State Consistent Edge-enhanced Perception for Connected and Automated Vehicles

Can Carlak*, Bo Yu†, Fan Bai†, Z. Morley Mao*
*University of Michigan, †General Motors

2024 IEEE 100th Vehicular Technology Conference (VTC2024-Fall) | 979-8-3315-1778-6/24/$31.00 ©2024 IEEE | DOI: 10.1109/VTC2024-Fall63153.2024.10757941

***Abstract*—Vehicle function offloading has been an active research topic in the connected and automated vehicles (CAV) domain. Mobile edge computing can execute sophisticated algorithms on abundant computing resources, leading to superior accuracy for vehicle system state estimation. On the other hand, cellular network latency causes edge-computed information to be obsolete.**

In this paper, by focusing on camera-based object tracking applications, we develop a novel state fusion framework that not only achieves the benefits (enhanced detection accuracy) but also mitigates the disadvantages (unpredictable network latency) of edge computing. This is achieved by carefully managing the system state consistency between the vehicle onboard system and the remote edge system through our novel *backward-and-forward* algorithms. We evaluate our system by comparing our method with the edge-only and onboard-only counterparts through extensive empirical experiments. The presented framework improves the accuracy of camera-based perception by at least 2x compared to traditional techniques, with an average response time of 48.13 ms (strictly less than the mission-critical latency threshold of 100 ms [1]).

***Index Terms*—connected vehicles, vehicle detection, edge computing, vehicle-to-infrastructure, real-time latency mitigation.**

I. Introduction

Modern vehicles, especially CAV, have an ever-increasing demand for computing and storage resources. These compute and storage resources are traditionally equipped on the vehicle's onboard computing platform, supporting resource-demanding tasks, such as perception, mapping, localization, and driver assistance. The onboard-only approach, nonetheless, is becoming less sustainable due to the growing cost and complexity of onboard hardware as well as the procreation of vehicular applications via post-sale over-the-air (OTA) updates. To tackle this issue, a recent research trend is to offload automotive functions to cloud or edge computing systems [2]–[4]. The emerging edge computing with abundant compute resources is expected to maintain its presence for automotive mission-critical applications. In the case of camera-based vehicle perception, more sophisticated machine-learning models can be utilized on edge-computing servers to provide improved object detection service. This model enhances vehicle perception without increasing the hardware complexity on the vehicle side.

A major challenge of vehicle function offloading is the unpredictable network latency. Using camera-based object tracking as a use case, this process typically involves three steps: 1) streaming vehicle sensor data to the edge via cellular networks; 2) processing the sensor data on edge servers; and 3) transmitting the results back to the vehicle via cellular networks. As a consequence, significant latency can arise during this lengthy process, posing a major challenge to mission-critical automotive applications because it may render the system states computed by the edge servers obsolete.

To address this system state inconsistency problem, we propose a novel *state fusion* framework to reconcile the two system states (local copy vs. remote edge copy) as we offload vehicle functions to the edge. Our design mitigates the impact of round-trip network latency in automotive offloading cases. The algorithm uses a state history data structure to maintain the system states of both the onboard vehicle and remote edge system via buffering the perception results in the past few seconds. When edge results are received by the vehicle, we first roll back the state history and use the edge results to correct the estimation error in the corresponding local state, followed by the fast-forward method to integrate up-to-date local sensors and thus predict the best-estimated state for the current moment. We call this *backward-and-forward algorithm.*

To study the behavior of our proposed framework, we have conducted extensive empirical experiments using real-world 5G networks and commercial edge computing environments. Our measurement results show that perception errors resulting from round-trip latency can be significantly reduced, through the mitigation strategies proposed in our systems. Compared to onboard-only and edge-only solutions, we can deliver at least 2x better accuracy for estimating the target's distance. Moreover, the average system response time is kept below 50 ms regardless of underlying network latency fluctuation. Evaluated under several network latency profiles, the proposed system provides strictly better accuracy for camera-based object tracking.

The major contributions presented in this paper are as follows:

- We developed a vehicle-edge integrated system to offload vehicle functions, and validated it with an automotive perception application;
- We designed a state fusion algorithm to mitigate the effects of round-trip latency from edge computing.
- We carried out extensive real-world experiments using commercial 5G networks and cloud computing providers to gather empirical data on edge-enhanced vehicle function offloading.

II. Edge-enhanced Vehicle Perception System

In this section, we present a comprehensive description of our system architecture designed to augment vehicle perceptual capabilities via edge computing. We break down the computation phases within the perception pipeline, detailing how each stage contributes to the refinement and enhancement of sensory input for improved decision-making in edge computing scenarios.

