Bandwidth-adaptive Cloud-Assisted 360-Degree 3D Perception for Autonomous Vehicles

Faisal Hawlader[a,*], Rui Meireles[b], Gamal Elghazaly[a], Ana Aguiar[c], Raphaël Frank[a]

[a] *Interdisciplinary Centre for Security, Reliability, and Trust (SnT), University of Luxembourg, L-1855, Luxembourg*
[b] *Computer Science Department, Vassar College, Poughkeepsie, NY 12604, USA*
[c] *Instituto de Telecomunicações, Faculdade de Engenharia, Universidade do Porto, Rua Dr. Roberto Frias, 4200–465 Porto, Portugal*

Abstract

A key challenge for autonomous driving lies in maintaining real-time situational awareness regarding surrounding obstacles under strict latency constraints. The high processing requirements coupled with limited onboard computational resources can cause delay issues, particularly in complex urban settings. To address this, we propose leveraging Vehicle-to-Everything (V2X) communication to partially offload processing to the cloud, where compute resources are abundant, thus reducing overall latency. Our approach utilizes transformer-based models to fuse multi-camera sensor data into a comprehensive Bird's-Eye View (BEV) representation, enabling accurate 360-degree 3D object detection. The computation is dynamically split between the vehicle and the cloud based on the number of layers processed locally and the quantization level of the features. To further reduce network load, we apply feature vector clipping and compression prior to transmission. In a real-world experimental evaluation, our hybrid strategy achieved a 72 % reduction in end-to-end latency compared to a traditional onboard solution. To adapt to fluctuating network conditions, we introduce a dynamic optimization algorithm that selects the split point and quantization level to maximize detection accuracy while satisfying real-time latency constraints. Trace-based evaluation under realistic bandwidth variability shows that this adaptive approach improves accuracy by up to 20 % over static parameterization with the same latency performance.

Keywords: Autonomous Driving, 3D Object Detection, Hybrid Edge-Cloud Computing, V2X Communication, Bandwidth Adaptation, Latency-Constrained Optimization

1. Introduction

The development of fully autonomous vehicles has driven significant progress in the 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]. Such perception capabilities are 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 Company indicated that future vehicles may need to allocate up to 47 % of their energy to onboard computing [2]. These observations highlight a fundamental tension between perception accuracy and real-time latency requirements under constrained onboard computing resources.

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 the 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 for cloud processing, post-training quantization [19], clipping (i.e., outlier removal) [20], and compression [10] can be employed. Together, these techniques significantly reduce the amount of transmitted data. By reducing bandwidth requirements and transmission latency [21], these techniques enable real-time processing within hybrid computing environments [20]. However, quantization, clipping, and compression can negatively impact detection quality due to data loss [20]. Moreover, in practical vehicular deployments, latency is not only determined by computation, but also by fluctuating wireless bandwidth and mobility induced variability. Static offloading configurations may therefore violate latency bounds under changing network conditions. Finding an optimal trade-off between end-to-end delay and detection quality under realistic network dynamics remains a critical area of research.

To explore these trade-offs, we propose a dynamic hybrid computing strategy based on BEVFormer that integrates cooperative perception for 360-degree 3D detection and adaptively adjusts the split layer and quantization level to optimize perception under real-time constraints. Cooperative percep-

*Corresponding author.

tion enables vehicles and infrastructure to share sensor data via Vehicle-to-Everything (V2X), overcoming limitations such as occlusions and sensor range constraints [22]. By exchanging Cooperative Perception Messages (CPMs), which include information about detected objects, this method extends perception beyond onboard sensors [23]. 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 showed a 72 % average reduction in end-to-end delay compared to onboard-only computing, for matched quantization levels.

Different split depths and quantization levels induce distinct latency accuracy trade-off profiles. With that in mind, we propose a dynamic parameter selection scheme that, given the available network bandwidth, maximizes detection accuracy while satisfying a target latency bound. Our evaluation demonstrates that the dynamic strategy can improve accuracy, relative to a static parameterization with the same end-to-end latency, by 10 to 20 %, over a wide gamut of network bandwidth and latency bound combinations. Our contributions can be summarized as:

- **Hybrid-computing perception scheme:** We introduce a BEVFormer-based hybrid computing perception scheme that is able to split computation between vehicle and cloud, transferring data over V2X. It applies data quantization, clipping, and compression, to reduce offloading latency while preserving detection quality.
- **Real-world testing:** We ran tests in a real-world scenario with vehicular mobility and V2X integration. We benchmarked both a fully-onboard computing solution, and our proposed hybrid scheme. We present an analysis of the impact of different hybrid-computing parameterizations on detection accuracy and overall latency.
- **Dynamic parameter selection:** We extend the base hybrid-computing scheme with a constrained-optimization dynamic parameter selection algorithm. It maximizes detection accuracy while respecting strict latency constraints under volatile network conditions.

The remainder of this paper is organized as follows: Section 2 reviews the relevant literature. Section 3 details our proposed method, including the on-board and cloud components, as well as the test route and communication technologies used. Section 4 presents experimental results, focusing on the latency versus accuracy trade-off. Section 5 introduces the hybrid-computing parameter optimization scheme. Finally, Section 6 summarizes our findings and outlines potential future research directions.

2. Related Work

Cooperative perception [22] has gained significant attention as a method to enhance the situational awareness of autonomous vehicles [1, 24, 25]. By enabling vehicles to share sensor data and computational resources [26, 27], these systems can significantly improve object detection [28, 9] and prediction in complex environments [29]. Several recent studies have explored vehicle-to-cloud (V2C) communication to offload perception tasks to remote servers [30, 10, 8]. By utilizing the superior compute capabilities of the cloud [31], these approaches reduce the onboard processing burden and energy consumption [32, 2]. Early works in the field, such as [33, 34], focused on transmitting raw data to the cloud.

However, these approaches suffered from bandwidth limitations and high transmission latency [35, 36], making them unsuitable for real-time applications [37]. To address these challenges, prior works such as [18, 20] explored feature-level offloading [4], where intermediate neural feature vectors are transmitted instead of raw data [34], significantly reducing bandwidth usage [38]. However, the size of these feature vectors can still be prohibitively large [39, 14], especially when generated by deep neural networks like ResNet101 [40] or BEVFormer [11]. Recent studies, such as [14], have proposed various methods for compressing feature vectors [16], including quantization and lossy compression [16, 41]. Although these methods reduce the size of the transmission data, they often result in a degradation of the detection accuracy [13]. Similarly, approaches such as DistillBEV [9] improve computational efficiency through knowledge distillation. Our work builds on these efforts by incorporating percentile based clipping within a bandwidth-adaptive hybrid offloading framework 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.

Nevertheless, the performance of feature offloading systems heavily depends on the choice of the split layer [15, 42] that is, the point at which the neural network is divided between the vehicle and the cloud [8, 43]. Prior studies have largely relied on static split configurations [16, 21], assuming stable or idealized communication links [44]. For example, DeepSplit [14] investigates model partitioning strategies to reduce inference latency, typically relying on profiled or fixed communication characteristics. However, such approaches do not explicitly integrate real-time bandwidth estimation into the inference loop. However, in real-world V2X deployments [45], communication links are subject to frequent variations in latency [46], jitter, and available bandwidth due to vehicle mobility and fluctuating network load [47]. In such settings, static offloading strategies are vulnerable to latency violations or inefficient resource usage [48]. Without real-time adaptability [48], such systems risk exceeding latency budgets or underutilizing available resources, ultimately degrading perception accuracy or introducing unnecessary delays.

Additionally, while most prior studies rely on static environments or simulation-based evaluations, our system is validated through real-world vehicular experiments using V2X communication. This enables us to account for network jitter and bandwidth variability, providing a more robust evaluation of real-time capabilities and performance of the Cooperative Perception System (CPS). Furthermore, we introduce a dynamic pa-

rameter selection algorithm that adapts the split layer and quantization level, optimizing detection accuracy while satisfying latency constraints under fluctuating network conditions.

3. Methodology

In this section, we describe the system architecture and hybrid-computing methodology for the proposed 360-degree 3D perception framework. Multi-view (6x) camera images are processed using BEVFormer [11], a 3D object detection framework for autonomous driving. The perception pipeline is partitioned between the vehicle and a remote cloud server, where early network layers are executed onboard and deeper layers are offloaded according to the selected split configuration. Intermediate feature representations are transmitted over the wireless link for cloud processing. 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. After completion of inference under the selected split configuration, the detected 3D objects are then encoded into CPMs and broadcast to nearby vehicles or infrastructure via standardized V2X communication protocols defined by The European Telecommunications Standards Institute (ETSI) [23]. The hybrid-computing evaluation accounts for realistic vehicular communication effects through empirically measured effective transfer latency, capturing bandwidth variability.

