Analysis of Field-Edge System Latency in Transport Monitoring Environment

Prof. Aistis Raudys, Lukas Baltramaitis and Robert Mackevič

“Taikomasis dirbtinis intelektas, UAB”, Vilnius, Lithuania

Abstract. This paper investigates the latency challenges in field-edge systems within transport monitoring environments, with a focus on collision detection and object tracking. Leveraging a field-edge network architecture, we explore the integration of image preprocessing at the field device level and complex object detection and tracking models at the edge device level. Our approach is focused on optimizing the balance between processing speed and accuracy, addressing the critical need for real-time responsiveness in dynamic scenarios such as smart city infrastructure and autonomous navigation. We build on the foundation of existing object detection algorithms, i.e. YOLO, and introduce a batch processing strategy with parallel threads to mitigate latency issues inherent in sequential processing frameworks. Our system is tested using a YOLOv8 model trained on a custom Traffic Object Detection dataset, demonstrating the ability to maintain high processing speeds, with performance exceeding 30 frames per second, without sacrificing accuracy. The approach combines advanced object detection and tracking algorithms with network optimization. Our system tests show positive results in reducing latency. This work sets a benchmark for the deployment of field-edge systems in transport monitoring and contributes to future advancements in smart transportation and autonomous systems.

Keywords: collision detection, edge, field, latency, object detection, object tracking, transport monitoring.

1 Introduction

In the rapidly advancing world of smart technologies, timely and reliable collision prediction could improve the safety of autonomous or other transport-oriented systems. Edge computing is one of the solutions in this context that could minimize latency and ensure sufficient accuracy. In this paper we analyze the use of edge devices specifically engineered for collision detection, presenting a field-edge architecture to optimize performance. Through our proposed architecture, this study aims to push the boundaries of current capabilities, addressing the urgent need for fast and accurate detection mechanisms in dynamic and complex environments.

2 Related work

In this section, we summarize related studies that address similar challenges: optimizing object detection models for low latency while maintaining acceptable accuracy; and splitting the detection into two physical modules: field and edge. Firstly, articles by Liu et al. [3] on SSD and Redmon et al. [4] on YOLO analyze the speed and accuracy trade-offs in object detection models. These studies present the potential of lightweight model architectures for real-time processing in various environments.

Secondly, research by Shi and Rajkumar [5] on edge computing frameworks for autonomous vehicles states the significance of distributing computational tasks between edge nodes and field devices to minimize latency. This concept is further researched by Wang et al. [6], who demonstrate the feasibility of improving performance using adaptive model partitioning between edge devices and centralized nodes.

Furthermore, studies on network optimization strategies, such as those by Howard et al. [2] on MobileNets, provide instructions for constructing efficient neural architectures that maintain high accuracy and save resources. The author aims to extend these discussions by proposing an architecture that reduces the high latency of complex object detection algorithms for real-time decisions. It addresses the challenges of maintaining real-time execution and ensuring the accuracy of collision prediction in complex environments.

To summarize, all studies mentioned above investigate either latency vs. accuracy questions or modular architecture for the object detection models.

3 Approach

3.1 Field-edge network architecture

Our field-edge network system architecture is designed to facilitate the efficient processing of video data in real-time applications. This architecture has two components: a field device responsible for initial video preprocessing, and an edge device that hosts more complex algorithms, including an object detection model, an object tracking algorithm, and specialized algorithms for applications, such as collision detection and identifying and tracking focal objects within a video frame (see Fig. 1).

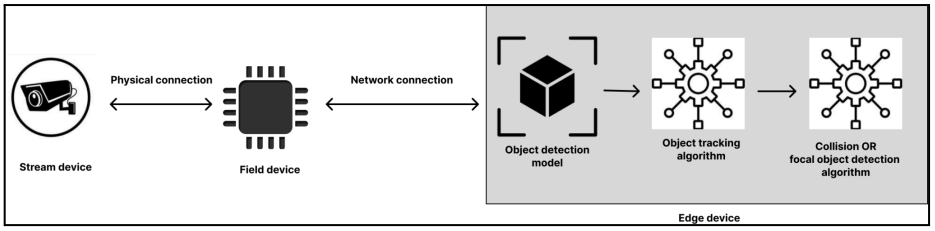

Fig. 1. Architecture components diagram.

- **Stream Device:** This device captures the video stream, which could range from a simple camera mounted on a drone to a security camera attached to a traffic light. For our testing, we focus on capturing traffic video data.
- **Field Device:** The field device preprocesses the video received from the stream device, optimizing it for further analysis, and then forwards it to the edge device.
- **Edge Device:** The edge device is divided into three main functions: the object detection model, the object tracking algorithm, and tracking delegate algorithms.