A. System Design

More accurate object detection algorithms can be executed on sensor data by offloading computing to powerful edge servers. *The problem*, however, is unpredictable network latency from the edge and cloud services often render results obsolete, leading to inconsistent states between local and edge components. Ensuring the seamless operation of mission-critical systems such as CAV demands real-time response from computational systems. Hence, we prioritize *reliable real-time response* as an important system design requirement, in the presence of network instabilities. Furthermore, the large amounts of sensor data generated by the *dense traffic* of metropolitan cities necessitate an infrastructure capable of processing such data efficiently. Therefore a *scalable* architecture emerges as another crucial design requirement alongside real-time responsiveness. Additionally, *flexibility in data formats and communication patterns* is essential to accommodate future advancements, allowing for the seamless integration of additional sensor data beyond camera images. This adaptability not only enhances current capabilities but also future-proofs the system, ensuring its relevance and efficacy in evolving environments.

Key Insight: Our work combines the strengths of local/onboard perception and edge computing, aiming to balance their benefits and limitations. On-board systems are prized for their *faster response* times, as they process data internally without the need for Internet communication. These low-latency systems are critical for real-time responses. However, they usually employ simpler detection algorithms leading to less robust target detection capabilities. On the contrary, edge servers, though suffering by *network latency*, offer *superior accuracy* due to more sophisticated algorithms. This presents a bottleneck since the data they produce, while more accurate, may no longer reflect the current state of the environment by the time they are received back by the vehicle.

Throughout this manuscript, we adopt the term *world state* to describe the collective knowledge possessed by an ego vehicle regarding its immediate environment. In a pragmatic sense, the term encapsulates relevant data structures utilized for the estimation of current spatial coordinates of the target vehicle(s) within the vicinity. Hence, one important performance metric for our vehicle detection and tracking system resides in the *accurate* estimation of the *world state* while satisfying the system requirements described in this section.

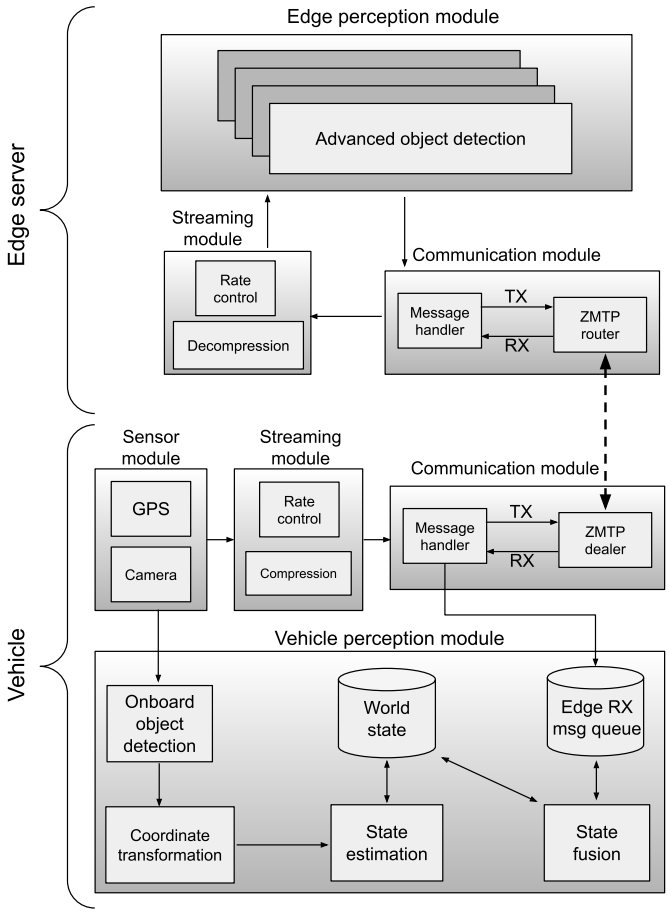

Fig. 1: Edge-enhanced vehicle perception system.

B. Data Flow and Perception Phases

Within the proposed framework, the synthesis of the *world state* information is realized through the systematic processing and refinement of historical sensor data, in conjunction with the processing within the edge server. The subsystem at the bottom in Figure 1 illustrates the principal components associated with vehicle-side computing, whereas the upper counterparts represent the edge service. To understand the system behavior thoroughly, we start explaining how the data flows; starting at the sensor module, wherein an onboard camera apparatus generates image frames.

