Real-Time Video Analytics for Urban Safety: Deployment over Edge and End Devices

Mahshid Ghasemi[1], Yongjie Fu[2], Xinyu Ouyang[1], Peiran Wang[1], Mehmet Kerem Turkcan[1], Jhonatan Tavori[3], Sofia Kleisarchaki[4], Thomas Calmant[4], Levent Gürgen[4], Zoran Kostic[1], Xuan Sharon Di[2], Gil Zussman[1], Javad Ghaderi[1]

[1]Electrical Engineering, Columbia University, [2]Civil Engineering, Columbia University, [3]Computer Science, Tel Aviv University, [4]Kentyou

{mahshid.ghasemi,yf2578,xo2122,peiran.wang,mkt2126,zk2172,sharon.di,gil.zussman,jg3465}@columbia.edu, jhonatant@mail.tau.ac.il,{sofia.kleisarchaki,thomas.calmant,levent}@kentyou.com

Abstract

This paper introduces PAVE (Pedestrian Awareness Via Edge analytics), a scalable real-time video analytics system that uses street cameras to enhance pedestrian safety while preserving their privacy. PAVE processes live camera streams on an edge server to track pedestrians and vehicles in real-time, predict vehicles' trajectories, and identify danger zones where pedestrians are present. The coordinates of these zones are sent to pedestrians' mobile devices via a custom iOS app, which locally determines if they are at risk without sharing any data with the edge server, hence preserving privacy. Moreover, anonymized metadata, including real-time location and speed/direction of pedestrians and vehicles, are visualized on a public map. PAVE's effectiveness was validated through deployment on the NSF COSMOS testbed, processing live video from cameras in diverse urban environments. Live field tests show that PAVE can alert at-risk pedestrians ~0.9 s before a vehicle reaches them. Through extensive profiling, we show that optimizing memory/compute configuration per pipeline stage can reduce latency by up to 10× compared to the default operating system configurations.

CCS Concepts

• **Computing methodologies** → **Distributed computing methodologies**; • **Computer systems organization** → **Real-time system architecture**.

SEC '25, Arlington, VA, USA

ACM ISBN 979-8-4007-2238-7/2025/12
https://doi.org/10.1145/3769102.3770618

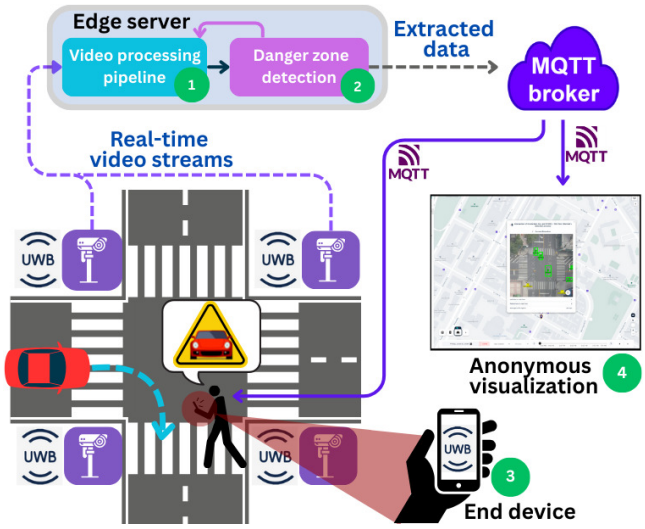

Figure 1: PAVE's workflow: The video streams from all cameras are analyzed and integrated into edge servers, and the obtained data is sent to an anonymous visualization map as well as to end-users' mobile phones to alert the pedestrians, if necessary.

Keywords

Video analytics, edge computing, testbed deployment.

ACM Reference Format:
Mahshid Ghasemi[1], Yongjie Fu[2], Xinyu Ouyang[1], Peiran Wang[1], Mehmet Kerem Turkcan[1], Jhonatan Tavori[3], Sofia Kleisarchaki[4], Thomas Calmant[4], Levent Gürgen[4], Zoran Kostic[1], Xuan Sharon Di[2], Gil Zussman[1], Javad Ghaderi[1] . 2025. Real-Time Video Analytics for Urban Safety: Deployment over Edge and End Devices. In *The Tenth ACM/IEEE Symposium on Edge Computing (SEC '25), December 3–6, 2025, Arlington, VA, USA*. ACM, New York, NY, USA, 17 pages. https://doi.org/10.1145/3769102.3770618

1 Introduction

We investigate how real-time video analytics can be used to enhance road users' safety while preserving their privacy.

We present PAVE (**P**edestrian **A**wareness **V**ia **E**dge analytics), which is a scalable video analytics system (illustrated in Fig. 1) distributed across edge servers and end-users' devices (e.g., iPhones). PAVE processes live video streams from traffic cameras to determine if there are any pedestrians in a dangerous area where a vehicle is approaching and triggers timely alerts on their mobile devices. The computationally intensive video processing components run on the GPU-equipped edge server, while the lighter components are executed on end-users' mobile devices. Additionally, extracted traffic/crowd-related metadata are aggregated and visualized anonymously through an interactive map developed for both real-time and historical monitoring.

Previous research has significantly advanced vision models for tasks such as object detection and tracking [3, 54]. However, practical challenges in deploying these models at scale in realistic urban settings remain insufficiently explored. In this work, we mainly focused on designing system architecture and optimizations that enable existing vision models to be deployed effectively in real-world applications and a realistic evaluation of the system. Specifically, we investigated and addressed some of the complexities inherent in deploying distributed video analytics systems for time-sensitive applications, such as pedestrian safety. We implemented and evaluated PAVE in the NSF COSMOS testbed [18], processing real-time video feeds from connected cameras.

Workflow. As illustrated in Fig. 1, PAVE's real-time workflow consists of the following steps:

(1) *Video Processing Pipeline (light blue box):* Real-Time Streaming Protocol (RTSP) video streams are transmitted to edge servers, decoded into raw frames, and processed by object detection (e.g., YOLOv8 [72]) and tracking (e.g., NVIDIA DCF) models. The pipeline outputs object bounding boxes, labels (pedestrian, car, truck), and tracking IDs, which are converted to a unified top-view to estimate vehicle and pedestrian speeds and directions.

(2) *Danger Zone Detection (light purple):* Vehicles' future trajectories are predicted using a Kalman Filter (KF)-based model to identify potential danger zones, particularly areas such as crosswalks where pedestrians might be at risk (an example is depicted in Fig. 2). Identified danger zones trigger a targeted Region Of Interest (ROI) analysis within the video processing pipeline to detect pedestrian presence with high accuracy. If pedestrians are detected, danger zone information is communicated through Message Queuing Telemetry Transport (MQTT) protocol [43, 53] to mobile devices via the iOS app.

(3) *End Device (iOS App):* The mobile app determines whether the pedestrian is (going) inside an identified danger zone using Ultra-Wideband (UWB)-based localization, subsequently issuing timely warnings, if necessary. To ensure the alert effectively captures the user's attention even when the phone is in silent or Do-Not-Disturb mode, the app leverages iOS Critical Alerts, which can bypass these restrictions, significantly improving the likelihood of capturing user attention.

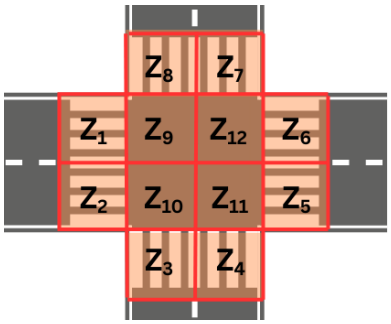

Figure 2: Predefined danger zones on areas near crosswalks where pedestrians might be present.

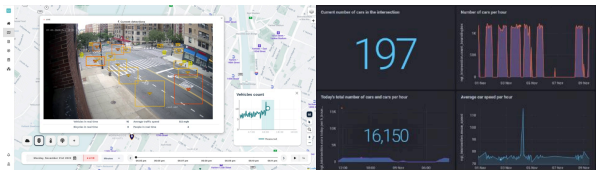

Figure 3: Visualization map interface, displaying real-time data collected from a camera in NYC (left), and a dashboard providing historical extracted data (right).

(4) *Visualization Dashboard:* Extracted metadata, including pedestrian/vehicle locations, speeds, and directions, are continuously sent to a visualization map via the MQTT protocol for anonymized display. This map, which is publicly accessible, presents only moving bounding boxes and associated metadata, without exposing identifiable images of pedestrians or vehicles. A sample snapshot of the interface of this map is shown in Fig. 3.

