023 IEEE Vehicular Networking Conference (VNC)*, 2023, pp. 159–160.

[16] R. A. Cohen and I. V. Choi, "Lightweight compression of neural network feature tensors for collaborative intelligence," in *IEEE International Conference on Multimedia and Expo (ICME)*. IEEE, 2020.

[17] H. Choi and I. V. Bajić, "Near-lossless deep feature compression for collaborative intelligence," in *2018 IEEE 20th International Workshop on Multimedia Signal Processing (MMSP)*. IEEE, 2018, pp. 1–6.

[18] L. Torrey and J. Shavlik, "Transfer learning," in *Handbook of research on machine learning applications and trends: algorithms, methods, and techniques*. IGI global, 2010, pp. 242–264.

[19] Z. Liu, Y. Wang, K. Han, W. Zhang, S. Ma, and W. Gao, "Post-training quantization for vision transformer," *Advances in Neural Information Processing Systems*, vol. 34, pp. 28 092–28 103, 2021.

[20] C. Liqun and H. Lei, "Clipping-based neural network post training quantization for object detection," in *IEEE International Conference on Control, Electronics and Computer Technology (ICCECT)*. IEEE, 2023.

[21] J. He, Z. Tang, X. Fu, S. Leng, F. Wu, K. Huang, J. Huang, J. Zhang, Y. Zhang, A. Radford *et al.*, "Cooperative connected autonomous vehicles (cav): research, applications and challenges," in *IEEE 27th International Conference on Network Protocols (ICNP)*. IEEE, 2019.

[22] ETSI, "V2. 1.1; intelligent transport system (its); vehicular communications; basic set of applications; collective perception service," *ETSI: Sophia Antipolis, France*, 2023.

[23] R. Girshick, "Fast r-cnn," in *Proceedings of the IEEE international conference on computer vision*, 2015, pp. 1440–1448.

[24] W. Feng, S. Lin, N. Zhang, G. Wang, B. Ai, and L. Cai, "C-v2x based offloading strategy in multi-tier vehicular edge computing system," in *GLOBECOM 2022 - 2022 IEEE Global Communications Conference*.

[25] S. Varrette, H. Cartiaux, S. Peter, E. Kieffer, T. Valette, and A. Olloh, "Management of an Academic HPC & Research Computing Facility: The ULHPC Experience 2.0," in *Proc. of the 6th ACM High Performance Computing and Cluster Technologies Conf. (HPCCT 2022)*. Fuzhou, China: Association for Computing Machinery (ACM), July 2022.

[26] A. E. Marvasti and Y. P. Fallah, "Bandwidth-adaptive feature sharing for cooperative lidar object detection," in *2020 IEEE 3rd Connected and Automated Vehicles Symposium (CAVS)*. IEEE.

[27] C. Liu, "Enhance the 3d object detection with 2d prior," *IEEE Access*.

[28] S. Ren, K. He, R. Girshick, and J. Sun, "Faster r-cnn: Towards real-time object detection with region proposal networks," *Advances in neural information processing systems*, vol. 28, 2015.

[29] F. Hawlader, F. Robinet, and R. Frank, "Vehicle-to-infrastructure communication for real-time object detection in autonomous driving," in *18th Wireless On-Demand Network Systems and Services Conference (WONS)*, 2023, p. 40.

[30] C. Liu and Shuang-Hua, "Cooperative perception with learning-based v2v communications," *IEEE Wireless Communications Letters*, 2023.

[31] M. Řeřábek and T. Ebrahimi, "Comparison of compression efficiency between hevc/h. 265 and vp9 based on subjective assessments," in *Applications of digital image processing XXXVII*. SPIE, 2014.

[32] H. Choi and I. V. Bajić, "Deep feature compression for collaborative object detection," in *2018 25th IEEE International Conference on Image Processing (ICIP)*. IEEE, 2018, pp. 3743–3747.

[33] K. He, X. Zhang, S. Ren, and J. Sun, "Deep residual learning for image recognition," in *Proceedings of the IEEE conference on computer vision and pattern recognition*, 2016, pp. 770–778.

[34] L. Liu and W. Shi, "Computing systems for autonomous driving: State of the art and challenges," *IEEE Internet of Things Journal*, 2020.

[35] T. Mehdi, E. Gamal, and F. Raphael, "Robocar: A rapidly deployable open-source platform for autonomous driving research," *arXiv*, 2024.

[36] A. B. De Souza, V. H. C., and B. Sikdar, "Computation offloading for vehicular environments: A survey," *IEEE Access*, 2020.

[37] K. He, X. Zhang, S. Ren, and J. Sun, "Deep residual learning for image recognition," in *2016 IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, 2016, pp. 770–778.

[38] T. Wang, X. Zhu, J. Pang, and D. Lin, "Fcos3d: Fully convolutional one-stage monocular 3d object detection," in *2021 IEEE/CVF International Conference on Computer Vision Workshops (ICCVW)*, 2021.

[39] zlib, "Compression," https://zlib.net/, Accessed: June. 9, 2023.

[40] J. Peeck, J. Schlatow, and R. Ernst, "Online latency monitoring of time-sensitive event chains in safety-critical applications," in *2021 Design, Automation & Test in Europe Conference & Exhibition (DATE)*, 2021.

[41] H. H. Imtiaz and S. Tang, "Multi-task partial offloading with relay and adaptive bandwidth allocation for the mec-assisted iot," *Sensors*, 2022.