The image frames traverse two distinct paths of processing: i) local onboard perception path, and ii) edge perception path. In the localized onboard perception pathway, each image frame undergoes a sequence of intermediary operations comprising target vehicle detection, coordinate transformation, target vehicle tracking via Bayesian filtering, and updating the ego vehicle's world state. Conversely, the secondary processing pathway of our system involves the compression, rate control, and transmission of data messages to an edge server via a cellular network. The frames are decompressed and processed by edge-perception modules for notably improved accuracy compared to a local-only detector. Upon completion of the edge server processing, the results accompanied by unique

identifiers are relayed back to the vehicle for correlation with locally computed counterparts. Subsequently, during the state-fusion phase, the *"backward-and-forward"* (section III-B) algorithm intelligently merges the edge-processed data and the locally computed data, using state history. This fusion yields superior target vehicle positioning accuracy coupled with real-time response; conforming to our primary system requirements in section II-A.

To address the remaining system requirements in section II-A, each major processing block is designed and implemented in the form of microservices for achieving higher horizontal scalability, and flexibility standards: i) The Sensor Module collects raw measurements from onboard hardware – camera and GPS data for the scope of this work, yet provides an extensible interface for future. ii) The Streaming Module encodes raw frames using H264 and employs adaptive streaming to optimize bandwidth and reduce packet loss; thereby ensuring continuous and reliable data flow. iii) The Communication Module implements a ZMQ [5] dealer/router communication model providing asynchronous messaging between the vehicle and the edge server. This pattern allows vehicles (dealer sockets) to send multiple requests before waiting for replies, allowing for higher throughput and efficiency. Also, vehicles can join and leave the network with minimal coordination or disruption; allowing for easy scaling of the edge-enhanced perception service, making our system highly flexible.

C. Object Localization and World State Mapping

Upon successful object detection phase, each target vehicle is enclosed in a bounding box with pixel coordinates in the camera's perspective. However, these coordinates are not directly applicable to *world state* computations. Therefore, we perform a 4-point perspective transformation that maps these pixel coordinates onto a bird-eye view of the scene using a transformation matrix $\mathbf{M}$.

For a point P represented in pixel coordinates as $\mathbf{p} = [x, y, 1]^T$, the mapping is defined using the transformation matrix $\mathbf{M}$ as:

$$\mathbf{p}' = \mathbf{M} \cdot \mathbf{p}$$

where $\mathbf{p}'$ represents the corresponding point in bird's-eye view coordinates. The matrix $\mathbf{M}$ is a 3x3 matrix composed of rotational $[r_1, r_3]^T[r_2, r_4]^T$, translational $[t_1, t_2, 1]^T$, and projectional $[c_1, c_2, 1]$ components accounting for camera orientation and position. This matrix provides the homography that transforms image coordinates into a plane parallel to the road surface:

$$\mathbf{M} = \begin{bmatrix} r_1 & r_2 & t_1 \\ r_3 & r_4 & t_2 \\ c_1 & c_2 & 1 \end{bmatrix}$$

The components of $\mathbf{M}$ are calculated through a calibration process where four points of known pixel coordinates are mapped to their corresponding positions in the bird's-eye view frame. A known distance in the calibration image is also recorded to enable the computation of real-world distances after transformation. This process is conducted once during the calibration phase.

Limitations and Stability: Two critical limitations of the 4-point perspective transformation are its vulnerability to z-axis distortion and the need to re-calibrate if the camera pose changes. Despite this, in mostly planar environments where roads lack significant elevation changes, this method maintains reliability in mapping positions from the camera image to the bird's-eye coordinate system. As our primary focus is on the impact of network latency on perception systems running on the edge, instead of the perception system itself, we do not delve further into image stabilization techniques. However, alternative methods like Structure from Motion or SLAM (Simultaneous Localization and Mapping) could be explored for enhancing global coordinate estimation from camera perspectives.

III. Vehicle State Consistency

In this section, we discuss the two critical components of the vehicle hybrid perception system – state estimation and state fusion phases, respectively.

A. State Estimation

Perspective pixel coordinates describing a bounding box undergo transformation to a bird-eye representation, corresponding to actual spatial coordinates. Nevertheless, these coordinates are not used directly for the true location of a target vehicle, due to the tendency of object detection algorithms to generate false positives. Instead, they serve as inputs to a state prediction/estimation algorithm to facilitate a more consistent representation of the world state. For state estimation, we employ particle filtering [6] — an approach prevalent in robotics and other domains for localization tasks.

To employ particle filtering for target position estimation using state variables for position and velocity in the x and y axes, we first define the state vector and then describe the process and measurement models.

State Vector: Let's x be a state vector at time step k as:

$$x_k = [x_k, y_k, dx_k, dy_k]^T$$

where: - x_k and y_k represent the target's position on the x and y-axis, respectively, at time step k. - dx_k and dy_k represent the target's velocity on the x and y-axis, respectively, at time step k.