We used traffic cameras in New York City (NYC), Florida, and Turkey to evaluate the performance of the system in various urban environments. Fig. 4 shows the location and viewpoint of two of these cameras in NYC.

Ethical consideration. All recorded and live video camera feeds that we used were either publicly available or obtained with appropriate permission. For example, our use of COSMOS' live and recorded video streams [9, 82] was designated IRB-exempt by Columbia University.

1.1 Challenges

Deploying end-to-end real-time video analytics systems at scale, particularly for time-sensitive and safety-critical applications, presents significant practical complexities. Key challenges encountered during the development and deployment of such systems include:

Figure 4: Cameras deployed on 1st and 2nd floor of a building and their view of an intersection in NYC.

Intensive computational demands: Real-time video analytics, such as object detection, require substantial computational resources. This necessitates careful model optimization (e.g., via layer fusion, precision calibration, and kernel auto-tuning [37, 58]) and adjusting video frame rate and resolution [38] to reduce latency and overhead and thus ensure practical deployment.

Pipeline integration complexities: Efficiently integrating the various components of a video analytics pipeline (an example is illustrated by the data flow in Fig. 5) is crucial. This involves overcoming significant data transfer overhead between processing stages (e.g., CPU-GPU memory copies, and data transfer across network interfaces in distributed systems) and mitigating buffering or queuing delays that can arise from mismatched component throughput. More details are available in Sec. 7.

Efficient memory management: Processing live video streams demands robust memory (de)allocation strategies. Without careful management, memory fragmentation [28] and garbage collection [4, 29] overheads can cause unpredictable delays and degrade performance.

1.2 Contributions

Design and deployment of PAVE for enhanced pedestrians' safety while preserving privacy: PAVE integrates real-time video processing, MQTT subscription-based communication, and UWB localization to detect danger zones where pedestrians might be at risk. Detected danger zones are sent to phones running PAVE's iOS app. Each phone determines locally—without sharing any personal data with the edge server—whether the user is entering a danger zone and triggers an alert if necessary (see Fig. 5). We deployed PAVE on an edge server within the NSF COSMOS testbed, and used it to process diverse urban camera real-time feeds in NYC, Florida, and Turkey. Our field tests using live cameras show that PAVE can alert at-risk pedestrians ~**0.9 s** before a vehicle reaches them. With low-latency cameras, this extends to ~**1.6 s**, within the **1-2 s** window pedestrians typically need to react (as reported by [31, 57]).

Efficient processing with ROI detection: To reduce computational and network overhead and improve scalability, PAVE initially processes video streams at low resolution, which is sufficient to detect vehicles (large objects). When PAVE detects potential danger zones where pedestrians may be at risk, it zooms into the smallest area encompassing all the danger zones (ROI) by cropping out the non-danger areas. Then it checks whether a pedestrian is (going) inside the danger zones. If so, a warning is sent to subscribed iPhones, where the device locally determines whether the user is among those at risk and, if confirmed, it triggers an alert. This selective ROI processing boosts average precision (mAP) by ~50% without the high GPU and bandwidth costs of continuous full high-resolution frames processing. More details are provided in Sec. 3.

Quantifying impact of fine-grained factors on performance: We investigated the impact of fine-grained factors on video analytics performance, such as memory configuration, element-level latency, AI model size, and environmental variations. These factors are often overlooked in the existing literature. We performed extensive profiling to quantify the impact of these factors on latency with microsecond precision. We identified bottlenecks and derived guidelines for efficient pipeline configuration. Our experiments show that strategically configuring memory and compute for each pipeline component can reduce their overhead by more than **10**×. For example, by choosing the right memory for each component in the pipeline, the I/O latency can drop from tens of milliseconds to a few hundred microseconds.

2 Related Work

Video analytics applications. There has been significant research into real-time video analytics applications in urban safety and smart cities applications. For example, [17, 18] process live camera feeds to generate traffic insights. The paper [16] combines object detection and tracking to detect collisions at urban intersections. GUTS [2] introduces a multi-object tracking algorithm that localizes and tracks road users in world coordinates using static street cameras. In StreetNav [32], we used a street camera to guide low vision users along outdoor routes. UTS [6] tracks vehicles using a 3D bounding box representation and a KF-based motion model. An automatic wrong-way vehicle detection system from on-road surveillance camera footage is proposed by [64]. SafeCross [78] improves intersection safety by detecting activity in the left-turn blind zone and issuing warnings. SafeCross is evaluated offline using recorded videos.

Video analytics optimization. Several recent papers have focused on optimizing video analytics, using methods like dynamic orchestration [69], load balancing [12], hierarchical and distributed processing [35, 62], sensor fusion [50], etc.

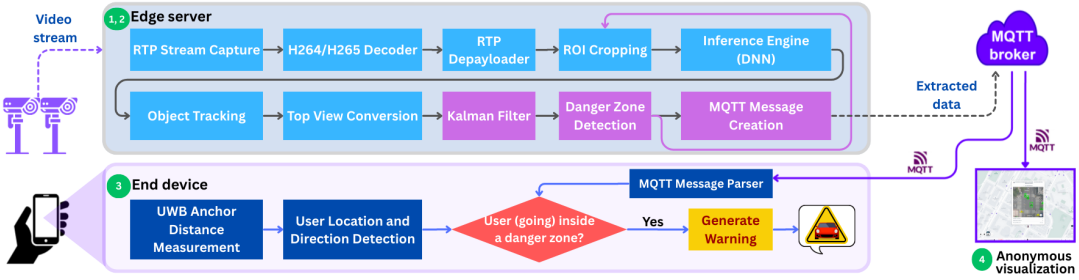

Figure 5: End-to-end components of PAVE deployed across an edge server and an end device, along with an anonymized visualization map.

Yoda [80] introduces a benchmarking framework to profile video analytics on diverse video content, assessing accuracy versus resource cost trade-offs. PECAM [79] uses a generative adversarial network to perform privacy filtering on video frames. Amadeus [10] achieves a balance between utility and privacy by leveraging object whitelisting. Gemini [49] dynamically allocates tasks to CPU versus GPU resources based on frame complexity. CASVA [85] uses deep reinforcement learning to optimize frame rate, resolution, and compression rate dynamically. MicroEdge [8] uses containerization and workload isolation to run diverse analytics simultaneously. EdgeCloudAI [20] achieves low-latency anomaly detection by filtering data at the edge and querying powerful cloud models sparingly. Jigsaw [23], Spatula [34], and Mosaic [24] use spatial and temporal cross-camera correlations to enhance throughput and scalability. DeepScale [55] uses dynamic frame-size selection, AdaMask [48] adapts video compression parameters. DCO [83] reduces offloading latency by compressing data prior to transmission. ComAI [36] uses collaboration among devices with overlapping fields of view to improve object detection accuracy. An optical flow-based method is used by [47] to identify ROIs and prioritize them for processing.

Compared to previous studies, this work is focused on operational feasibility and scalability of the end-to-end system with a specific emphasis on latency. Each component is specifically designed and configured to achieve reasonable accuracy while maintaining low latency, deployment cost, and complexity. Unlike prior research that primarily explored algorithmic improvements, this study demonstrates an operational framework that reduces computational and network overhead by performing low-resolution detection for large objects and selective Region of Interest (ROI) processing only when potential danger zones are detected. It also preserves pedestrian privacy by transmitting only danger-zone information to the end device, which locally determines whether the user is inside the zone. Additionally, unlike prior research, we implemented PAVE in low-level programming and measured component-wise latency with high precision, allowing us to report realistic latency achievable in practice.

3 Real-time Data Analytics at the Edge

In this Section, we describe the video processing pipeline and danger zone detection components, illustrated in Fig. 5 by light blue and purple elements in the edge server. These correspond to the expanded light blue and purple boxes labeled 1 and 2, respectively, in Fig. 1.

3.1 Analytics Workflow

The real-time RTSP video streams from traffic cameras are sent to an edge server for processing. As shown in Fig. 5, the workflow begins with RTP stream capture, where incoming video streams are ingested. The captured streams are then decoded using H264/H265 decoder and subsequently are passed through an RTP depayloader to extract raw frames. If necessary, ROI Cropping is applied on raw frames to focus on regions of interest in the frame (danger zones). These cropped regions or the whole frame are processed by the inference engine to detect and classify relevant objects, i.e., pedestrians and vehicles.