3.1. Test Scenarios & Routes

The tests were carried out on public roads in the Kirchberg area of Luxembourg, which includes various road layouts and traffic conditions. This environment allowed us to evaluate perception scenarios in highly realistic settings. Communications between vehicles and infrastructure were handled by the YoGoKo Y-Box module, which supports ITS-G5 V2X and C-V2X technologies. For more information on the test vehicle, sensors, hardware, and software stack, refer to [49]. For this work, we set up 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 a CPM and transmitted to nearby vehicles and 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 conditions. 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. The stationary receiver enables controlled and repeatable measurement of CPM latency and packet delivery as a function of distance and channel quality, while avoiding additional variability from dual mobility. Although dense multi-vehicle scenarios may introduce stronger multipath and contention effects, this setup isolates communication performance under single vehicle mobility.

In the **hybrid computing scenario**, BEVFormer is partitioned into a local component and a cloud component. Multi-view images are processed locally by BEVFormer through its early network layers to extract feature vectors, which are then clipped, compressed, and sent to the cloud via C-V2X. The cloud server executes the remaining layers to complete 3D object detection. The detection results are encoded into CPMs and broadcast to surrounding vehicles via C-V2X, enabling cooperative perception. The test route spans approximately 4 km^2. We use the cellular mode of C-V2X to communicate with a cloud server located at the University of Luxembourg. Twelve commercial base stations, including 4G and 5G (non-standalone) sites, operate along the route on low-band (700 MHz) and mid-band (3.6 GHz) frequencies. Drive tests along this route showed an average downlink throughput of 57 Mbit/s for 4G and 115 Mbit/s for 5G, with uplink speeds ranging from 25 to 35 Mbit/s. These measurements were obtained under real vehicular mobility, capturing practical bandwidth fluctuations and scheduling variability typical of commercial cellular deployments. We use UDP for data offloading to the cloud to minimize transport-layer overhead and end-to-end latency [50]. Given the real-time constraints of perception offloading, latency minimization is prioritized over strict reliability guarantees. Potential packet loss is mitigated at the application level through frame-level processing and periodic updates.

3.2. Hardware Configuration & Detection Model

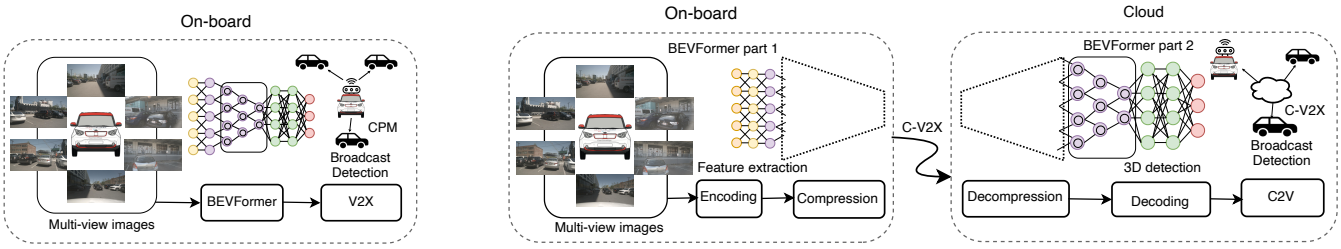
The hardware configurations used in this study are outlined in Table 1. The onboard setup utilizes a Jetson Orin, selected for its low power consumption and processing capabilities in embedded perception tasks. The cloud platform employs 4 Tesla V100 GPU nodes, designed to handle computationally intensive tasks. For more details on GPU node configurations, we refer the reader to [32]. We use the BEVFormer model with a ResNet101 backbone [51], initialized from the FCOS3D checkpoint [12]. BEVFormer is a transformer-based 3D object detection framework designed for autonomous driving perception. It processes multi-view camera images to produce a BEV representation of the scene, enabling 360-degree perception. The model architecture consists of three key stages: (i) a CNN-based backbone (e.g., ResNet101) extracts 2D features from multi-view camera inputs, (ii) a view transformer fuses the features into a unified BEV space, and (iii) a BEV encoder refines these representations using temporal and spatial self-attention.

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 1: Hardware setups for the onboard and cloud computing platforms described in Section 3.1.

[1] **Onboard computing test route:** `http://g-o.lu/3/GsHC`

[2] **Hybrid computing test route:** `http://g-o.lu/3/96TS`



(a) Onboard computing scenario

(b) Hybrid computing scenario

Figure 1: 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 processing, with detection results broadcast to nearby vehicles.

The detection head then predicts 3D bounding boxes, including object positions, orientations, and sizes. Although BEVFormer is trained using LiDAR data for supervision [52], it operates solely on camera inputs during inference, making it well suited for real-time, camera-only 3D perception. For more details on BEVFormer, we refer the reader to [11]. In our hybrid setup, we split computation 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. The split point is selected to balance onboard computational load and transmitted feature dimensionality, directly impacting lat_{local} and communication latency as formalized in Eq. 4. Alternative split positions are explored experimentally in Section 4.

3.3. Input Dataset and Evaluation Metric

For this study, we use the **nuScenes** dataset [52], a large-scale dataset specifically created for autonomous driving research. The dataset consists of 1600×900 resolution images from six cameras, five radars, and one LiDAR, providing full 360-degree coverage, perfectly aligning with our objective of achieving 3D detection. The dataset also includes detailed annotations for 3D object detection, tracking, and segmentation across various classes such as vehicles, pedestrians, and cyclists. We use a BEVFormer model pre-trained on the nuScenes dataset without performing any additional training. The dataset is used exclusively for perception accuracy evaluation under different quantization, clipping, and compression configurations, while communication performance is evaluated separately through real-world experiments and trace-based analysis. The evaluation aims to quantify potential detection accuracy loss resulting from quantization, clipping, and compression. To evaluate performance across all cases, we use the **nuScenes Detection Score (NDS)**, which offers a comprehensive assessment of detection tasks. The NDS ranges from 0 to 1, is calculated using the following formula [11]:

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

The score is composed of two equal-weight halves. One half is the mean average precision (mAP), calculated over different object classes and matching distance thresholds.

mAP reflects the precision-recall trade-off across detection thresholds and object categories. The other half measures detection quality through bounding-box and attribute error metrics. This is calculated through five types of error: translation, scale, orientation, velocity, and an other-attributes error term (e.g., whether a pedestrian is sitting or standing). $\mathbb{TP}$ denotes the set of mean errors mTP for each of the five types. The errors are capped to 1 and the result is subtracted from 1, so that the score is maximized when the errors are all zero.

3.4. Lightweight Features Offloading: Hybrid Computing

In hybrid computing, multi-view images captured around the vehicle are first processed onboard by the initial backbone layers, extract intermediate feature vectors. These features are then clipped and compressed to their dimensionality and transmission size before being transmitted 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. Algorithm 1 defines the perception loop, which periodically runs the perception task by calling upon the local and cloud routines.

Algorithm 1 Perception loop

Require:
$nets$: set of n-layer backbone networks, one per precision/quantization level q
$split$: split layer index, integer $\in \{1, \ldots, n\}$
q: quantization level
$cliPcen = (cliPcen_{low}, cliPcen_{up})$: clipping percentiles
Δt: time period between consecutive perception runs

procedure PERCLOOP($nets$, $split$, q, $cliPcen$, Δt)
  while vehicle is driving **do**
    $t_{start} \leftarrow$ current time
    $net \leftarrow nets_q$ ▷ backbone corresponding to q
    $fVec_{comp} \leftarrow$ percLoc(net, $split$, $cliPcen$)
    percCloud($fVec_{comp}$, $split$, q) ▷ cloud routine
    $t_{end} \leftarrow$ current time
    sleep max $(0, \Delta t - (t_{end} - t_{start}))$
  end while
end procedure

On-board Component: Algorithm 2 defines the processing performed on the vehicle. It begins by acquiring the raw

Algorithm 2 On-board perception function

Require:
net: n-layer backbone network (configured at precision q)
$split$: split layer index, integer $\in \{1, \ldots, n\}$
$cliPcen = (cliPcen_{low}, cliPcen_{up})$: clipping percentiles
Ensure: $fVec_{comp}$: clipped and compressed feature vector

function PERCLOC(net, $split$, $cliPcen$)
  $imgData \leftarrow$ raw multi-view image input
  $fVec \leftarrow imgData$
  for $l = 1$ to $split$ **do** ▷ execute onboard backbone layers
    $fVec \leftarrow net_l(fVec)$
  end for
  $thres_{low} \leftarrow$ percentile$(fVec, cliPcen_{low})$
  $thres_{up} \leftarrow$ percentile$(fVec, cliPcen_{up})$
  $fVec_{clip} \leftarrow$ clip$(fVec, thres_{low}, thres_{up})$
  $fVec_{comp} \leftarrow$ compress$(fVec_{clip})$ ▷ lossless (zlib)
  return $fVec_{comp}$
end function