Process Model: A linear motion model with a Gaussian process noise is assumed; therefore, the state transition matrix can be written as:

$$F = \begin{bmatrix} 1 & 0 & \Delta t & 0 \\ 0 & 1 & 0 & \Delta t \\ 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix}$$

where Δt is the time difference between the current state and the next state.

Measurement Model: The measurement model describes the relationship between the state vector and the measurements received. We can directly measure the position (via image pixel coordinates) but not velocity, therefore, the measurement matrix H:

$$H = \begin{bmatrix} 1 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 \end{bmatrix}$$

The measurement equation would then be:

$$z_k = Hx_k + v_k \tag{1}$$

where: - z_k is the measurement at time step k (it contains the measured x and y positions). - v_k is the measurement noise, which is modeled as zero-mean Gaussian.

Step 1 - Initialization: Generate N particles $\{x_0^{(i)}\}_{i=1}^N$ from the prior distribution, which represents the initial belief about the state of the system.

Step 2 - Prediction: For each particle, predict the next state based on the process model:

$$x_{k+1|k}^{(i)} = Fx_k^{(i)} + \eta_k^{(i)} \tag{2}$$

where $\eta_k^{(i)}$ represents the process noise, which is the stochastic disturbance that affects the state transition. It is sampled from a Gaussian distribution.

Step 3 - Measurement Update: Once a new measurement z_{k+1} is observed, the weights of each particle are updated based on how likely the measurement is, given the predicted state:

$$w_{k+1}^{(i)} = p(z_{k+1}|x_{k+1|k}^{(i)}) \tag{3}$$

The weights are determined by evaluating a Gaussian probability density function centered at the predicted measurement for each particle.

Step 4 - Resampling: Particles with higher weights are more likely to be selected. This step results in a new set of particles $\{x_{k+1}^{(i)}\}_{i=1}^N$ that represent the posterior distribution reflecting the latest measurement.

$$x_k^{(i)} \; p(x_k|z_k) = \sum_{j=1}^{N} w_k^{(j)} * \delta(x_k - x_k^{(j)}) \tag{4}$$

Step 5 - Estimation: The estimate of the state is computed using the particles and their weights. We use the weighted average:

$$\hat{x}_{k+1} = \sum_{i=1}^{N} w_{k+1}^{(i)} x_{k+1}^{(i)} \tag{5}$$

Each new sensor data triggers repeating the prediction, measurement update, resampling, and estimation steps for each time step. Lastly, the number of particles used can dramatically affect the performance of the perception pipeline. Typically using a large number of particles converges more reliably as more distinct particles can cover a bigger area in a search space, at a cost of slower processing. To balance accuracy and response time, we experimentally found that around 500 particles for representing the target vehicle state, provide the best performance.

B. State Fusion

So far, we explained how the sensor data goes through a series of operations to synthesize the world state. In this section, we explain state-fusion: the process of maintaining state consistency between a vehicle perception and an outdated edge perception. Once an edge perception result is received asynchronously; the *backward-forward algorithm* is triggered on the existing world state and newly received edge server result.

At any given time, multiple world states are likely to be already produced and inserted into the world states data structure by the local perception. The reasoning is that processing camera frames in the onboard perception subsystem is much faster than acquiring an edge server response for the same frame due to network delay. When the corresponding analysis from the edge is received at the vehicle, chronologically ordered world state history is searched to find the matching unique frame ID. Any obsolete states are discarded in the search. Subsequently, the system restores to the exact world state which includes the past timestamps, camera frames, and particles: we call this operation a *rollback*.

After a successful rollback, we can safely update the locally perceived past states with the robust edge perception result. Updating a past state is similar to state estimation processes discussed in section III-A; only this time the edge-computed measurements are incorporated in the estimation functions. At this point, a superior accuracy edge-computed world state is obtained despite being outdated. Similar to a rollback analogy but this time in the forward direction, we need to reconstruct intermediate world states. This step is referred to as *fast-forwarding*. Starting from the oldest to the most recent state, their corresponding local measurements are fed into state estimation. Note that, even though the prediction-update-resampling cycle of the particle filter is supplied with the recorded local-only data, now the starting points of the particles are enhanced with robust edge perception. Therefore, the reconstructed states become more reliable than the local-only replicas at every iteration of *fast-forwarding*. As long as there are intermediate states in the world states list, fast-forwarding is applied to each sequentially. The entire state-fusion pipeline is triggered every time new edge data is received. A pseudo-code is given in Algorithm 1 to describe the process.

IV. Evaluation

We empirically assess the performance of our system by conducting driving experiments under real network impairments.