Detected objects are then passed to the Object Tracking module, which maintains object identities across frames. The trajectories of tracked objects are converted to a top view representation, facilitating the merging of data from multiple cameras viewing the same area from different angles and calculating speed/direction of pedestrians and vehicles. A KF-based method is used to predict the future trajectory of vehicles. Based on these predictions, danger zones are identified where pedestrians might be at risk (more details on KF and danger zone detection are provided in Sec. 4). Additionally, the extracted traffic and crowd data are visualized on a web-based map interface, showing anonymous information, such as bounding boxes, speed, and direction of vehicles and pedestrians for each camera feed. Camera icons on the map's web interface can be clicked to display background images overlaid with bounding boxes and numerical statistics. Once potential danger zones are detected,

their coordinates are sent to the ROI cropping element to zoom into the area encompassing all detected zones. This enhances the accuracy of pedestrian detection without the need to continuously process high-resolution video frames, which is computationally expensive and not scalable.

To demonstrate the effectiveness of this approach, we conducted experiments showing that processing only the ROI, rather than the full video frame, increases detection accuracy with pre-processing overhead of only around 0.1 ms (further latency details are provided in Sec. 6.2). Fig. 6 compares pedestrian detection performance (class "person") between ROI cropping and full-frame processing (no ROI), using the same model size (640×352). In this experiment, the ROI size is 1280×704, whereas the full frame is 3840×2160 (4K). The processing is done on an Nvidia A100 GPU in the COSMOS testbed. These measurements were obtained from recorded videos captured by the cameras shown in Fig. 4, totaling 10 minutes and covering several traffic light cycles. The videos were intentionally selected from days with **high pedestrian density** to evaluate PAVE's performance under worst-case conditions. Results show that, without increasing the model size, targeting the ROI significantly improves detection accuracy. This is because ROI cropping helps preserve small objects that might otherwise be lost during strided convolutions or pooling operations in the model's network. As illustrated in Fig. 7, ROI cropping not only improves accuracy but also reduces GPU cycles/memory usage and power consumption compared to high-resolution full-frame processing. By excluding irrelevant parts of the frame, the input image size is reduced, resulting in lower memory/computation usage and faster inference. In this paper, we used an Nvidia A100 GPU as our edge device because that was the hardware available in the COSMOS testbed. However, the GPU usage per camera is only less than 10% (as reported in Fig. 7), which is comparable to a much smaller GPU such as Nvidia AGX Orin. Given A100 = 19.5 Tera Floating Point Operations per Second (TFLOPS) and AGX Orin = 5.3 TFLOPS, the Orin provides ∼0.27× the capacity of A100, implying per-camera usage would be below 40% (∼10% / 0.27) [71]. This means that in practice, the system runs effectively on a small edge GPU while maintaining the reported performance.

In essence, the pipeline first processes all frames at low resolution (e.g., 480p), sufficient for detecting large objects like vehicles. When danger zones are identified, the pipeline crops out non-danger areas in **a single frame** (with the same resolution) to accurately detect pedestrians without the overhead of high-resolution processing. If a pedestrian is detected in a danger zone, the danger zone coordinates are sent to subscribed iPhones, enabling timely notifications to pedestrians at risk. To further optimize video analytics, we have developed a system that automatically selects the

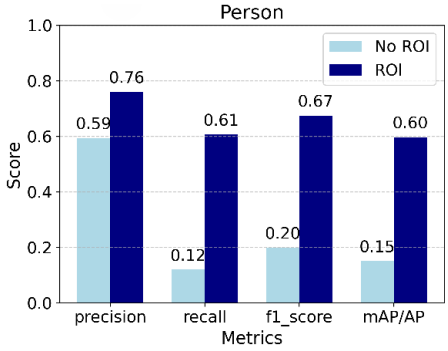

Figure 6: Object detection performance (class "person"): ROI cropping versus full-frame processing (no ROI) using the same model size.

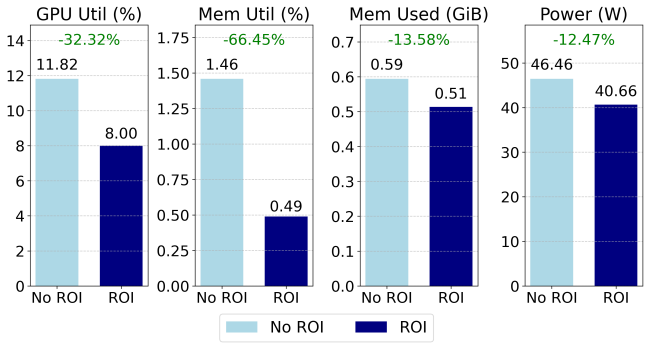

Figure 7: GPU resource usage: ROI versus full-frame processing (no ROI) using the same model size.

optimal resolution and frame rate based on real-time conditions, which is currently patent pending [21]. However, that system is not used in this paper, and its integration with PAVE is subject to future work.

3.2 Top View Conversion

PAVE converts detected bounding box coordinates of vehicles and pedestrians from angled camera views to a top view, facilitating integration of multiple cameras viewing the same intersection. This transformation also enables PAVE to estimate the speed and direction of vehicles and pedestrians using known on-ground distances in the unified top view.

In our previous studies [19, 22], we experimented with traditional geometric methods such as homography and planar perspective transformations based on standard photogrammetry equations to estimate on-ground distances from pixel measurements. However, for cameras with wide and angled views of urban intersections (such as the one in Fig. 4), these purely geometric methods produced unsatisfactory and highly approximate results due to distortions and non-planar surfaces. Motivated by these limitations, we adopt a deep learning based approach in this work to achieve more

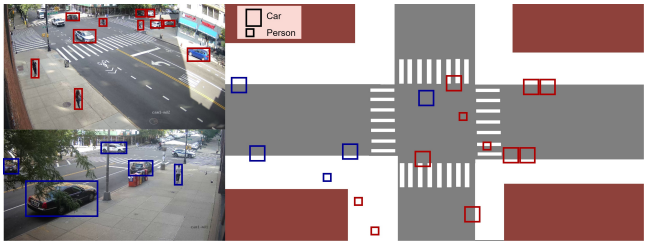

Figure 8: Bounding box conversion from angled perspectives to top view perspective using *TransformNet*.

accurate and flexible top-view mappings. Specifically, we present two complementary methods: (i) one that leverages a lightweight neural network trained on manually annotated correspondences, and (ii) another that achieves perspective transformation automatically without requiring training data. The former is applicable when road measurements are available and provides higher accuracy, whereas the latter is suitable when such measurements are unavailable, achieving comparable accuracy in a fully automated manner.

Conversion with training. We trained a lightweight neural network to transform pixel coordinates from two street-level cameras in NYC, shown in Fig. 4, to a top view perspective of the intersection (referred to as *TransformNet*). The primary goal of *TransformNet* is to provide a mapping from the input coordinates in the cameras' perspective to the corresponding points in the top view. We manually annotated ~**350** corresponding points from each camera perspective and the corresponding top view. Transformation of coordinates from one perspective to another is fundamentally a regression problem [65], where the network learns to map input coordinates to output coordinates. Hence, *TransformNet* consists of a simple architecture with three intermediate layers using ReLU activation (64, 128, and 64 neurons, respectively).

Fig. 8 shows an example of using *TransformNet* to convert coordinates of detected pedestrians and vehicles from both cameras' perspectives into the top view. This approach simplifies the visualization and facilitates the integration of multiple cameras viewing the same scene, such as an intersection. Additionally, unlike the original tilted views, where a fixed ground distance maps to varying pixel lengths across the frame, the top view preserves an approximately constant scale. This makes it easier to compute the speed and direction of pedestrians and vehicles. Table 1 shows the impact of cameras' integration on accuracy. The results indicate that integrating the detections from both cameras enhances accuracy by over **10%** for both pedestrians and vehicles. For pedestrians, the bottom center pixel is a sufficient center point approximation. However, due to the larger size of vehicles, identifying their physical center point as they move is more challenging, leading to some degradation in accuracy.

Table 1: Comparison of detection performance using individual cameras and multi-camera integration.

Class	Metric	1st Cam	2nd Cam	Multi-Cam
Pedestrian	Recall	0.628	0.756	0.812
	Precision	0.787	0.708	0.827
	F1 Score	0.699	0.731	**0.819**
Vehicle	Recall	0.690	0.560	0.847
	Precision	0.519	0.740	0.623
	F1 Score	0.592	0.637	**0.718**

Conversion with no training. *TransformNet* training requires manual annotation of corresponding points. However, manual annotation for a large number of cameras is very time consuming. Therefore, we introduce a more scalable method for perspective transformation that does not require manual annotation with comparable accuracy. To automate perspective transformation, we combine panoptic segmentation with monocular depth estimation [14, 27] and identify the perspective transformation parameters from the 3D point cloud of the ground points.