In the edge device, the object detection model processes the traffic video stream captured by the field device. This model, optimized for edge execution, quickly identifies multiple vehicles within the frame with high accuracy. After detection, an object tracking algorithm integrates with the object detection model to maintain consistent vehicle identification, crucial in fast-moving or crowded scenarios.

The system employs two selectable algorithms: collision detection and focal object tracking. These algorithms showcase the system's ability to enhance the total processing latency of the edge device. The collision detection algorithm analyzes tracked vehicles to predict potential collisions. In contrast, the focal object tracking algorithm identifies the largest vehicle in the frame and follows it over time, providing its center point position until the vehicle exits the frame.

Reducing latency in this system was the most important. A single field device, such as a Raspberry Pi, lacks the computational power to independently perform all required analyses. Additionally, real-time feedback is critical for controlling PTZ cameras or triggering alerts upon detecting potential collisions. This feedback loop allows timely responses to dynamic scenarios, emphasizing the need for the system to operate at speeds exceeding 30 frames per second. Operating at such speeds ensures continuous frame processing without skipping, essential for maintaining prediction accuracy and supporting real-time decision-making.

Our field-edge network system, with its intelligent integration of stream processing and robust field devices, is suited for applications demanding quick situational assessments and responses, such as smart city infrastructures, autonomous navigation systems, and security operations.

3.2 Latency problem and solution

In the current configuration, our system had a sequential processing workflow: an initial model in the field device processes an input frame before transferring it to the edge device with the more complex model for further analysis. We used the following notation for the theoretical latency problem analysis. The time required for the first model to process a single frame and transfer it to the edge model was marked as (*a)* milliseconds. While time to process the frame in the edge model was marked as (*b*) milliseconds and ($b > a$).

To simplify the calculations we assigned ($a = 1$), and defined the ratio ($b / a = k$), which represents how many times longer the processing time of the second model is compared to that of the first. Based on this setup, the theoretical latency (L) when processing the (n)-th frame can be calculated using the formula:

$$L = 2 + (k - 1) \cdot n \tag{1}$$

This formula (1) indicates a linear relationship between latency and the number of processed frames. This results in the problem that after some time the latency can become too large for the real-time processes.

To address potential scalability and efficiency challenges, we introduce the concept of batch processing for the object detection model. By incorporating (m) parallel threads in the edge device, the latency formula becomes:

$$L = [(n - 1) \div m] \cdot (k - m) + k + 1 \tag{2}$$

The strategy to mitigate latency issues involves increasing (m) to approximate (k) (referring to the 2nd formula), thereby optimizing the system's performance by aligning the processing capabilities of the parallel nodes with the complexity of the second model. This approach aims to effectively balance the workload and minimize processing delays, providing a scalable solution to handling increased frame rates within our system.

3.3 Custom traffic object detection dataset

To train the necessary object detection models for our solution, a custom dataset was created called the Traffic Object Detection dataset. This dataset comprises images capturing city roads and intersections sourced from diverse outlets, with a primary emphasis on leveraging data from prominent sources such as JUDU's live traffic feed cameras accessible on their website. The dataset for now is only focused on detecting one class of vehicles: cars. However, in the future, there are plans to expand the dataset to encompass a variety of vehicles. The general dataset properties can be seen in Table 1. Object position and size distributions are presented in Figure 2.

Table 1. Dataset properties.

Subset	Images	Positive samples	Negative samples	Number of objects (instances)	Max objects in the image	Number of objects per positive sample
Train	712	640	72	2155	26	3.367
Validation	88	80	8	303	15	3.788
Test	88	80	8	277	15	3.462

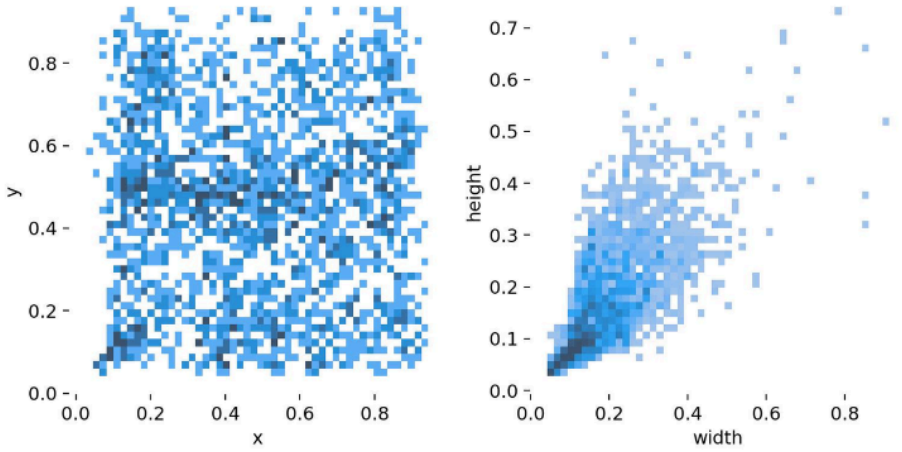

Fig. 2. Left graph shows a scatter plot of object center point positions, where x and y correspond to the image width and height pixels normalized to the scale [0, 1]. The right graph shows the object size scatter plot, normalized to the scale [0, 1].