A. Experiment Environment

To deploy our edge server, we worked with one of the major commercial services in the US: powering 8 CPUs, 56 GB memory, and 1 Nvidia Tesla T4 GPU. Given the computing resources, we run a full YOLO [7] instance under 60 ms prediction response time with an input resolution size of 608x608, up to 35 frames-per-second (FPS). For the on-board

Algorithm 1: Backward-Forward Algorithm

Input: $edgeData$: received edge perception result;
$worldStates$: list of chronological world states;

```
frameID ← edgeData.frameID
DiscardObsolete(worldStates, frameID)
matchState ← Rollback(worldStates, frameID)
if matchState ≠ null then
    p ← matchState.particles
    r ← edgeData.measRes // using edge
    newState ← StateEstimator(p, r)
    UpdateSt(worldStates, matchState, newState)
    // Fast-forward loop
    foreach state in worldStates do
        p ← state.particles
        r ← state.measRes // using local
        newState ← StateEstimator(p, r)
        UpdateSt(worldStates, state, newState)
```

system, a laptop device with a 4-core CPU (Intel Xeon E3-1505M), 32 GB of memory, and a Nvidia Quadro M1000M 2 GB graphics card is used. By direct comparison, deploying the full YOLO object detection model on the onboard setup yields either an unacceptably slow performance or a "not enough memory" exception. As expected, the highest FPS we obtained from the onboard computing system was 3.9, along with more than 250 ms latency. Since this performance is not practical to be deployed in real-world scenarios, a smaller footprint called YOLO-tiny, is preferred for our vehicular setup. With the lighter model, the on-board detector can process camera images up to an average rate of 10 FPS.

In order to evaluate the effectiveness of our edge-enhanced perception, we conducted a series of driving tests using two vehicles: the ego vehicle and the target vehicle. Both vehicles were equipped with UBlox RTK-GPS units to obtain ground-truth distance measurements. Additionally, the ego vehicle is equipped with a camera, and a USB tethered mobile phone for providing network access. The evaluation was performed using four different datasets including a vehicle test facility, and several driveways in Ann Arbor, Michigan. Each dataset presents unique challenges in terms of speed, network delay, and network stability.

Dataset-I: Both the ego vehicle and the target vehicle approach each other at a constant speed of 25 mph. The test was conducted on a straight test road designed for controlled tests. The area has very good network coverage, ensuring minimal network latency and stable connectivity throughout the tests. The main purpose is to evaluate and compare the initial detection distances achieved by local-only, edge-only, and edge-enhanced methods without the confounding effects of network impairments commonly encountered in real-world scenarios.

Dataset-II: This dataset involves the ego vehicle chasing the target vehicle in a residential area east of Ann Arbor. The driving speeds reached up to 50 mph, with network delays up to 120 ms and relatively stable connections with fewer network fluctuations.

Dataset-III: In this dataset, the ego vehicle chased the target vehicle along the North 23 highway and through the University of Michigan's North Campus in Ann Arbor. Speeds reached up to 70 mph, with network delays extending up to 900 ms, moderate packet loss, and frequent network fluctuations due to high-speed cellular handovers. A GPS-based trajectory is given in Figure 2.

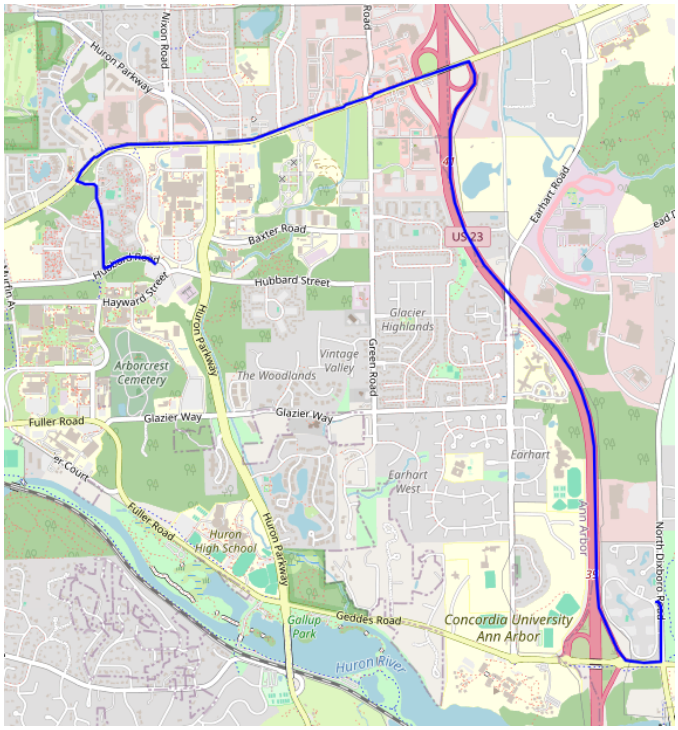

Fig. 2: Trajectory of dataset-III comprising University of Michigan North Campus and 23 North highway.