Algorithm 3 Cloud perception procedure

Require:
$nets$: set of n-layer backbone networks, one per precision/quantization level q
$fVec_{comp}$: compressed perception feature tensor
$split$: split layer index, integer $\in \{1, \ldots, n\}$
q: quantization level

procedure PERCCLOUD($nets$, $fVec_{comp}$, $split$, q)
  $net \leftarrow nets_q$ ▷ backbone corresponding to q
  $fVec \leftarrow$ decompress$(fVec_{comp})$
  for $l = split + 1$ **to** n **do** ▷ execute remaining layers
    $fVec \leftarrow net_l(fVec)$
  end for
  percepRes $\leftarrow$ BEVFormer.remainder$(fVec)$
  CPM $\leftarrow$ encode(percepRes)
  broadcast CPM
end procedure

multi-view image data. The backbone network $net^{(q)}$, configured at precision level q, extracts an intermediate feature tensor $fVec \in \mathbb{R}^{C \times H \times W}$, where C denotes the number of channels and H, W represent the spatial height and width dimensions. Computation proceeds up to the layer specified by the $split$ parameter. Layers beyond this point are executed in the cloud. The split point determines the balance between onboard and cloud computation. Because the feature tensor typically decreases in spatial resolution and channel dimensionality across layers, the split depth also directly influences the amount of data transmitted to the cloud. To reduce transmission overhead, $fVec$ is percentile-clipped using thresholds $cliPcen_{low}$ and $cliPcen_{up}$, where values outside this range are truncated. Based on empirical evaluation, the 10^{th} and 90^{th} percentiles provide a favorable trade-off between feature size reduction and detection accuracy. Clipping reduces the dynamic range and entropy of the feature distribution, thereby improving compressibility. The clipped feature tensor is subsequently compressed using the lossless algorithm [53] before being transmitted to the cloud via C-V2X.

Cloud Component: Algorithm 3 specifies the perception operations executed in the cloud. The procedure begins by decompressing the received feature tensor and instantiating the backbone corresponding to precision level q. The remaining backbone layers (i.e., layers $split + 1$ to n) are then executed to complete feature extraction. A multi-GPU setup (see Section 3.1) is employed for this stage, significantly accelerating processing compared to onboard execution. Upon completion of the backbone, the remaining BEVFormer modules (view transformation, BEV encoding, and detection head) are executed, resulting in a set of 3D bounding boxes. Finally, as in the onboard-only scenario, the detection results are encoded into a CPM and transmitted via C-V2X to nearby vehicles and infrastructure, thereby enabling cooperative perception.

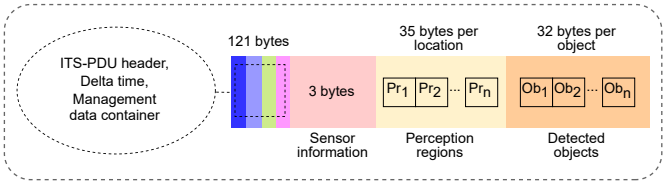


Figure 2: A basic overview of different containers included in the CPM message format as defined by the ETSI standard [23].

3.5. CPM Encoding

The CPM encoding process packages detected objects and environmental data into a standardized format defined by ETSI [23], ensuring CPS interoperability. As illustrated in Figure 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 use of ETSI-compliant CPM messages ensures compatibility with existing cooperative perception systems and facilitates integration into practical V2X deployments. The encoding process is managed by the YoGoKo Y-Box module [49], which supports both ITS-G5 and C-V2X technologies, enabling seamless V2X communication.

4. Experiments and Results

In this section we assess the performance of our proposed 3D perception system, using the setup described in Section 3. Each experiment was repeated five times, for statistical robustness.

4.1. Onboard Computing and CPM Transmission

To establish a baseline, we performed inference tests on the onboard platform, as detailed in Section 3.2, using the nuScenes dataset described in 3.3. 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 of 100 ms [10]. The 100 ms bound is widely adopted in the literature as a practical upper limit for maintaining safe and responsive perception-driven control in urban driving scenarios. 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 tool. Such sustained utilization levels indicate limited headroom for additional perception or control tasks, further constraining real-time operation. These results highlight the limitations of onboard computing, particularly regarding resource usage and latency.

Model Optimization using TensorRT: To address resource usage and latency, we optimized the model with TensorRT [5]. TensorRT enhances performance by applying precision calibration, or quantization, (FP32, FP16, or FP8, with the numeric suffix indicating how many bits are used) and reducing the model's computational complexity. The experimental results in Table 2 show how TensorRT optimization significantly decreased inference times, without major degradation in detection accuracy. For instance, inference time dropped from 486 ms with FP32 to 194 ms with FP8, with only a marginal decrease in NDS from 0.52 to 0.51, indicating that detection performance was 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 [37, 29]. While quantization substantially improves onboard performance, the resulting latency still remains above stringent real-time perception targets, motivating the need for hybrid offloading.

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

Table 2: 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 3.2. The table includes inference time and CPM transmission latency, which together form the end-to-end delay. Standard deviations (±) 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 ITS-G5. We performed a CPM transmission test to measure the associated end-to-end latency in a cooperative perception scenario. Table 3 summarizes the network configuration used. The experiments were conducted under real vehicular mobility conditions to capture practical channel variability and interference characteristics of commercial ITS-G5 deployments.

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 Mbit/s
Number of CPM Transmitted / loss ratio	6000 / 0.09

Table 3: Important network parameters for V2X.

A single static receiver node was placed at a fixed location, stated in the caption of Figure 3. The sender node was placed in a moving vehicle, as detailed in Section 3.1.

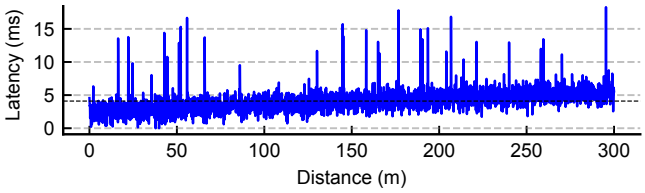


Figure 3: CPM transmission latency versus distance between a moving vehicle (25 km/h avg.) and a stationary receiver at fixed coordinates (longitude: 6.161993, latitude: 49.626478). Dashed line shows average latency 4.10 ms.

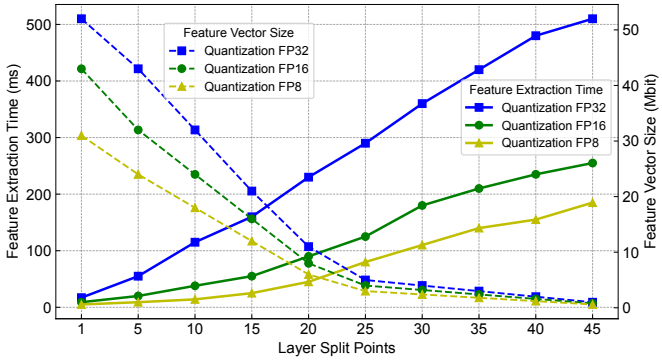


Figure 4: Feature vector size and extraction time versus split depth. Solid lines represent feature extraction time (left y-axis), while dashed lines indicate feature size (right y-axis). Lower-precision quantization (e.g., FP16, FP8) reduces both extraction time and feature size.

The results, 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 distance grows, likely due to propagation delay 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 distance exceeded 300 m, highlighting sensitivity to longer distances. These observations are consistent with previous simulation-based research [37].

Although TensorRT helped significantly reduce end-to-end delay, as shown in Table 2, onboard processing still falls short of the 100 ms real-time perception target. For example, FP8 quantization yields an end-to-end delay (onboard inference plus transmission) of 198.9 ms, which is almost double the desired threshold. Such latency corresponds to an effective perception update rate of approximately 5 Hz, significantly below the 10 Hz rate typically associated with 100 ms class perception systems. Consequently, the end-to-end delay restricts CPM transmission rates to below 5 Hz, highlighting the need for a more efficient solution to support real-time perception.

4.2. Hybrid 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 the network, focusing on split layer selection, accuracy retention, and end-to-end delay.