4 Results

4.1 Object detection model

The YOLOv8 car detection model was trained on the Traffic Object Detection dataset. The properties of the trained model are outlined in Table 2. The training process spanned 100 epochs and was stable as shown by the consistently declining loss graphs presented in Figure 3.

Table 2. Model properties.

Pre-trained weights	YOLOv8x
Number of parameters	68,153,571
Size on disk	130.4 MB
Input image resolution	640x640
Number of classes	1

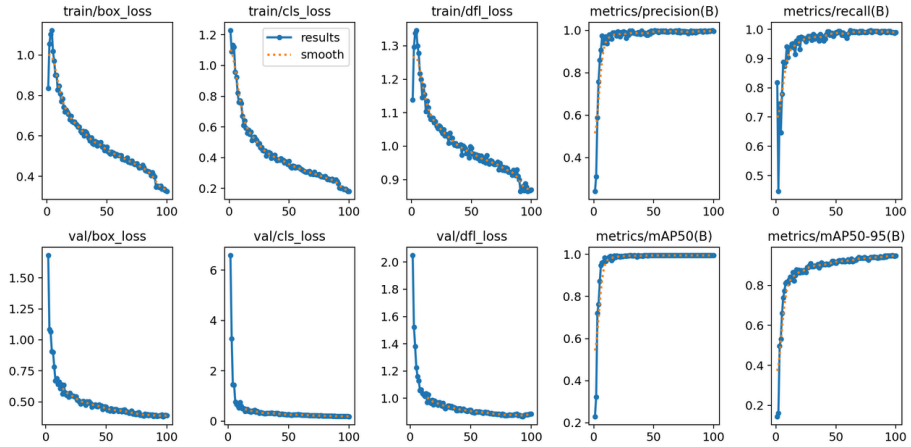

Fig. 3. Training and validation loss and metrics graphs.

Table 3. Model evaluation.

Subset	Images	Instances	Precision	Recall	mAP50	mAP50-95
Train	712	2155	0.996	0.990	0.995	0.945
Validation	88	303	0.996	0.990	0.995	0.950
Test	88	277	0.999	0.993	0.995	0.926

Final model evaluation was conducted on all 3 subsets of the dataset with the detection confidence threshold set to 50% and the IoU threshold set to 60%. The evaluation results are shown in Table 3. Although model accuracy was not the primary focus of this setup, it's noteworthy that the model exhibits impressive performance on unseen data, evident in the mAP50-95 score of 0.926 achieved on the testing subset. This evaluation establishes an accuracy baseline, providing a reference point for assessing the impact of future modifications that might be made to enhance the model's efficiency, such as model quantization.

4.2 System performance

Edge node hardware:

- CPU: i9-11900KF

- GPU: NVIDIA GeForce RTX 3060
- RAM: 32GB

Field node hardware:

- Raspberry Pi 4 Model B 2GB Version

Table 4. Testing video properties.

Filename	Resolution	Duration, min:sec	FPS	Frames
video_1.mp4	852×480	00:09	30	276
video_2.mp4	852×480	00:43	30	1312

The system underwent a testing procedure utilizing two videos featuring busy intersection traffic, aimed at assessing the efficiency of frame processing. The properties of both testing videos are outlined in Table 4. Notably, these videos differ in duration, with one being shorter than the other. This intentional discrepancy enables a dynamic evaluation of system performance over time. An adverse impact on performance with the longer video would signify the system's inability to process frames at a rate equivalent to their capture, resulting in a potential backlog accumulation over time somewhere in the system.

Before conducting the system test, a decision was made regarding the selection of the tracking algorithm, with the choice narrowed down to BoT-SORT [1] and ByteTrack [7]. In pursuit of efficiency, the decision was made to opt for the faster of the two algorithms. Subsequently, an additional latency test was performed, aiming to compare the processing speed of both BoT-SORT and ByteTrack algorithms. The latency comparison result between both algorithms is shown in Table 5. ByteTrack is seen as the faster of the two.

Table 5. BoT-SORT and ByteTrack latency comparison.

Latency	BoT-SORT	ByteTrack
AVG, ms	8.165	**0.941**
STD, ms	1.129	**0.869**

Further assessments were imperative to gauge the latency of tracking delegates: collision detection and focal object tracking. Given the system's capability to operate in two distinct modes, it became imperative to ascertain which mode exhibited lesser efficiency. This determination facilitated the selection of the less efficient mode for subsequent system testing, allowing for an in-depth analysis of system performance under more demanding operational conditions. The latency comparison result between both modes is shown in Table 6. Collision detection was deemed less efficient.