Dataset-IV: This dataset captures scenarios where the ego and target vehicles are approaching each other from opposite lanes on the University of Michigan's North Campus. Speeds reached up to 50 mph with network delays up to 1000 ms, very high packet loss, and occasional connectivity outages due to poor cellular coverage of a particular intersection.

B. Performance Analysis of the Edge-enhanced Perception

We define the following benchmarks:

- Edge-enhanced (EE): The proposed hybrid perception system is employed at the vehicles; leveraging both local and edge-computed perception via the backward-forward state fusion.
- Edge-only (EO): Edge perception is executed without performing latency mitigation techniques. Due to network latency, the perceptions are outdated once received at the ego vehicle.
- Local-only (LO): Local perception results are used without edge support. Note that the vehicle is equipped with

a less powerful GPU than an edge server, thus, only a lower-quality perception is available.
- Baseline: Vehicles leverage edge-computing grade perception under theoretically perfect network conditions (with zero latency). As an offline step, the edge-computing perception is directly applied to the traffic data we collected; resulting an upper bound for our evaluation.

First, we assess the performance metric of *vision* by evaluating the maximum distance at which the target vehicle is initially detected. Experiments in dataset-I shows that local-only (LO) perception detects the target vehicle at an average distance of 40 meters, while edge-computing methods detect it at an average distance of 70 meters. As illustrated in figure 3, EO and EE detection extends the maximum vision distance ranging from 90 to 95 meters in certain cases, effectively doubling the vision range of LO. These findings imply that offloading camera perception tasks to the edge substantially enhances the maximum detection distance.

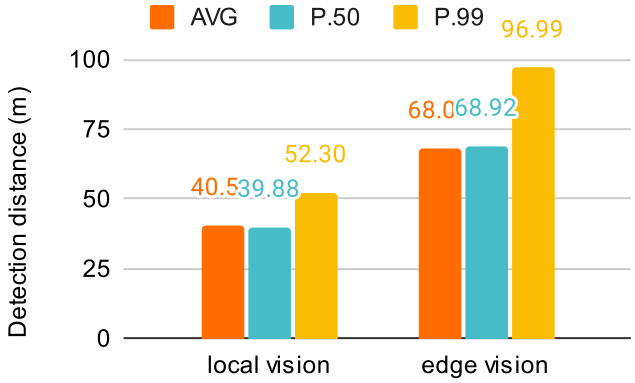

Fig. 3: Dataset-I: Maximum distances of initial detection.

Under conditions of moderate speed and stable network connections (dataset-II), both EE-2 and EO-2 methods performed similarly for the bulk of our measurements. However, in the higher error percentiles of Figure 5, EO-2 produced errors that were 1.5 times larger than those of EE-2. This illustrates that even minor network inconsistencies can have a significant adverse effect on edge-only perception and tracking.

High speed and frequent cellular handovers resulted in challenging conditions for EO. The CDF plot Figure 5 shows that EO-3 (dataset-III) experienced at least a 2x larger error in distance estimations in the 90th percentile and above. This gap widened even further when network latency reached extreme values (greater than 600 ms), highlighting EO-3's vulnerability under increased network stress, see Figure 4.

Dataset-IV presented the most demanding conditions, with very high packet loss and interrupted connectivity. The vehicles' relative speeds were also higher, magnifying the negative effects of network impairments. In this challenging scenario, EO-4 exhibited errors at least 4x greater than EE-4 for the highest percentiles as illustrated in Figure 5. This underlines EE's significant advantage in compensating for network-induced disruptions.

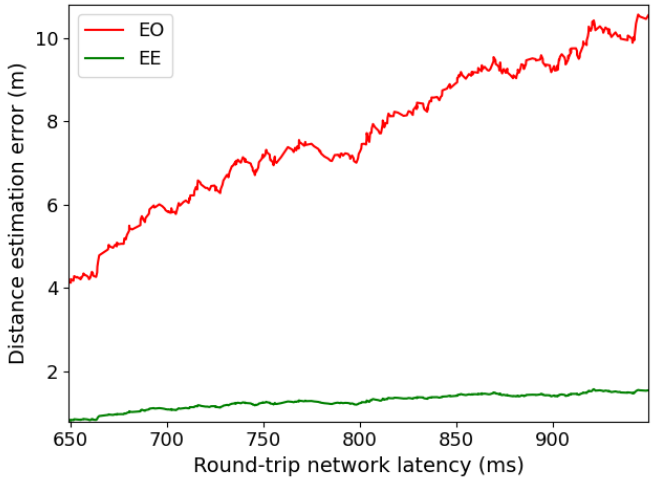

Fig. 4: Performance under extreme RTT in Dataset-III.