Layer Partitioning and Feature Extraction: In the hybrid computing case, determining the optimal backbone partition layer is crucial, as it affects both the onboard feature extraction time and the size of transmitted features.

In BEVFormer, partitioning earlier, e.g., layer 1, minimizes onboard computation but requires transmitting larger feature vectors to the cloud. Conversely, partitioning later reduces the size of the transmitted feature vector $S_{feat}(split, q)$—defined as the compressed payload (in bits) generated at the selected split point and quantization level—at the cost of additional onboard inference time. Figure 4 illustrates the trade-off between inference time and feature vector size for different split points and quantization levels.

- **Feature extraction time:** As more backbone layers are executed locally, feature extraction time increases, significantly impacting real-time perception feasibility. A split at layer 5 with FP32 quantization 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 splits, where even FP8 exceeds 140 ms by layer 30. These results indicate that split depth has a dominant influence on onboard latency, and aggressive quantization alone cannot compensate for excessive local computation.

- **Feature vector size:** Shallow or earlier splits generate large feature vectors, making data transmission slow. For example, FP32 split after layer 1 produces 52 MB, requiring a throughput of 520 Mbit/s at 10 Hz, far exceeding typical V2X bandwidth. Even FP16 (43 MB, 430 Mbit/s) and FP8 (31 MB, 310 Mbit/s) require too much bandwidth. Deeper splits, however, reduce vector sizes: FP32 split after layer 25 requires 5 MB (50 Mbit/s), and FP8 only 3 MB (30 Mbit/s), which are more suitable for real-time transmission. Nevertheless, deeper splits increase onboard processing time, creating a fundamental latency bandwidth trade-off. This trade-off highlights the need for adaptive split selection under dynamic bandwidth constraints, motivating the dynamic parameter optimization scheme introduced in the next Section 5.

Feature Compression and Transmission: To reduce feature vector sizes beyond what quantization can offer, we employed dynamic clipping and zlib compression. These techniques reduced feature size by approximately 97 % for FP32, 90 % for FP16, and 80 % for FP8, significantly improving transmission efficiency across the board. The positive correlation between encoding bits and compressibility means that, in this dataset, entropy grows sublinearly with the number of bits used, allowing the compression algorithm to take advantage of the added repetition or structure in the data. Figure 5 shows the observed transmission latency for different quantization and split layer combinations. We focus on layers 1 to 5 because deeper splits yielded minimal improvements in transmission latency, while increasing feature extraction time. FP8 exhibited the lowest latency, with medians of 52 ms at layer 1 and 35 ms at layer 5, meeting the threshold required for real-time perception systems. In contrast, FP32 exhibited the highest latency, reaching 90 ms at layer 1 and 70 ms at layer 5. Variance was also the highest among the different quantization levels. FP16 provided a good middle ground, with median latencies of 67 ms at layer 1 and 45 ms at layer 5, making it suitable for applications that can tolerate moderate delays. FP8, with its consistently low latency, is ideal for latency-critical applications communicating over low bandwidth-networks, whereas FP16 and FP32 are more suitable for faster networks, or scenarios with more relaxed latency bounds. These observations confirm that quantization level and split depth jointly determine the feasible latency region, reinforcing the need for adaptive configuration under dynamic bandwidth conditions.

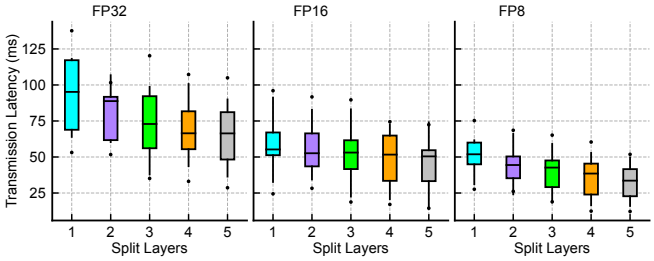


Figure 5: Transmission latency of feature vectors from vehicle to cloud across five split layers for FP32, FP16, and FP8 over a 5G network using C-V2X. FP8 demonstrates the lowest and most stable latency, suitable for real-time transmission. FP32 exhibits the highest latency and variability, especially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

End-to-End Delay: The results in Table 4 detail the impact of split layer and quantization level choice on the different latency components. For FP32 quantization at split layer 1, the total end-to-end delay was 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 yielded a significantly lower total delay of 73.8 ms, mainly due to the shorter 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 0.47, compared to 0.52 for FP32.

As the split layer deepens (e.g., layer 5), onboard processing time increased, due to the more complex feature extraction. For FP32, the total delay at split layer 5 grew to 137.7 ms, with 60.5 ms for local processing and 62.1 ms for data transmission. In comparison, FP8 achieved a lower end-to-end delay of 61.9 ms at the same split layer, primarily due to its shorter local processing time of 12.7 ms and transmission latency of 36.3 ms. FP8 at split layer 5 achieved the lowest end-to-end delay (61.9 ms) and bandwidth usage (4.1 Mbit/s), with only a marginal drop in accuracy (NDS = 0.43). Cloud processing times (decompression and detection head) remained consistently low across all quantization levels due to the powerful hardware, with decompression ranging from 1.4 to 2.6 ms. This stability ensured that the majority of delay originated from lo-

FP	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 (Mbit/s)
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)	(±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 4: Performance metrics 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. Rows with end-to-end delay below 100 ms are highlighted in green.

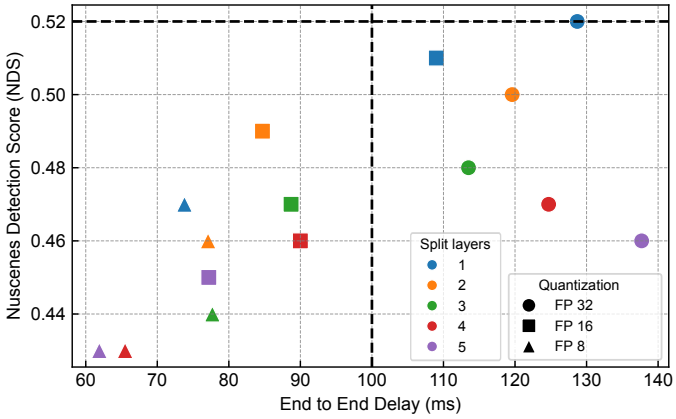


Figure 6: End-to-end delay vs. detection accuracy (NDS) across split layers and quantization levels. Earlier layers result in higher accuracy but increased delay, while intermediate layers provide a balance between delay and accuracy.

cal processing and transmission, emphasizing the need to carefully select the optimal split point and quantization level for real-time systems. Transmission latencies showed greater variability than local or cloud processing, with the highest deviations observed at split layer 1 for FP32 (±4.00 ms V2C) and FP8 (±2.80 ms V2C), reflecting the influence of fluctuating network conditions. While lower precisions like FP8 reduced total delay, they introduced minor accuracy degradation a trade-off further examined in the following section.

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. A clear trend is that deeper split points lead to reduced detection accuracy, regardless of the selected quantization level. This suggests that clipping and quantization applied to deeper feature representations may remove information that has become more semantically compact and task-relevant. As network depth increases, feature representations become progressively more compressed and discriminative, so additional information loss due to reduced precision or clipping has a proportionally larger impact on detection performance.

The other major trend is that increasing the quantization level, while holding the split layer constant, increased both delay and detection accuracy alike. From this it follows that (split=1, FP32) yielded the highest accuracy and delay, and (split=5, FP8) the lowest accuracy, delay combination. A particularly well-performing combination was (split=1, FP16), which yielded 98 % of (split=1, FP32)'s accuracy, but only 85 % of the delay. The (split=3, FP16) combination, a visual demonstration of which is shown in Figure 7, offered an NDS of 0.47 and a delay of 88.7 ms. This configuration satisfies commonly adopted sub-100 ms real-time perception bounds while preserving competitive detection accuracy. This makes it a practical solution for perception systems with a 100 ms latency bound.

As shown in Table 4, the impact of changing the split layer on end-to-end delay is not straightforward, even when the quantization level is held constant. For instance, with FP32 quantization, splitting at layer 2 results in a lower delay (119.6 ms) compared to layer 1 (128.7 ms), despite the deeper layer. Conversely, moving from layer 3 to layer 4 increases the delay from 113.5 ms to 124.7 ms, highlighting the non-monotonic nature of the trade-off. This behavior stems from a complex interplay between feature vector size, onboard computation, and transmission time. Consequently, static configurations are often suboptimal under real-world network conditions that exhibit fluctuating bandwidth and latency. To address this limitation, we introduce a dynamic parameter selection mechanism that jointly adapts the split layer and quantization level based on real-time bandwidth availability. The objective is to maximize detection accuracy while satisfying a strict end-to-end latency budget, as detailed in Section 5.