This approach, illustrated in Fig. 9, consists of three main components. First, a monocular depth estimation model is employed to predict dense depth maps from a single RGB image [5, 81, 84], while a semantic segmentation model identifies drivable surfaces such as roads and sidewalks [33]. Optionally, focal length is estimated to improve geometric consistency [40]. The estimated depth and segmentation outputs are combined to reconstruct the 3D scene geometry by back-projecting depth values using a pinhole camera model. 3D points corresponding to segmented ground regions are extracted to estimate the ground plane, and a planar model is fit via linear regression. This ground plane defines a local coordinate system for the scene. Detected objects are then localized in 3D by projecting the bottom center of each object's bounding box onto the ground plane. Finally, the full RGB image is re-projected onto a perspective-correct top-down heightmap by mapping each pixel's 3D coordinate onto the local ground plane, preserving semantic layout and visual appearance. This process yields dense and geometrically accurate top view visualizations without the need for multiple views or LiDAR's depth information.

4 Kalman Filter and Danger Zone Detection

To identify danger zones where pedestrians might be at risk, it is necessary to predict the trajectories of vehicles first. Given the critical importance of latency in this application, a lightweight yet reliable trajectory prediction method is required. We adopted the Kalman Filter (KF) [41] for its low computational overhead and latency, and its ability to operate without requiring offline training. Common alternatives include LSTM-based encoder-decoder models that capture

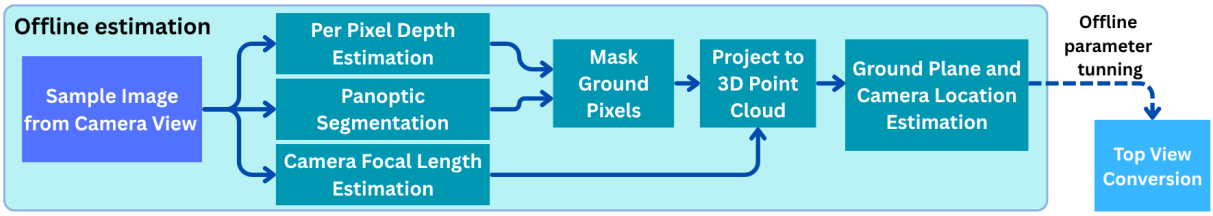

Figure 9: Monocular pipeline for top view conversion using depth estimation, semantic segmentation, and ground plane fitting that only requires a single RGB image.

long-term temporal dependencies but require heavy computation and training [39], and graph/attention-based neural architectures that model moving-objects interactions with high accuracy at the cost of increased latency and resource demands [15]. In addition, we explored several lightweight machine learning classification methods (Gradient Boosting, Support Vector Machines, and K-Nearest Neighbors) to forecast vehicle movements. These models achieved over **90%** accuracy with latencies under **2 ms**. However, their performance was constrained by the granularity of the manually labeled training data, which had to be tailored for each camera view. In contrast, KF requires no pre-training and operates directly on real-time observations using only current and previous states, making it an efficient and highly scalable choice for our pedestrian safety application.

KF operates in two phases: the **time update** (prediction) and the **measurement update** (correction), governed by the following equations,

Prediction step:

$$\hat{x}_k^- = A\hat{x}_{k-1} + Bu_{k-1} \tag{1}$$

$$P_k^- = AP_{k-1}A^\top + Q \tag{2}$$

Update step:

$$K_k = P_k^- H^\top (HP_k^- H^\top + R)^{-1} \tag{3}$$

$$\hat{x}_k = \hat{x}_k^- + K_k(z_k - H\hat{x}_k^-) \tag{4}$$

$$P_k = (I - K_k H)\, P_k^- \tag{5}$$

For the k^{th} frame, the state vector $x_k \in \mathbb{R}^{4\times1}$ represents the 2D position and velocity of a vehicle in the top view, i.e., $x_k = [x, y, \dot{x}, \dot{y}]^T$. The predicted state $\hat{x}_k^-$ is estimated using the transition matrix $A \in \mathbb{R}^{4\times4}$, which models the motion dynamics assuming constant velocity. Since external inputs such as acceleration are not available, we set the control-input matrix $B = 0$ and omit the control vector u_{k-1}. The matrix $P_k \in \mathbb{R}^{4\times4}$ denotes the state estimation error covariance, and $Q \in \mathbb{R}^{4\times4}$ is the process noise covariance that captures uncertainty in the motion model. The measurement $z_k \in \mathbb{R}^{2\times1}$ is obtained from the video processing pipeline and provides the observed 2D position of the vehicle. The observation matrix $H \in \mathbb{R}^{2\times4}$ maps the state space to the measurement space, and $R \in \mathbb{R}^{2\times2}$ is the measurement noise covariance. The Kalman gain $K_k \in \mathbb{R}^{4\times2}$ determines the weighting between prediction and measurement during the update step. Sec. 6.3 describes how to select the KF's parameters.

KF for trajectory prediction. Each time PAVE receives new observation data, it uses it to update its state and make predictions. Then the next position is computed using the position and velocity stored in the state. This predicted position is then used as the measurement $z_k = HA\hat{x}_{k-1}$ during the update step. By iteratively applying this process, a predicted trajectory of arbitrary length can be generated. Each vehicle's predicted trajectory is used to infer its travel direction (i.e., straight, turn left/right). As shown in Fig. 10(a), when a vehicle intersects a red trigger line, the KF's velocity estimate is used to determine whether the vehicle is traveling straight or making a left/right turn. According to recent studies [31, 57] a minimum reaction time of approximately **1 to 2 seconds** is required for pedestrians to respond to a warning and begin moving to avoid a collision. Fig. 10 illustrates the trade-off between prediction accuracy and pedestrians' time to react. The average time between a vehicle crossing the red trigger line and subsequently crossing the black finish line is recorded as the time to react. As shown in Fig. 10(b), KF achieves **82%** accuracy for turning vehicles and **99%** for going straight vehicles at **1 s** reaction time. However, accuracy drops to **39%** for turning vehicles at **2 s** reaction time, demonstrating the trade-off. To be conservative, PAVE predicts trajectories for both 1 s and 2 s when each new frame arrives and uses both endpoints to identify danger zones.

Danger zone detection. The next step is to use the predicted trajectories to identify danger zones (see Fig. 5). Danger zones are pre-determined offline and correspond to areas where pedestrians are likely to be present, such as regions near crosswalks. An illustrative example is shown in Fig. 2, where the entire area is divided into 12 zones that represent potential pedestrian risk. If a predicted endpoint of a vehicle's trajectory intersects any of these zones, it is labeled as a danger zone.

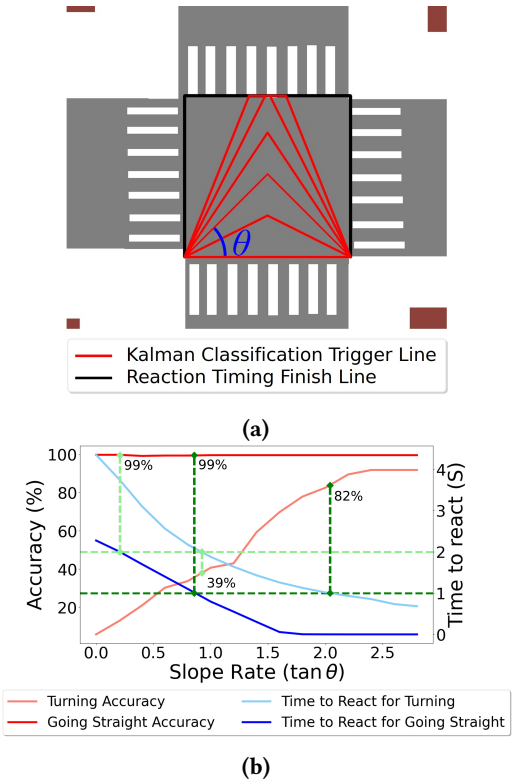

Figure 10: Prediction of vehicle travel direction and pedestrian reaction time. When a vehicle intersects the red trigger line in (a), its direction is inferred from its estimated velocity. The time between crossing the red line and the black finish line is recorded as reaction time. (b) shows how varying the slope angle θ of the red line affects prediction accuracy and reaction time.