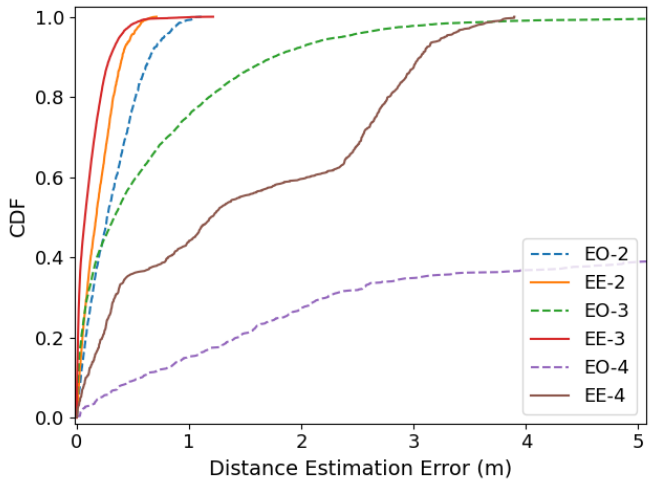

Fig. 5: Comparison of EE/EO-[2,3,4] in Dataset-[2,3,4].

Our comprehensive experimental evaluation, conducted across a series of uniquely challenging datasets, highlights the robust resilience of the proposed framework (EE) in high-speed vehicular contexts where mobile cell coverage, real-time performance during handovers, and network stability are critical factors. Edge-only (EO) methods consistently experienced performance degradation under conditions of high latency and frequent network fluctuations, while EE demonstrated superior adaptability - shown in Figure 4, effectively mitigating these adverse effects. Meanwhile, local-only (LO) methods exhibited faster detection times, its perception and tracking quality was significantly inferior compared to both edge-based techniques. Thus, our results suggest that EE is better suited for deployment in edge computing environments characterized by variable and often sub-optimal network conditions.

V. Related Work

A. Edge/Cloud Offloading

Research studies [8], [9], [10], and [2] have examined the application offloading to edge computing systems. Odessa's [11] method increased computer vision performance. In Outatime [12], speculative offloading achieved reduce latency. Despite similar techniques are covered, unique challenges in the CAV domain makes their work impractical.

[3] and [4] study offloading between multiple clients and edge servers to perform efficient load balancing, job scheduling, and content delivery services in the CAV domain. [13] does adaptive offloading on computer vision applications to cloud services from the vehicle. The study performs an estimation of network latency to decide whether to offload or not. Our work, however, benefits from edge computing resources even under a high network latency profile by the forward correction stage (state fusion). Lastly, accuracy is not the main goal of [13], but application response time is.

B. State Consistency

Studies such as [12] [14] [15] have taken the approach of sharing their applications' states across their clients and servers. In [14], the system requires sharing the internal states of a vehicle when it switches mode by propagating the state between a client and a cloud server. The study [15] proposed output-driven state preservation across vehicle and edge to balance the "accuracy vs network bandwidth" trade-off. The key insight is to use only the pose data (output of their main application) to reduce state complexity between vehicle and cloud copies. On the other hand, we maintain a global state only on the vehicle side in our system design. Therefore, the edge is solely responsible for running a stateless perception algorithm on the data it receives. The consistency of our system's state is conserved via state history, rollback and fast-forward stages upon reception of the edge result on the vehicle. In study [12], an edge server speculatively pre-computes data for an improved response time and quality at the client. In case the edge server makes an incorrect prediction, however, a rollback mechanism is executed to revert to a sound global state. To an extent, we have built on this idea, yet taken a reverse direction: In our design, we leverage an edge computed result of a near-past moment instead of a future speculation. Moreover, we consolidate the state consistency at the vehicle side instead of correcting at the edge as in their work.

VI. Conclusion

Offloading mission-critical automotive applications to an edge computing node is not a trivial task for CAVs, because of the technical challenge introduced by unreliable network conditions. In our study, we concentrated on the use case of camera-based perception and vehicle tracking, due to its widespread adoption in the CAV domain. A novel state-fusion system is presented to enable vehicles with limited computational power to benefit from advanced perception algorithms by intelligently incorporating edge-computed vehicle information with on-vehicle detection data. Experiments in a typical network with nearly 300 ms round-trip latency show that our "edge-enhanced" perception at least doubles the accuracy for vehicle position estimation compared to local-only and edge-only counterparts. While providing superior accuracy, edge-enhanced perception produces results as fast as local-only (real-time) perception; whereas edge-only methods drastically suffer from round-trip latency.