5. Dynamic Hybrid-Computing Parameter Selection

We have shown that varying the split point and quantization level leads to different trade-offs among bandwidth usage, end-to-end detection latency, and perception accuracy. In this section, we propose a dynamic selection algorithm that jointly opti-

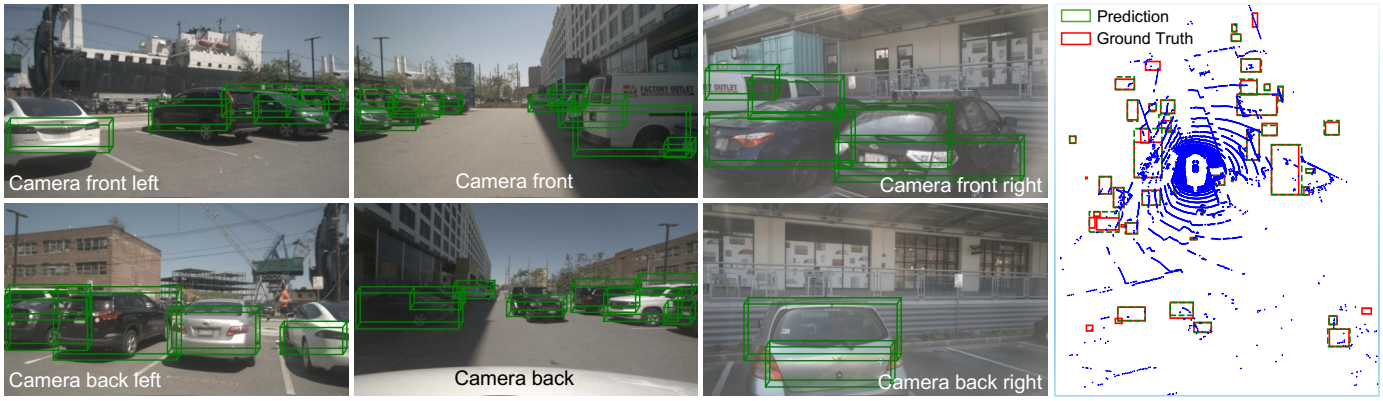


Figure 7: Qualitative results at split layer 3 with FP16 on the nuScenes validation set (SceneID: n008-2018-05-21-11-06-59-0400), an NDS score of 0.47. We show 3D bounding box predictions in multi-view (6x) camera images and the 360-degree BEV.

mizes these two parameters to maximize detection performance while adhering to bandwidth and latency constraints.

5.1. Problem Formulation

The goal is to select the split point *split* and quantization q parameters that maximize the NDS accuracy metric from Eq. 1:

$$(split, q) = \arg\max_{split,q} NDS(split, q). \tag{2}$$

This maximization must be subject to two constraints:

Bandwidth budget (bw_{upl}, bw_{dwn}): the available uplink and downlink bandwidth for the offloading process. This may reflect total channel capacity or a reserved slice allocated for the perception task, depending on network sharing policies, and leaving room for other applications.

Latency limit (lat_{max}): the total amount of time available for the perception task. Ideally set to 100 ms for real-time cooperative perception [10]. The total perception latency, lat_{total}, which is a function of the split point, the quantization level, and the bandwidth budget, must therefore not be larger than lat_{max}:

$$lat_{total}(split, q, bw_{upl}, bw_{dwn}) \leq lat_{max}. \tag{3}$$

This constraint, together with the bandwidth budget, forms the optimization boundary. The total latency can be decomposed into the sum of four distinct phases:

$$\begin{aligned} lat_{total}(split, q, bw_{upl}, bw_{dwn}) = {} & lat_{local}(split, q) \\ & + lat_{upl}(split, q, bw_{upl}) \\ & + lat_{cloud}(split, q) \\ & + lat_{dwn}(split, q, bw_{dwn}). \end{aligned} \tag{4}$$

lat_{local} is the latency incurred locally, computing the BEVFormer backbone layers pre-split, clipping, and compressing the feature vector. lat_{upl} and lat_{dwn} denote the uplink and downlink transmission times, respectively. These terms are modeled using an effective transfer time that is measured or estimated online. This formulation captures the dominant delay components in terrestrial V2X and cellular systems, including radio access scheduling, medium access contention, retransmissions, and queuing effects under vehicular mobility. Propagation delay is negligible in terrestrial deployments [44].

Even over tens of kilometers, propagation delay remains submillisecond, on the order of a few microseconds per kilometer in fiber or radio links. Consequently, propagation delay is excluded from the explicit decomposition in Eq. 4. Mobility induced variability in end-to-end communication delay is implicitly captured in the effective transfer time. As a result, the optimization remains valid under dynamic vehicular conditions and does not compromise real-world applicability. lat_{cloud} is the time required to complete the remaining perception tasks in the cloud. All latency components depend on the selected split point and quantization level, while the communication terms additionally depend on the available bandwidth budget.

5.2. Dynamic Optimization Algorithm

We begin by constructing an array *pArray* that enumerates all valid $(split, q)$ parameter tuples. This array is pre-sorted in non-increasing order of NDS, based on the empirical accuracy measurements in Table 4, such that higher-ranked configurations yield better detection accuracy. During each iteration of the perception loop, the effective available uplink and downlink bandwidths are estimated. The optimization process then selects the first (i.e., highest-ranked) tuple in *pArray* that satisfies the latency constraint defined in Equation 3, where lat_{max} corresponds to the perception cycle period Δt. Given the profiled accuracy and latency components, this approach selects the most accurate configuration that meets the real-time latency constraint, thereby maximizing detection performance under dynamic network conditions.

Algorithm 4 illustrates how this parameter optimization mechanism is embedded into the hybrid perception loop. Modifications relative to the baseline implementation (Algorithm 1) are underlined for clarity. Specifically, the fixed *split* and q inputs are replaced with the adaptive parameter array *pArray*. In each loop iteration, the estimated uplink and downlink bandwidths, denoted bw_{upl} and bw_{dwn}, are used as inputs to the optimization. While the bandwidth estimation algorithm itself is outside the scope of this paper, we assume the existence of a reliable estimator. The current bandwidth estimates, along with

pArray and the latency constraint Δt, are passed to the function *optPar*(), which selects the split layer and quantization level that satisfy the constraint.

The selected parameters are then used in the subsequent perception operations, which otherwise remain unchanged. Algorithm 5 specifies the behavior of the parameter optimization function *optPar*(). The correctness of the optimization function is established by the following theorem, ensuring it selects parameters that maximize accuracy within the latency bound, or minimize delay if the bound is unattainable.

Theorem 1. *optPar*() *returns the highest NDS-yielding parameter tuple* $(split, q)$ *that satisfies the latency bound* $lat_{total} \leq lat_{max}$ *or, if no such tuple exists, the tuple that minimizes* lat_{total}.

First, consider the case where at least one latency-bound-satisfying (i.e., acceptable) tuple exists. `optPar()` traverses the parameter tuple array sequentially from front to back. For each $(split, q)$ combination, it estimates the total latency lat_{total} using Eq. 4, with latency values derived from empirical measurements in Table 4. It then compares lat_{total} with the maximum acceptable latency lat_{max}. The first tuple that satisfies the latency constraint is returned. Since the parameter array is sorted in non-increasing NDS order, this guarantees the returned tuple yields the highest NDS among all acceptable configurations, thereby proving the first part of Theorem 1. Now consider the scenario where no tuple satisfies the latency constraint. To demonstrate the correctness of the algorithm in this case, we define the following loop invariant: at the start of each iteration with index *idx*, *latMin* holds the lowest latency observed among all tuples examined so far (i.e., indices 0 to $idx - 1$), and *latMinIdx* stores the index of the corresponding parameter tuple. We prove that this invariant holds by induction:

Initialization: At the start of the first iteration, the invariant holds trivially because no parameter tuples have been evaluated yet. The variables *latMin* and *latMinIdx* are initialized to a placeholder maximum value and index zero, respectively, ensuring that the first tuple's latency will always be accepted as the current minimum for comparison.

Maintenance: Assume the invariant holds at the beginning of iteration *idx*. After computing the latency *lat* for tuple *pArray*[*idx*], if $lat < latMin$, then lines 10-11 update *latMin* and *latMinIdx* accordingly. Otherwise, these values remain unchanged. Therefore, at the start of the next iteration $(idx + 1)$, *latMin* and *latMinIdx* still represent the minimum latency and corresponding index among all tuples examined so far. Hence, the invariant holds.