5 Pedestrian Localization and Warning Generation

Once pedestrians are detected in danger zones, PAVE must identify the pedestrians at risk and notify them. This task runs on users' iPhones as shown in Fig. 5 (the box labeled 3).

Rationale for UWB selection. Recent studies explore a variety of localization technologies to improve urban safety. For example, Wi-Fi Fine Timing Measurement (FTM), when fused with Global Navigation Satellite Systems (GNSS) and Inertial Measurement Units (IMUs), can improve 3D localization. However, it requires considerable computational overhead [46]. Bluetooth Low Energy (BLE) and Radio Signal Strength Indicator (RSSI) are effective for assistive applications but suffer from signal fluctuation and limited granularity [68]. Radio Frequency Identification (RFID) solutions provide proximity detection in industrial settings, but passive variants have a limited range, and active variants are vulnerable to packet loss and RF interference [61].

Multi-view video streams can also be used, but typically offer only meter-level accuracy and are sensitive to lighting conditions and occlusion [25]. Moreover, camera based methods cannot identify which pedestrian device to send alerts to without requiring pedestrians to identify themselves to the system which compromises their privacy. LiDAR provides precise line-of-sight ranging but is costly and degrades in adverse weather conditions and occlusion [25, 51]. Satellite positioning (GPS/GNSS) is another option, but it performs poorly in dense urban areas [77] and suffers from low update rates [68]. Wi-Fi RTT is primarily used for indoor localization, and its outdoor accuracy is meter-level (even exceeding 2 m in realistic deployments) [1, 42]. Furthermore, iOS devices do not expose a Wi-Fi RTT API, as the feature is currently supported only on Android platforms.

PAVE employs UWB technology to locate pedestrians with **centimeter-level accuracy** and **millisecond-level latency**, both of which are critical for timely collision warnings [11, 60]. Compared to other methods, UWB provides the fine-grained precision necessary to determine whether a pedestrian is truly "inside" a small crosswalk danger zone. Moreover, iOS devices natively support UWB through the Nearby Interaction API, enabling seamless integration with users' devices. Therefore, PAVE's usage of UWB allows us to implement and evaluate an end-to-end system.

Deployment practicality. In urban environments, existing infrastructure such as traffic lights and lamp posts already provide power and protective enclosures for installing sensors (e.g., cameras, LiDARs). Since UWB modules are compact, inexpensive, and operate on a simple 2.5–3.6 V input, they can be easily integrated into such infrastructure without major modifications.

Localization systems usually have fixed **anchors** and **moving tags**. In PAVE, anchors refer to UWB devices with known and fixed positions, while tags refer to mobile devices with unknown positions that need to be determined. The anchors are built using the Qorvo DWM3001CDK, a development kit based on the DWM3001C UWB transceiver module, which integrates Bluetooth Low Energy (BLE) and supports Two-Way Ranging (TWR). The accompanying firmware handles UWB configuration and communication using BLE as the control channel, which is necessary for UWB Out-of-Band (OOB) communication link [63]. The tags are UWB-equipped iPhones, with an iOS app that uses Apple's Nearby Interaction framework. The app facilitates BLE connections, enables TWR between iPhones and UWB anchors, and computes the device's location in real-time using trilateration, which estimates a tag's location based on its distances to at least three known anchors. Additionally, linear regression is used to determine the direction of pedestrian movement according to the five most recent locations.

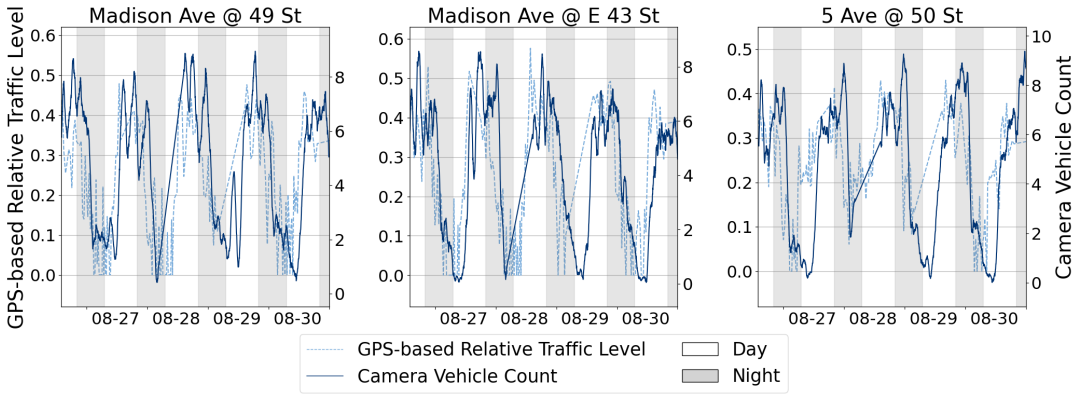

Figure 11: GPS-based traffic levels vs. camera-based vehicle counts, from August 26 to August 31, 2024.

Every time a pedestrian near or in a danger zone is detected, pedestrians with the iOS app installed on their phone receive an MQTT message containing the coordinates of the danger zone. The app then uses its location and direction of movement to determine if the pedestrian is moving inside the danger zone, in which case a warning is emitted. If the pedestrian is leaving the danger zone, no alarm goes off to avoid shocking the pedestrian.

6 Measurements and Evaluations

6.1 Video Analytics Performance

Comparing video with GPS traffic data. To assess the potential reliability and practicality of PAVE as an alternative to traditional methods, we compare its results with the most widely used approach for traffic and navigation management, GPS-based data. Most navigation platforms rely on gathered data from GPS receivers installed in vehicles, smartphones, and other devices. To carry the comparison, we collected traffic information for several locations in NYC using two sources: (1) TomTom [74], an API that provides real-time traffic and navigation data based on GPS data from millions of devices, as well as other sources, (2) Extracted traffic data from the NYC Department of Transportation (DoT)'s public cameras [56], which were analyzed using our developed pipeline. The results are shown in Fig. 11.

The NYC DoT cameras are updated every few seconds and can be publicly accessed [56]. By running our end-to-end video analytics pipeline, we extracted the number of vehicles captured in each frame, generating continuous traffic information data over time. This data, named camera vehicle count and represented by the dark blue curves in Fig. 11, was collected from three inspected cameras. In addition, we collected matching information using the TomTom Traffic Flow API, which provides real-time GPS-based traffic levels information [76]. The traffic level for each road segment is represented by a value between 0 and 1, indicating the *relative* traffic level. This value is calculated as a fraction of the current speed in the road, relatively to free-flow conditions, highlighting areas of congestion. Traffic conditions are classified based on how much slower vehicles are moving compared to free-flow conditions [73]. The GPS-based relative traffic levels are depicted in Fig. 11 with light blue dash curves, where for each inspected NYC camera location we collected traffic level from the nearby road segment. As can be seen in Fig. 11, there is a significant correlation between the two metrics, supporting the effectiveness of our video processing pipeline.

To assess this relationship quantitatively, we employ *Spearman's rank correlation coefficient* ρ, which evaluates whether peaks and troughs in the two series align regardless of their absolute scales [7, 26, 66]. Since our goal is to validate that the pipeline provides consistent measurements of traffic jams and loads, we focus on testing for a strong monotonic correlation. Comparing the average traffic levels and number of vehicles detected per frame, we find a strong correlation with Spearman's ρ around **0.6**. This correlation is also evident visually in Fig. 11, where the two curves closely follow each other's trends over time.

GPS-based traffic density typically updates every few minutes, with frequency depending on the platform and available data. TomTom updates roughly every minute in high-traffic areas but less frequently in rural regions [75]. Google Maps intervals vary based on traffic volume and crowd-sourced data availability [52]. In contrast, real-time camera analysis provides more accurate and granular data. Cameras can update at high rates (e.g., 30 fps), making them potentially more effective for real-time applications.

Top view conversion performance. To evaluate perspective transformation method illustrated in Fig. 9, we used 12 camera views from NYC, Florida, Turkey, and the VisDrone dataset [86]. Since ground truth top view data is unavailable for these cameras, we assess how well right angles are

Table 2: Deviation from 90° angles across different cities using perspective transformation in Fig. 9.