Table 6. Collision detection and focal object tracking latency comparison.

Latency	Collision detection	Focal object tracking
AVG, ms	2.780	**0.190**
STD, ms	1.515	**0.244**

For the final system performance test, it was decided to use the faster ByteTrack algorithm and the slower, more demanding collision detection mode, as evidenced by Table 5 and Table 6 respectively. The total work time of the system was tracked, spanning from the acquisition of the first frame from the video capture until the reception of the results of the final frame from the edge node. This test was conducted using both video files outlined in Table 4. The subsequent processing speed results of the system are detailed in Table 7.

Table 7. System performance test results.

Filename	System work time, sec	Processing rate, FPS
video_1.mp4	8.621	32.014
video_2.mp4	37.300	35.174

The system performance results presented in Table 7 underscore the system's capability to process frames at a speed surpassing 30 FPS. Specifically, the testing outcomes with a shorter video reveal a processing rate of approximately 32 FPS, while those with a longer video demonstrate a processing rate of around 35 FPS. Importantly, these findings show that the system maintains its efficiency over time without losing the performance under load. Consequently, these results suggest the feasibility of deploying such a system in production-level environments, handling 30 FPS real-time camera footage with acceptable latency.

5 Discussion

The development and evaluation of our field-edge network architecture, which includes a field device for image preprocessing and an edge device equipped with advanced algorithms for object detection, tracking, and collision or focal object detection, contributes to other innovations in real-time video analytics for smart city and security applications.

Our system's architecture is designed to handle the challenges of processing large volumes of video data in real-time. We integrated an optimized YOLOv8 model for car detection, together with advanced tracking algorithms such as BoT-SORT and ByteTrack, achieving high precision and efficiency in object identification and tracking. This is proved by the model's performance metrics, which show sufficient accuracy across training, validation, and testing datasets (mAP50-95 score of 0.926). High accuracy is crucial for reliable object detection and tracking in real-world scenarios, particularly in applications like traffic monitoring and autonomous navigation.

Reduced latency was the most important in our system's development due to the real-time processing in the intended use cases. Our initial sequential processing framework had potential scalability issues, so we moved to a batch processing approach with parallel threads. This change significantly reduced latency, as demonstrated by test results. The system also maintained high processing speeds even with increasing video data volumes. Our system performance tests showed the ability to process frames at a 30 FPS rate.

We chose the ByteTrack algorithm in the final system configuration because it was faster than the BoT-SORT. Furthermore, the system's architecture achieves a balance between hardware utilization and algorithmic efficiency. By employing edge devices equipped with powerful CPUs and GPUs, alongside lightweight field devices like the Raspberry Pi 4, we managed to optimize the performance.

6 Conclusion

In conclusion, our field-edge network system is a step forward in the application of edge computing and real-time video analytics. After analyzing hardware and software components, we developed a high-speed, accurate object detection and tracking system. In the future, we plan on further reducing latency, improving flexibility, and adding another software module that would allow for real-time monitoring and visualization of the system processes.

References

1. Aharon, N., Orfaig, R., Bobrovsky, B.-Z.: BoT-SORT: Robust Associations Multi-Pedestrian Tracking. arXiv preprint arXiv: arXiv:2206.14651 (2022).
2. Howard, A.G., Zhu, M., Chen, B., Kalenichenko, D., Wang, W., Weyand, T., Andreetto, M., Adam, H.: MobileNets: Efficient Convolutional Neural Networks for Mobile Vision Applications. arXiv preprint arXiv:1704.04861 (2017).
3. Liu, W., Anguelov, D., Erhan, D., Szegedy, C., Reed, S., Fu, C.-Y., Berg, A.C.: SSD: Single Shot MultiBox Detector. In: Leibe, B., Matas, J., Sebe, N., Welling, M. (eds.) ECCV 2016, LNCS, vol. 9905, pp. 21–37. Springer, Cham (2016).
4. Redmon, J., Divvala, S., Girshick, R., Farhadi, A.: You Only Look Once: Unified, Real-Time Object Detection. In: Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition, pp. 779–788. IEEE, Las Vegas (2016).
5. Shi, W., Rajkumar, R.: Edge Computing: Vision and Challenges. IEEE Internet of Things Journal 3(5), 637–646 (2020).
6. Wang, S., Xu, J., Zhang, N., Liu, Y.: A Survey on Service Migration in Edge Computing. IEEE Access 6, 23511–23528 (2018).
7. Zhang, Y., Sun, P., Jiang, Y., Yu, D., Weng, F., Yuan, Z., Luo, P., Liu, W., Wang, X.: ByteTrack: Multi-Object Tracking by Associating Every Detection Box. arXiv preprint arXiv: arXiv:2110.06864 (2022).