References

[1] S.-C. Lin, Y. Zhang, C.-H. Hsu, M. Skach, M. E. Haque, L. Tang, and J. Mars, "The architectural implications of autonomous driving: Constraints and acceleration," *SIGPLAN Not.*, vol. 53, no. 2, p. 751–766, Mar. 2018. [Online]. Available: https://doi.org/10.1145/3296957.3173191

[2] K. Kumar, J. Liu, Y.-H. Lu, and B. K. Bhargava, "A survey of computation offloading for mobile systems," *Mobile Networks and Applications*, vol. 18, pp. 129–140, 2013.

[3] J. Tang, R. Yu, S. Liu, and J.-L. Gaudiot, "A container based edge offloading framework for autonomous driving," *IEEE Access*, vol. 8, pp. 33 713–33 726, 2020.

[4] Q. Yuan, H. Zhou, J. Li, Z. Liu, F. Yang, and X. S. Shen, "Toward efficient content delivery for automated driving services: An edge computing solution," *IEEE Network*, vol. 32, no. 1, pp. 80–86, 2018.

[5] P. Hintjens, "Zeromq - the guide," http://zguide.zeromq.org/, accessed: [Apr 21, 2024].

[6] N. J. Gordon, D. J. Salmond, and A. F. M. Smith, "Novel approach to nonlinear/non-gaussian bayesian state estimation," *IEE Proceedings F - Radar and Signal Processing*, vol. 140, no. 2, pp. 107–113, 1993.

[7] A. Bochkovskiy, C.-Y. Wang, and H.-Y. M. Liao, "Yolov4: Optimal speed and accuracy of object detection," 2020.

[8] B.-G. Chun, S. Ihm, P. Maniatis, M. Naik, and A. Patti, "Clonecloud: Elastic execution between mobile device and cloud," in *Proceedings of the Sixth Conference on Computer Systems*, ser. EuroSys '11. New York, NY, USA: Association for Computing Machinery, 2011, p. 301–314. [Online]. Available: https://doi.org/10.1145/1966445.1966473

[9] E. Cuervo, A. Balasubramanian, D.-k. Cho, A. Wolman, S. Saroiu, R. Chandra, and P. Bahl, "Maui: Making smartphones last longer with code offload," in *Proceedings of the 8th International Conference on Mobile Systems, Applications, and Services*, ser. MobiSys '10. New York, NY, USA: Association for Computing Machinery, 2010, p. 49–62. [Online]. Available: https://doi.org/10.1145/1814433.1814441

[10] X. Gu, K. Nahrstedt, A. Messer, I. Greenberg, and D. Milojicic, "Adaptive offloading for pervasive computing," *IEEE Pervasive Computing*, vol. 3, no. 3, pp. 66–73, 2004.

[11] M.-R. Ra, A. Sheth, L. Mummert, P. Pillai, D. Wetherall, and R. Govindan, "Odessa: Enabling interactive perception applications on mobile devices," in *Proceedings of the 9th International Conference on Mobile Systems, Applications, and Services*, ser. MobiSys '11. New York, NY, USA: Association for Computing Machinery, 2011, p. 43–56. [Online]. Available: https://doi.org/10.1145/1999995.2000000

[12] K. Lee, D. Chu, E. Cuervo, J. Kopf, Y. Degtyarev, S. Grizan, A. Wolman, and J. Flinn, "Outatime: Using speculation to enable low-latency continuous interaction for mobile cloud gaming," in *Proceedings of the 13th Annual International Conference on Mobile Systems, Applications, and Services*, ser. MobiSys '15. New York, NY, USA: Association for Computing Machinery, 2015, p. 151–165. [Online]. Available: https://doi.org/10.1145/2742647.2742656

[13] A. Ashok, P. Steenkiste, and F. Bai, "Enabling vehicular applications using cloud services through adaptive computation offloading," in *Proceedings of the 6th International Workshop on Mobile Cloud Computing and Services*, ser. MCS '15. New York, NY, USA: Association for Computing Machinery, 2015, p. 1–7. [Online]. Available: https://doi.org/10.1145/2802130.2802131

[14] K. Sasaki, N. Suzuki, S. Makido, and A. Nakao, "Vehicle control system coordinated between cloud and mobile edge computing," in *2016 55th Annual Conference of the Society of Instrument and Control Engineers of Japan (SICE)*, 2016, pp. 1122–1127.

[15] K.-L. Wright, A. Sivakumar, P. Steenkiste, B. Yu, and F. Bai, "Cloud-slam: Edge offloading of stateful vehicular applications," in *2020 IEEE/ACM Symposium on Edge Computing (SEC)*, 2020, pp. 139–151.