Dataset	Perspective Transformation Error (degrees)
NYC	4.08 ± 3.20
VisDrone	6.99 ± 5.75
Florida	10.92 ± 7.02
Turkey	12.76 ± 7.04
Combined	8.60 ± 6.77

preserved by annotating scene elements with known perpendicular structures, such as vehicles, street corners, and crosswalks. The results presented in Table 2 show that the reconstruction error is particularly low for grid-structured intersections, such as those in NYC and the VisDrone dataset. However, intersections in Florida and Turkey, which feature curved streets, present challenges for monocular depth estimation models in accurately capturing the 3D structure of the ground plane, leading to slightly higher errors.

6.2 Component-Wise Latency and Time to React

Components' latency. To better understand the complexities of real-time video analytics, we measured the incurred latency by main elements of the video analytics pipeline while processing live RTSP video streams from NYC cameras. The cameras use H264 encoding with I-frame interval equal to 10. The COSMOS edge server, where the video streams are processed, is equipped with an A100 GPU and Intel(R) Xeon(R) Gold 6326 CPU @ 2.90GHz. The video streams are 4K resolution with a frame rate of 30 fps. YOLOv8 [72] is used as the object detection model (with batch size equal to 1) with the Nvidia DCF-based tracker. The YOLOv8 model was configured for input images with 832×832 resolution. We performed these latency measurements for three sizes of YOLOv8 models: small, medium, and large (YOLOv8s, YOLOv8m, and YOLOv8x, respectively). The results are presented in Table 3, which compares the average and standard deviation of the latency associated with the main pipeline's element per frame.

Since the system is implemented in low-level languages (C and CUDA), we can directly access buffers and memory resources, which allows timestamping with nanosecond resolution. On Linux, we use `clock_gettime(CLOCK_MONOTONIC_RAW)` for CPU events (nanosecond resolution, tied to the hardware clock, typically in the GHz range) and CUDA events for GPU operations (also providing nanosecond resolution). In practice, this yields **microsecond-level accuracy**. To further enhance precision, we leverage GStreamer's signaling mechanism: timestamps are captured when data hit the input and output pads of each element. This ensures that latency is measured at clearly defined points in the pipeline. By combining high-resolution hardware clocks with OS-level signaling, we minimize jitter and capture reliable per-component latency.

The latency was measured under two conditions: ***sparse*** (low traffic and pedestrian volume) and ***busy*** (high traffic and pedestrian volume). In the following, we explain the function of each element in a video analytics pipeline and their incurred latency according to the results in Table 3.

RTP session handler: Manages RTP sessions, ensuring that data packets are properly handled within the session. It exhibits consistently low latency with minor variations, which is relatively stable under varying traffic volumes.

RTP depayloader: Strips off RTP packet headers to extract the payload containing the encoded video data. Its latency, though slightly higher than the RTP session handler, is still relatively low. The increased standard deviation in latency during busy times is due to the variability in the number of packets per frame, driven by the increased movement and density in the scene.

Stream parser: Assembles the encoded byte stream into a format that the decoder can understand. The stream parser shows noticeable variations in latency, especially under busy conditions, caused by variability in frame sizes that leads to fluctuation in parsing time.

Accelerated decoder: Decodes encoded video frames into raw frames. The decoder demands high latency relative to other elements, particularly in busy conditions. In busy conditions with more movement and objects, frame sizes are larger as more data is required to encode the details. Therefore, the decoder has to process larger amounts of data per frame, leading to a higher average latency. The variability in the data rate and frame size causes fluctuations in decoding times, leading to a higher standard deviation in latency.

ROI cropping: Crops the ROI from the raw video frames and discards the rest. The latency for ROI cropping remains consistently low across all conditions and models, with minimal standard deviation. This suggests that ROI cropping is an efficient method for enhancing video analytics performance with minimal overhead.

Inference engine (YOLOv8): Runs object detection model (YOLOv8) on the cropped ROIs. As expected, the inference engine introduces the highest latency in the pipeline. The latency increases with the model size, with YOLOv8x (large) exhibiting the highest latency, but it is relatively stable across busy and sparse conditions. YOLOv8x exhibits higher variation in latency due to its larger architecture.

Object tracking: Assigns unique IDs to the same objects across the video frames detected by the inference engine. The object tracking element shows higher latency in busy conditions due to the increased number of detected objects that need to be tracked.

Table 3: Average latency (Avg.) and Standard Deviation (Std Dev.) for different pipeline elements across YOLOv8 variants during busy and sparse times.

Pipeline Element	YOLOv8s				YOLOv8m				YOLOv8x			
	Busy		Sparse		Busy		Sparse		Busy		Sparse	
	Avg. (ms)	Std Dev. (ms)	Avg. (ms)	Std Dev. (ms)	Avg. (ms)	Std Dev. (ms)	Avg (ms)	Std Dev. (ms)	Avg. (ms)	Std Dev. (ms)	Avg. (ms)	Std Dev. (ms)
RTP session handler	0.063	0.019	0.066	0.017	0.065	0.017	0.064	0.0162	0.063	0.012	0.065	0.015
RTP depayloader	0.172	0.246	0.188	0.187	0.185	0.261	0.188	0.169	0.183	0.259	0.176	0.171
Stream parser	0.342	0.605	0.341	0.459	0.367	0.645	0.3459	0.439	0.348	0.610	0.339	0.459
Accelerated decoder	1.472	2.173	1.088	1.176	1.437	2.071	1.233	0.884	1.092	1.577	0.962	0.964
ROI cropping	0.113	0.019	0.095	0.010	0.107	0.029	0.110	0.017	0.095	0.010	0.110	0.020
Inference engine (YOLOv8)	4.120	0.118	3.948	0.119	7.241	0.113	7.191	0.129	11.020	2.613	11.26	2.469
Object tracker	1.191	0.142	0.873	0.078	1.097	0.161	0.850	0.185	1.283	0.161	0.928	0.122
MQTT message creator	0.107	0.027	0.039	0.021	0.109	0.024	0.054	0.018	0.128	0.051	0.065	0.017

MQTT message creator: Convert the obtained metadata into MQTT format [53]. Its latency is consistently low across all models and conditions, though it increases slightly in busy conditions due to the larger message sizes that need to be assembled.

Field test to measure time to react. To measure how fast PAVE can alert pedestrians at risk before vehicles reach the danger zone, we conducted a field test. The test was deliberately performed on a rainy day at a crowded and challenging intersection in NYC, surrounded by tall buildings, trees, and other obstacles. These conditions were chosen to realistically assess the system performance in complex urban environments. We went to this intersection shown in Fig. 4 and programmed an iPhone to locate itself in a danger zone while holding it and standing on the sidewalk. This avoided the need for an actual person to be at risk during testing. We then started PAVE to process live video stream from the 2nd floor camera with KF prediction time set to 2 s. We then recorded the time gap between when the user received the alert and when the vehicle reached the identified danger zone (i.e., time to react). We repeated this test **24** times, and the results show that reaction time ranges from **0.2 s** to **0.94 s**, with an average of ∼**0.6 s**. Since our cameras are not low-latency and incur at least ∼**0.7 s** latency based on our measurements, using low-latency cameras would enable PAVE to achieve the **1 to 2 seconds** of reaction time required for pedestrians to move to safety, as discussed in Sec. 4.

6.3 Kalman Filter and UWB Performance Measurement

To measure the accuracy of the trajectory prediction, we used a **94-minute** recorded video from the 2nd floor camera shown in Fig. 4 and manually annotated the vehicle trajectories. Then we used *TransformNet* to transform the data from the camera view to top view. We used these as measurement data and applied a moving average to smooth the trajectories and eliminate noise, as vehicle motion is inherently smooth. Using these data, we estimated the KF parameters. We computed the process noise covariance matrix Q from the residuals between motion model predictions and ground truth, and the measurement noise covariance matrix R from the residuals between raw camera measurements and ground truth. Q captures the uncertainty in vehicle motion dynamics and R represents the noise in camera-based position measurements. The calculated matrices are given below:

$$Q = \begin{bmatrix} 0.99 & -0.57 \\ -0.57 & 1.20 \end{bmatrix}, \quad R = \begin{bmatrix} 6.91 & -4.16 \\ -4.16 & 8.71 \end{bmatrix} \tag{6}$$

Using the above parameters, we measured KF performance via two common metrics: average displacement error (ADE) and final displacement error (FDE). ADE compares all predicted points with the ground truth and calculates the average, while FDE only compares the last point. The prediction accuracy results are shown in Fig. 12. As expected, accuracy degrades as the length of prediction window increases. By dividing the area into danger zones, we can eliminate the impact of small prediction errors to a great extent. As discussed in Sec. 4, we chose a prediction length of **1 s and 2 s** with **0.8-2 m** FDE, sufficient to trigger danger zones spanning several meters. To be precise, we measured accuracy of danger zone detection using the annotated **94-minute** video. When prediction length is **1 s** (**2 s**), the results show that **99%** (**98%**) of the predicted zones were correctly activated, with only **0.5%** (**1%**) false positives and **0.6%** (**1.3%**) false negatives.