Termination: If no latency-bound-satisfying tuple exists, the loop completes after examining all tuples. By the loop invariant, *latMinIdx* then holds the index of the tuple with the minimum total latency. The algorithm returns this tuple in line 14. This confirms the second part of Theorem 1 and thus establishes the correctness of the algorithm.

The asymptotic running time of the algorithm depends on the input. Assuming that lat_{total} can be computed in constant time, the **best case** occurs when the first tuple in the array satisfies the latency constraint. In this case, the loop terminates immediately, yielding a running time of $O(1)$. The **worst case** arises when no tuple satisfies the latency bound. The algorithm must then evaluate all entries in the array, resulting in a time complexity of $O(|pArray|)$. The **average case** depends on current network conditions and the distribution of latency values across tuples. If a valid configuration is typically found early within a constant number of iterations, the expected runtime approaches $O(1)$. Otherwise, it becomes $O(|pArray|)$.

Algorithm 4 Perception loop (adaptive version)

Require:
nets: set of *n*-layer backbone networks, one per quantization
pArray: array of valid parameter tuples $(split, q)$, sorted in non-increasing NDS order
cliPcen: clipping percentiles
Δt: time period between consecutive perception runs

1: **procedure** PERCLOOP(*nets*,*pArray*,*cliPcen*, Δt)
2: **while** vehicle is driving **do**
3: $t_{start} \leftarrow$ current time
4: $(bw_{upl}, bw_{dwn}) \leftarrow$ estimateBandwidth()
5: $(split, q) \leftarrow optPar(pArray, bw_{upl}, bw_{dwn}, \Delta t)$
6: $net \leftarrow nets_q$
7: $fVec_{comp} \leftarrow$ percLoc($net, split, cliPcen$)
8: percCloud($fVec_{comp}, split, q$) ▷ remote call
9: $t_{end} \leftarrow$ current time
10: **sleep** max $(0, \Delta t - (t_{end} - t_{start}))$
11: **end while**
12: **end procedure**

Algorithm 5 Hybrid-computing parameter optimization

Require:
pArray: non-empty NDS-sorted array of par. tuples $(split, q)$
bw_{upl}: upload bandwidth
bw_{dwn}: download bandwidth
lat_{max}: maximum admissible perception latency
Ensure: $(split, q)$: optimal parameter tuple

1: **function** OPTPAR($pArray$, bw_{upl}, bw_{dwn}, lat_{max})
2: $latMin \leftarrow +\infty$
3: $latMinIdx \leftarrow 0$
4: **for** $idx = 0$ **to** $|pArray| - 1$ **do**
5: $(split, q) \leftarrow pArray[idx]$
6: $lat \leftarrow lat_{total}(split, q, bw_{upl}, bw_{dwn})$
7: **if** $lat \leq lat_{max}$ **then**
8: **return** $(split, q)$ ▷ highest-NDS feasible
9: **else if** $lat < latMin$ **then** ▷ lowest latency so far?
10: $latMin \leftarrow lat$
11: $latMinIdx \leftarrow idx$
12: **end if**
13: **end for**
14: **return** $pArray[latMinIdx]$ ▷ minimum-latency fallback
15: **end function**

5.3. Evaluation

We now describe how we evaluated the proposed dynamic parameter selection algorithm, and present the results.

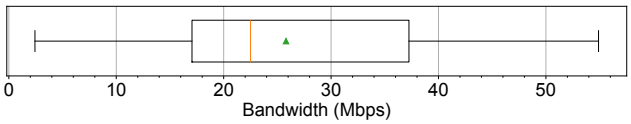


Figure 8: Upload bandwidth distribution used to evaluate the dynamic hybrid-computing parameter selection algorithm. The green triangle indicates the mean, and the orange line marks the median.

Setup: We performed a trace-based evaluation. The bandwidth dataset and source code used in this evaluation are publicly available on GitLab [54].

We collected 1 Hz bandwidth measurements from a realistic vehicular scenario and used our algorithm to select the optimal split layer *split* and quantization level for each entry. The process was repeated under varying latency limits lat_{max} and bandwidth allocations for the perception task to assess performance under different constraints. Because the dataset described in Section 3.1 does not feature direct bandwidth measurements, we collected a new one for this purpose. We used `iperf` running on a Raspberry Pi 4 Model B to stream UDP data to a cloud server, using a commercial 5G cellular network. A total of 654 s of data were collected from a vehicle traveling on an urban route in the city of Porto, Portugal[3]. Figure 8 depicts the distribution of measured bandwidths. The mean and standard deviation were 25.8 Mbit/s and 12.1 Mbit/s, respectively.

Table 4 provided all additional information required to run the algorithm. Specifically, it supplied the NDS values used to rank parameter tuples by accuracy, as well as the data needed to compute the individual latency components in Equation 4.

lat_{local}: The sum of the onboard processing time components associated with each parameter tuple.

lat_{upl}: The feature payload per perception cycle is obtained by converting the measured throughput at 10 Hz into bits per frame. The upload latency is then computed as the ratio between this payload and the available uplink bandwidth bw_{upl}.

lat_{cloud}: The sum of the cloud processing time components associated with each parameter tuple.

lat_{down} is derived from the average C2V transmission latency reported in Table 4, eliminating the need to rerun the experiments. This choice is justified by the empirical observation that C2V latency exhibits low standard deviation (e.g., ±0.70 ms) and consistent average values across test runs, making it a reasonable proxy for downstream latency in our setting.

5.4. Results Analysis

We start by evaluating the dynamic algorithm's ability to adapt to different bandwidth budgets, with the maximum latency held constant at 100 ms. Figure 9 shows the relative frequency with which each parameter tuple was selected, as a function of the bandwidth percentage dedicated to the perception task. As the available bandwidth increases, configurations yielding higher detection accuracy are selected more frequently.

[3]Bandwidth dataset collection route: `https://www.google.com/maps/d/viewer?mid=1ZpoUDMGTsPOjuKaYRngHe1botQZqyY8`

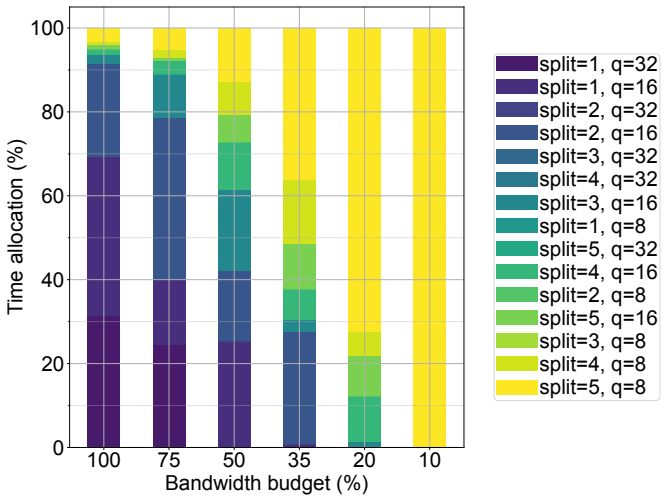


Figure 9: Distribution of parameter tuple usage by dynamic selection algorithm, as a function of the amount of bandwidth dedicated to the perception task.

When the budget was set to 100 %, the two most-accurate tuples, (split=1, FP32) and (split=1, FP16), accounted for almost 70 % of the total selections.

This dropped to slightly above 40 % for a budget of 50 %, and to zero for budgets of 20 % or less. This behavior is consistent with the design of the algorithm: decreasing the bandwidth budget increases communication latency, thereby forcing the optimizer to select lower-accuracy configurations in order to satisfy the latency constraint. Still holding maximum latency constant at 100 ms, we now compare our dynamic algorithm's performance to that of the static ones, in terms of both detection accuracy, and number of latency bound violations. Figure 10 plots these two metrics as a function of the bandwidth percentage assigned to the perception task. Three algorithms are represented: (i) the dynamic selection one, (ii) the accuracy-maximizing static configuration (split=1, FP32) and, (iii), the latency violations-minimizing static configuration (split=5, FP8).

We now examine detection accuracy (NDS). Because static configurations do not adapt to bandwidth variations, their accuracy remains constant: 0.52 for (split=1, FP32) and 0.43 for (split=5, FP8). In contrast, the dynamic algorithm's accuracy increases with available bandwidth, as higher bandwidth permits transmission of richer feature representations to the cloud. With a 100 % bandwidth budget, the dynamic algorithm achieves accuracy within 5 % of (split=1, FP32), while outperforming (split=5, FP8) by approximately 15 %. For a 50 % bandwidth budget, the gap to (split=1, FP32) increases to roughly 10 %, yet the dynamic method still outperforms (split=5, FP8) by about 10 %. Overall, the dynamic algorithm preserves the latency robustness of the lowest-latency static configuration while providing a substantial improvement in detection accuracy.