To evaluate the potential of using UWB anchors for pedestrian localization, we set up a 10 m×10 m test field and placed UWB anchors at each of the three corners of the rectangular area, as shown in Fig. 13. Our results showed that a

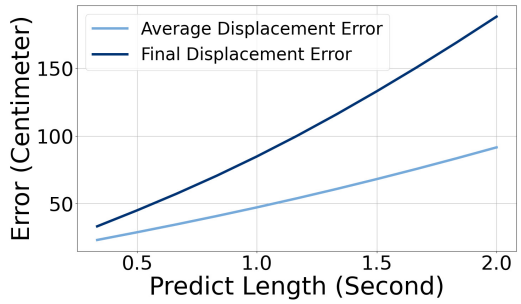

Figure 12: KF performance based on prediction length.

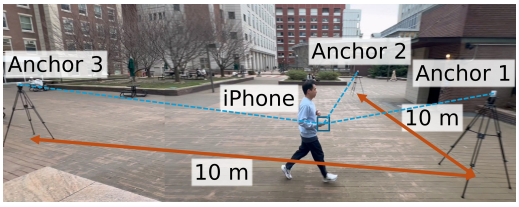

Figure 13: UWB pedestrian localization settings at 10 m×10 m test field.

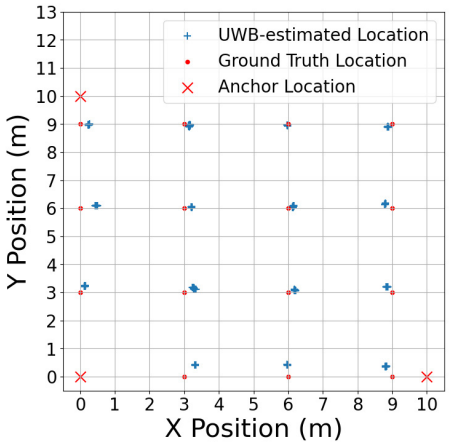

Figure 14: Comparison of UWB-estimated versus ground truth locations in a 10 m×10 m test field.

pedestrian holding a UWB-enabled smartphone could be localized with an accuracy of **less than 30 cm** and an update frequency of **5 HZ**.

To measure pedestrians' UWB-based localization performance, we conducted static and dynamic localization tests. Three anchors were installed at a height of 1.8 m above ground with 2D coordinates of (0 m, 0 m), (10 m, 0 m), and (0 m, 10 m), while a pedestrian was holding a smartphone with the iOS app installed. We compared the localization measurements at fifteen predefined positions with the ground truth location, as shown in Fig. 14. Results reveal an average error of only **26.9 cm** with a standard deviation of **12.7 cm**.

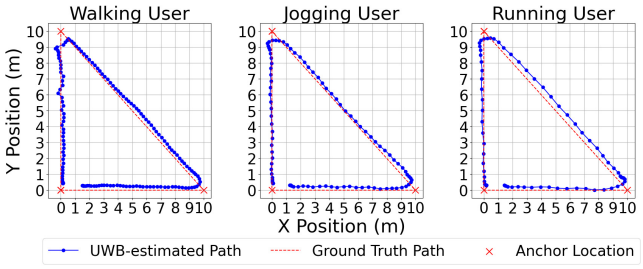

Figure 15: Dynamic localization of a user while walking, jogging, and running.

For the dynamic test, localization measurements were collected for a pedestrian walking, jogging, and running along a fixed trajectory connecting the three anchors, as shown in Fig. 15. The figure shows that the estimated path closely aligns with the true path. Table 4 shows the mean error and variance with respect to the real path.

7 Lessons Learned

In this section, we discuss some of the important lessons we learned while developing and deploying PAVE.

7.1 Video Analytics Components Integration

Data transfer overhead. Video analytics pipelines are composed of several components as presented in Fig. 5–labeled (1,2). To process the video streams, data are continuously transferred between components. These transfers can introduce significant latency (e.g., 10-100 milliseconds per frame), particularly when copying data between the CPU and GPU or across network interfaces in distributed systems. To minimize this overhead, subsequent components should be designed to share memory space whenever possible.

Buffering and queuing delays. Components often rely on buffers to regulate data flow. The buffer sizes depend on the data arrival rate and throughput of each component. If these buffers become full due to a mismatch in the processing speeds of different components, it can lead to additional delays (tens of milliseconds) or pipeline failure. To address this, dynamic memory allocation can be used to adjust buffer sizes at runtime based on observed data rates. However, this

Table 4: Localization error average and standard deviation for walking, jogging, and running users.

Error	Activity Type		
	Walking	Jogging	Running
Avg (cm)	19.4	19.3	20.5
Std Dev (cm)	10.8	11.6	12.2

approach must be implemented carefully, as frequent allocation and deallocation can introduce overhead and degrade system performance.

Memory allocation and deallocation. Frequent memory allocation and deallocation can incur tens of milliseconds of latency, depending on data size. Two primary factors contributing to this latency in video analytics are:

(i) Fragmentation: Constant memory allocation and deallocation can cause fragmentation (small unused gaps between allocated blocks) that reduces effective memory availability and increases latency [28]. This can be mitigated using memory pooling or pre-allocated memory regions to reduce allocation frequency and fragmentation.

(ii) Garbage collection overhead: In environments with automatic memory management (e.g., high-level languages like Python), frequent memory operations trigger garbage collection more often, introducing unpredictable delays and degrading performance [4, 29]. For time-sensitive applications, explicit memory management using low-level languages like C/C++ is necessary to avoid unpredictable garbage collection overhead.

7.2 Video Processing Pipeline Optimization

Real-time video data processing is resource-intensive. To ensure scalability, we employ several optimization techniques to reduce overhead. Our pipeline is implemented using the GStreamer multimedia framework [70] in C, allowing for customized, low-level optimization and efficient memory management.

TensorRT engine. TensorRT engine [59] optimizes AI models by performing operations like layer fusion and kernel auto-tuning. These optimizations reduce the computational overhead of the models, thereby lowering the latency. Additionally, TensorRT's dynamic tensor memory management helps minimize memory usage during inference, ensuring that the models run efficiently on available resources.

Accelerated decoding. By offloading video decoding to dedicated GPU hardware (e.g., Nvidia's NVDEC), CPU load is reduced, and decoding is accelerated. This lowers per-frame decode time and data transfer overhead. Details on hardware- vs. software-based decoding are provided in Sec. 7.3.

Memory management. To reduce latency introduced by frequent data transfers between the CPU and GPU, we optimized the pipeline by using CUDA memory [67] for components running on GPU, and V4L2 (Video for Linux 2) memory [45] for the hardware-accelerated decoder. CUDA memory allocates video frames directly in GPU memory, allowing the GPU to access data without transferring it over the Peripheral Component Interconnect express (PCIe) bus. This minimizes the time required to move video frames between different pipeline components. For video decoding, V4L2 memory is used, which is specifically designed for hardware-accelerated video processing on the GPU. This eliminates the overhead of memory fragmentation and ensures direct memory access (DMA) for GPU operations. More details on the impacts of memory type are presented in Sec. 7.4.

Table 5: Comparison of per-frame latency incurred by hardware and software decoding for real-time H264 encoded video streams.

		HW Decode		SW Decode	
		Sparse	Busy	Sparse	Busy
Decoding latency (ms)	Avg	1.233	1.437	20.801	19.386
	Std Dev	0.884	2.071	17.704	21.596
I/O latency (ms)	Avg	0.001	0.002	8.265	8.293
	Std Dev	0.001	0.001	0.112	0.142

7.3 Impact of Video Decoding on Latency

We assessed the impact of using hardware-accelerated (denoted by HW) decoding compared to software-based (denoted by SW) decoding. We measured the incurred latency per frame for both methods, including the data I/O latency and decoding latency. As shown in Fig. 5, after each frame is decoded, it should be prepared for inference that runs on GPU. To do this, each decoded frame must be transferred to GPU memory and preprocessed to match the inference engine's input format–this includes color space conversion, scaling, and format adjustment. We define ***decoding latency*** as the time required to decode each frame, while the ***I/O latency*** is the time to transfer decoded frames to the GPU memory accessible by the inference engine and be preprocessed for compatibility.