Finally, we explore the combined impact of latency limits and bandwidth budgets on detection accuracy. Figure 11 presents a surface plot of the accuracy gain achieved by the dynamic algorithm relative to the static configuration that

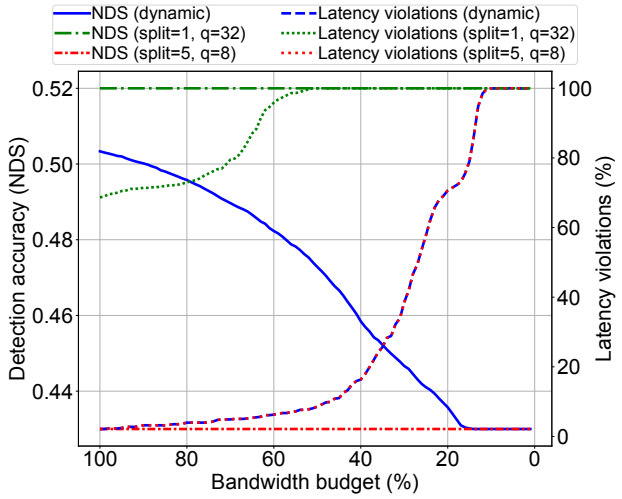


Figure 10: Detection accuracy (NDS) and number of latency bound violations, as a function of bandwidth dedicated to the perception task, for the dynamic parameter selection algorithm, the highest-accuracy static algorithm ($split = 1, FP32$), and the latency minimizing static algorithm ($split = 5, FP8$).

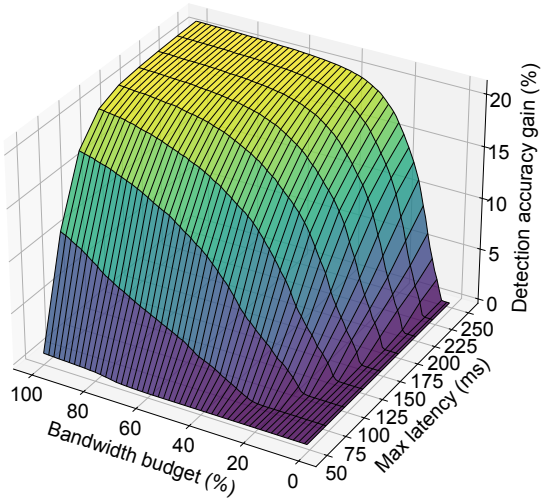


Figure 11: Detection accuracy (NDS) gain offered by dynamic selection algorithm, as a function of the amount of bandwidth assigned to the perception task.

minimizes latency violations for each (lat_{max}, bandwidth) pair. Across all evaluated scenarios, this baseline corresponds to the (split=5, FP8) configuration.

The curve corresponding to $lat_{max} = 100$ ms closely follows the trend observed in Figure 10. Reducing the latency bound further significantly decreases the achievable accuracy gains, which become nearly zero when $lat_{max} = 50$ ms. This outcome is expected, since stricter latency constraints reduce the feasible set of high-accuracy configurations. Conversely, relaxing the latency constraint enables the use of more data-intensive, higher-accuracy configurations, thereby increasing accuracy gains. For example, with a 100 % bandwidth budget, increasing lat_{max} from 100 ms to 250 ms raises the accuracy gain from 15 % to 20 %. Under a 50 % bandwidth budget, the gain increases from 10 % to 19 %. These results demonstrate the ability of the dynamic selection mechanism to adaptively optimize perception performance across diverse network conditions and latency requirements.

5.5. Model Choice and Generalizability

We employ BEVFormer with a ResNet101 backbone as a representative high-accuracy, compute-intensive camera-only BEV detector [11]. On the nuScenes benchmark, BEVFormer-ResNet101 achieves approximately 52 NDS depending on training configuration, while requiring substantial GPU memory and inference time. This makes it a suitable stress case for evaluating hybrid edge-cloud offloading under strict latency constraints.

Alternative BEV-style camera detectors (e.g., PETR [55], StreamPETR [56]) and different backbone depths exhibit distinct accuracy complexity trade-offs. For example, replacing ResNet101 with ResNet50 typically reduces FLOPs by approximately 20-30% and decreases GPU memory usage accordingly, typically reducing NDS by a few points on nuScenes. Similarly, lighter BEV variants reduce intermediate feature dimensionality and inference latency at the cost of moderate performance degradation. In our adaptive framework, such changes directly affect (i) local computation time lat_{local}, (ii) transmitted feature size S_{feat}, and consequently (iii) the feasible $(split, q)$ configurations under a fixed latency constraint, as formalized in Eq. 4. The proposed optimization framework is therefore not tied to a specific backbone, but rather to measurable accuracy, and latency trade-offs.

Importantly, the proposed dynamic optimization framework is architecture-agnostic. It does not rely on model-specific structural assumptions. Instead, it operates on empirically profiled quantities for each candidate $(split, q)$ configuration: (i) detection accuracy (e.g., NDS), (ii) local computation time $lat_{local}(split, q)$, (iii) cloud computation time $lat_{cloud}(split, q)$, and (iv) transmitted feature size $S_{feat}(split, q)$. The latter determines the communication latency components under the available bandwidth. Given these profiles and the effective uplink/downlink transfer times, the optimizer selects the highest-accuracy configuration that satisfies the end-to-end latency constraint. Consequently, the same methodology can be applied to alternative BEV detectors or backbone variants by re-profiling their split layers and quantization levels under the target hardware and network stack.

Beyond backbone choice, the framework is compatible with alternative split learning and compression strategies. For instance, DeepSplit [14] proposes learned intermediate-layer partitioning and compression policies for split inference. Such learned compression modules could replace the current quantization and compression pipeline. In contrast, our contribution focuses on bandwidth-adaptive selection of split depth and precision under strict latency constraints that explicitly account for network variability. Similarly, DistillBEV [9] reduces computation and communication overhead through knowledge distillation, requiring additional training to obtain a compact model. The proposed optimizer remains directly applicable to such architectures, without structural modification, after profiling the corresponding $(split, q)$ configurations.

6. Conclusions and Future Work

We proposed a hybrid-computing 360-degree 3D object detection system featuring a BEVFormer-based model coupled with V2X communication. The approach offloads intensive computations to the cloud while maintaining lightweight feature extraction onboard, enabling real-time perception. Experimental results demonstrated that dynamic clipping, compression, and 5G-enabled C-V2X communication significantly optimize latency and bandwidth utilization. For instance, offloading FP32 feature vectors at 10 Hz following a layer 1 split reduced the bandwidth requirement by approximately 98 %, from 520 Mbit/s to 10.5 Mbit/s

We also investigated the trade-off between end-to-end delay and detection accuracy across various split layers and quantization levels. Our results demonstrated that while FP32 offers the highest accuracy, its substantial end-to-end delay renders it impractical for real-time applications. In contrast, FP8 achieved significantly lower latency with reasonable accuracy, making it suitable for latency-sensitive scenarios. FP16 provided a good compromise between accuracy and latency for applications that require both timely responses and adequate detection performance. This study underscores the importance of selecting appropriate split points and quantization levels based on operational requirements and conditions. Shallower splits with FP32 are optimal for accuracy-focused tasks, whereas deeper splits with FP8 cater to applications with strict latency constraints. Based on these findings, we introduced a dynamic parameter optimization algorithm that varies the split layer and quantization level as a function of the available network bandwidth and target latency bound. In a trace-based evaluation, this algorithm matched the latency violation performance of the fastest static configuration while achieving double-digit accuracy gains across varying bandwidths and latency limits. The achievable accuracy gain depends on the selected latency constraint. When the bound is strict, only a limited set of configurations remains feasible. When the bound is relaxed, higher-accuracy parameter combinations can be selected more frequently. The optimizer relies on empirically profiled latency and accuracy values. Therefore, it can be applied to other BEV backbones or compression strategies after profiling their configurations under the target hardware and network conditions.

A promising direction is to integrate the dynamic parameter selection algorithm into a complete hybrid perception prototype and validate it in real-world driving scenarios, including multi-vehicle and dense traffic settings that exhibit stronger interference, higher contention, and rapid topology changes. This would enable end-to-end evaluation under realistic latency and bandwidth variations. Additionally, our results highlight the sensitivity of perception quality to bandwidth fluctuations, suggesting that combining the algorithm with 5G network slicing, specifically Ultra-Reliable Low Latency Communication (URLLC) slices could ensure consistent latency bounds and improve robustness in dynamic environments. Finally, exploring its applicability over emerging technologies such as 6G and satellite-based networks may extend its benefits to remote or under-connected regions, where stable communication is otherwise difficult to guarantee.

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*. The authors would also like to thank Raquel Lopes from Instituto de Telecomunicações, for help collecting the vehicular cellular bandwidth dataset.

References

[1] S. Sonko, E. A. Etukudoh, K. I. Ibekwe, V. I. Ilojianya, C. D. Daudu, A comprehensive review of embedded systems in autonomous vehicles: Trends, challenges, and future directions, World Journal of Advanced Research and Reviews 21 (1) (2024) 2009–2020.

[2] J. H. Gawron, G. A. Keoleian, R. D. De Kleine, T. J. Wallington, H. C. Kim, Life cycle assessment of connected and automated vehicles: sensing and computing subsystem and vehicle level effects, Environmental science & technology 52 (5) (2018) 3249.

[3] V. Bhardwaj, Ai-enabled autonomous driving: Enhancing safety and efficiency through predictive analytics, International Journal of Scientific Research and Management (IJSRM) 12 (02) (2024) 1076.

[4] J.-S. Lee, T. Ebrahimi, Perceptual video compression: A survey, IEEE Journal of selected topics in signal processing 6 (6) (2012).

[5] T. Pham, B.-J. Maghoumi, J. Truong, 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. doi:10.1109/WACV57701.2024.00721.

[6] S.-W. Kim, B. Qin, Z. J. Chong, X. Shen, W. Liu, M. H. Ang, E. Frazzoli, D. Rus, Multivehicle cooperative driving using cooperative perception: Design and experimental validation, IEEE Transactions on Intelligent Transportation Systems 16 (2) (2015) 663–680. doi:10.1109/TITS.2014.2337316.

[7] E. Marti, M. A. De Miguel, F. Garcia, J. Perez, A review of sensor technologies for perception in automated driving, IEEE Intelligent Transportation Systems Magazine 11 (4) (2019) 94–108.

[8] A. Yaqoob, T. Bi, G.-M. Muntean, A survey on adaptive 360 video streaming: Solutions, challenges and opportunities, IEEE Communications Surveys & Tutorials 22 (4) (2020) 2801–2838.

[9] Z. Wang, C. Li, 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, 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, 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, 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, R. Q. Hu, Challenges and solutions for cellular based v2x communications, IEEE Communications Surveys & Tutorials 23 (1) (2020) 222–255.

[14] R. Mehta, 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, 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. doi:10.1109/VNC57357.2023.10136339.

[16] R. A. Cohen, 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, 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, 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, W. Gao, Post-training quantization for vision transformer, Advances in Neural Information Processing Systems 34 (2021) 28092–28103.

[20] C. Liqun, 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] F. Hawlader, F. Robinet, G. Elghazaly, R. Frank, Cloud-assisted 360-degree 3d perception for autonomous vehicles using v2x communication and hybrid computing, in: 2025 20th Wireless On-Demand Network Systems and Services Conference (WONS), 2025, pp. 1–8.

[22] 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.

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

[24] B. Gao, J. Liu, H. Zou, J. Chen, L. He, K. Li, Vehicle-road-cloud collaborative perception framework and key technologies: A review, IEEE Transactions on Intelligent Transportation Systems (2024).

[25] Z. Wang, Y. Wang, Z. Wu, H. Ma, Z. Li, H. Qiu, J. Li, Cmp: Cooperative motion prediction with multi-agent communication, IEEE Robotics and Automation Letters (2025).

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

[27] L. Bai, J. Cao, M. Zhang, B. Li, Collaborative edge intelligence for autonomous vehicles: Opportunities and challenges, IEEE Network (2025).

[28] H.-k. Chiu, R. Hachiuma, C.-Y. Wang, S. F. Smith, Y.-C. F. Wang, M.-H. Chen, V2v-llm: Vehicle-to-vehicle cooperative autonomous driving with multi-modal large language models, arXiv preprint arXiv:2502.09980 (2025).

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

[30] J. Chen, Leveraging scalable cloud infrastructure for autonomous driving data lakes and real-time decision making, in: 2025 5th International Conference on Artificial Intelligence and Industrial Technology Applications (AIITA), IEEE, 2025, pp. 1750–1753.

[31] H. Wang, J. Liu, H. Dong, Z. Shao, A survey of the multi-sensor fusion object detection task in autonomous driving, Sensors 25 (9) (2025) 2794.

[32] S. Varrette, H. Cartiaux, S. Peter, E. Kieffer, T. Valette, 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), Association for Computing Machinery (ACM), Fuzhou, China, 2022.

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

[34] C. Liu, Enhance the 3d object detection with 2d prior, IEEE Accessdoi:10.1109/ACCESS.2024.3398373.

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

[36] F. Hawlader, F. Robinet, 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. doi:10.23919/WONS57325.2023.10061953.

[37] C. Liu, Shuang-Hua, Cooperative perception with learning-based v2v communications, IEEE Wireless Communications Letters (2023). doi:10.1109/LWC.2023.3295612.

[38] M. Řeřábek, 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.

[39] H. Choi, 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.

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

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

[42] Z. Xiao, J. Shu, H. Jiang, G. Min, H. Chen, Z. Han, Perception task offloading with collaborative computation for autonomous driving, IEEE Journal on Selected Areas in Communications 41 (2) (2022) 457–473.

[43] V. Babaiyan, O. Bushehrian, A deep-reinforcement-learning-based strategy selection approach for fault-tolerant offloading of delay-sensitive tasks in vehicular edge-cloud computing, The Journal of Supercomputing 81 (5) (2025) 1–37.

[44] G. J. Sullivan, J.-R. Ohm, W.-J. Han, T. Wiegand, Overview of the high efficiency video coding (hevc) standard, IEEE Transactions on circuits and systems for video technology (2012).

[45] L. Bai, Z. Huang, Y. Ge, R. Yu, L. Wang, X. Cheng, Cellular vehicle-to-everything (c-v2x) testing: From theory to practice, IEEE Network (2025).

[46] S. Quah, B. Jo, C. Geniesse, L. Uddin, J. Mumford, D. Barch, D. Fair, I. Gotlib, R. Poldrack, M. Saggar, A data-driven latent variable approach to validating the research domain criteria framework, Nature Communications 16 (1) (2025) 830.

[47] M. Tavasoli, A. Sarrafzadeh, M. Khaleghi, M. Zakaria, H. B. Pasandi, A. Karimoddini, Data communication challenges of connected and automated vehicles in rural areas, IEEE Access (2025).

[48] I. Khan, M. S. Haladappa, Dynamic resource reservation using deep reinforcement learning: Optimizing resource allocation for platoon-based c-v2x networks., Engineering Letters 33 (5) (2025).

[49] M. Testouri, G. Elghazaly, R. Frank, Robocar: A rapidly deployable open source platform for autonomous driving research, IEEE Intelligent Transportation Systems Magazine 17 (4) (2025) 83–95. doi:10.1109/MITS.2025.3546755.

[50] A. B. De Souza, V. H. C., B. Sikdar, Computation offloading for vehicular environments: A survey, IEEE Access (2020). doi:10.1109/ACCESS.2020.3033828.

[51] K. He, X. Zhang, S. Ren, J. Sun, Deep residual learning for image recognition, in: 2016 IEEE Conference on Computer Vision and Pattern Recognition (CVPR), 2016, pp. 770–778. doi:10.1109/CVPR.2016.90.

[52] T. Wang, X. Zhu, J. Pang, D. Lin, Fcos3d: Fully convolutional one-stage monocular 3d object detection, in: 2021 IEEE/CVF International Conference on Computer Vision Workshops (ICCVW), 2021. doi:10.1109/ICCVW54120.2021.00107.

[53] zlib, Compression, https://zlib.net/ (Accessed: June. 9, 2023).

[54] R. Meireles, F. Hawlader, A. Aguiar, Cloud-assisted perception - dynamic parameter selection evaluation, https://gitlab.com/rui-meireles/cloud-assisted-perception (2025).

[55] Y. Liu, T. Wang, X. Zhang, J. Sun, Petr: Position embedding transformation for multi-view 3d object detection, in: European conference on computer vision, Springer, 2022, pp. 531–548.

[56] S. Wang, Y. Liu, T. Wang, Y. Li, X. Zhang, Exploring object-centric temporal modeling for efficient multi-view 3d object detection, in: Proceedings of the IEEE/CVF international conference on computer vision, 2023, pp. 3621–3631.