We measured the average and standard deviation of these latencies while the pipeline processes live RTSP streams from the NYC cameras shown in Fig. 4. It runs on the edge server with A100 GPU and Intel(R) Xeon(R) Gold 6326 CPU @ 2.90GHz. Similar to Sec. 6.2, we performed these measurements for ***sparse*** (low traffic and pedestrian volume) and ***busy*** (high traffic and pedestrian volume). The results are presented in Table 5. The data show that SW decoding imposes a latency that is **more than 10**× higher than HW decoding. Additionally, since SW decoding is performed on the CPU, the decoded frames must be transferred to GPU memory before inference, whereas HW decoding allows frames to be directly accessible by the GPU. This memory transfer introduces an additional latency of **over 8 ms**, which is significant. This highlights the importance of minimizing data transfers between CPU and GPU memory.

Table 6: Comparison of the impact of various memory types on latency while processing a real-time stream.

Element	Pinned Memory		Unified Memory		Device Memory	
	Avg (ms)	Std Dev (ms)	Avg (ms)	Std Dev (ms)	Avg (ms)	Std Dev (ms)
Hardware accelerated decoder	1.383	2.090	0.674	2.007	0.685	1.920
I/O latency	0.003	0.002	0.003	0.001	0.001	0.001
ROI cropping	1.093	0.041	0.116	0.024	0.110	0.007
YOLOv8m	7.356	0.199	7.336	0.210	7.338	0.189
Object tracker	1.838	0.240	1.467	0.289	1.476	0.245
MQTT message creator	0.176	0.037	0.161	0.058	0.163	0.038

7.4 Impact of Memory Type on Latency

To evaluate the impact of different memory types on pipeline latency, we measured latency using three common memory types for key elements in the pipeline, including decoder, ROI cropping, inference engine, and tracker. In all experiments, these elements run on the A100 GPU. The setup is similar to Sec. 6.2, where an H264 encoded live RTSP stream at 4K resolution and 30 fps is processed using YOLOv8m and NVIDIA DCF-based tracker. The three memory types evaluated in this study are described below [13, 30, 44]:

Pinned memory: A region of host (CPU) memory that is *pinned*, meaning it cannot be paged out by the operating system. This enables the GPU to access the data directly via DMA (zero-copy), eliminating the need for an explicit host-to-device copy. However, since the data still traverses the PCIe bus, some latency remains.

Unified memory: Creates a shared memory space that both the CPU and GPU can access. It dynamically migrates data between CPU and GPU as needed, eliminating the need for explicit memory transfers. However, on-demand paging introduces some latency when data is not already in the required memory space.

Device memory: In this configuration, all components directly use memory that resides entirely on the GPU, eliminating PCIe bus transfers. The decoder uses V4L2 memory, while other components use CUDA memory. Both V4L2 and CUDA memory reside on GPU memory. V4L2 Memory is optimized for decoding operations. CUDA memory is optimized for general-purpose GPU computing tasks, such as image processing.

The results, presented in Table 6, indicate that pinned memory generally shows higher latency, especially for the decoder, ROI cropping, and tracker. Since pinned memory resides in host (CPU) memory, accessing it from the GPU requires data transfer over the PCIe bus. While pinned memory allows zero-copy GPU access, it still incurs higher latency than on-device memory due to PCIe bus transfer. This demonstrates that pinned memory is most effective when the majority of processing occurs on the CPU, with only partial tasks offloaded to the GPU.

Unified memory and device memory show similar latency for most elements. This is expected since, in the case of unified memory, the physical location of shared memory depends on the access pattern. In a live video analytics pipeline, where the GPU is the primary consumer of data, CUDA proactively migrates the data to GPU memory ahead of time. Once the first few frames are transferred to GPU memory, subsequent frames would be placed directly in GPU memory if the memory system predicts that the GPU will continue needing this data. Device memory is already on GPU memory, providing the fastest access for GPU operations. The choice between unified and device memory depends on specific workload characteristics and whether CPU access is also required. The elements toward the end of the pipeline, such as the inference engine and tracker, show relatively consistent latency across memory types. This is due to the effective caching of video frames in GPU memory by the time they are processed by these later-stage elements.

8 Conclusion

We presented the implementation details of a scalable real-time video analytics system, PAVE, that enhances pedestrians' safety by leveraging street cameras and distributed processing across edge servers and mobile devices. PAVE detects and tracks pedestrians and vehicles, predicting potential danger zones and delivering timely warnings via a custom mobile application. Through detailed profiling, we show that optimizing memory configurations and resource allocation can reduce components' latency significantly, enabling efficient deployment for time-critical applications such as pedestrian safety. Future work includes equipping PAVE with automated optimization of video streams' resolution, frame rate, and encoding to enhance performance.

Acknowledgments

This work was supported in part by NSF and Center for Smart Streetscapes (CS3) under NSF Cooperative Agreement EEC-2133516, NSF grant CNS-2038984 and corresponding support from the Federal Highway Administration (FHWA), NSF grant CNS-2148128 and by funds from federal agency and industry partners as specified in the Resilient & Intelligent NextG Systems (RINGS) program, NSF grant CNS-2450567, and by computing resources provided by the NVIDIA Academic Grant Program and the Empire AI Consortium.

References

[1] Shivang Aggarwal, Ramanujan K Sheshadri, Karthikeyan Sundaresan, and Dimitrios Koutsonikolas. 2022. Is wifi 802.11 mc fine time measurement ready for prime-time localization?. In *Proc. ACM WiNTECH*.

[2] Martin Ahrnbom, Ivar Persson, Håkan Ardö, and Mikael Nilsson. 2022. Generalized urban traffic surveillance (GUTS): World-coordinate tracking for traffic safety applications. In *Proc. IEEE ITSC*.

[3] Temitope Ibrahim Amosa, Patrick Sebastian, Lila Iznita Izhar, Oladimeji Ibrahim, Lukman Shehu Ayinla, Abdulrahman Abdullah Bahashwan, Abubakar Bala, and Yau Alhaji Samaila. 2023. Multi-camera multi-object tracking: a review of current trends and future advances. *Neurocomput.* 552 (2023), 126558.

[4] Emery D Berger, Sam Stern, and Juan Altmayer Pizzorno. 2023. Triangulating python performance issues with SCALENE. In *USENIX OSDI*.

[5] Shariq Farooq Bhat, Reiner Birkl, Diana Wofk, Peter Wonka, and Matthias Müller. 2023. Zoedepth: Zero-shot transfer by combining relative and metric depth. *arXiv preprint arXiv:2302.12288* (2023).

[6] Henry Bradler, Adrian Kretz, and Rudolf Mester. 2021. Urban Traffic Surveillance (UTS): A fully probabilistic 3D tracking approach based on 2D detections. In *Proc. IEEE IV*.

[7] David V Budescu and Thomas S Wallsten. 1985. Consistency in interpretation of probabilistic phrases. *Organizational behavior and human decision processes* 36, 3 (1985), 391–405.

[8] Difei Cao, Jinsun Yoo, Zhuangdi Xu, Enrique Saurez, Harshit Gupta, Tushar Krishna, and Umakishore Ramachandran. 2022. MicroEdge: a multi-tenant edge cluster system architecture for scalable camera processing. In *Proc. ACM/IFIP Middleware*.

[9] COSMOS Project. 2025. Hardware: Cameras. https://wiki.cosmos-lab.org/wiki/Hardware/Cameras.

[10] Sandeep Dsouza, Victor Bahl, Lixiang Ao, and Landon P Cox. 2020. Amadeus: Scalable, privacy-preserving live video analytics. *arXiv preprint arXiv:2011.05163* (2020).

[11] Salah Fakhoury and Karim Ismail. 2023. Improving pedestrian safety using ultra-wideband sensors: a study of time-to-collision estimation. *Sensors* 23, 8 (2023), 4171.

[12] Youssouph Faye, Francescomaria Faticanti, Shubham Jain, and Francesco Bronzino. 2024. VideoJam: Self-balancing architecture for live video analytics. In *Proc. IEEE/ACM SEC*.

[13] Juan Fumero, Florin Blanaru, Athanasios Stratikopoulos, Steve Dohrmann, Sandhya Viswanathan, and Christos Kotselidis. 2023. Unified shared memory: Friend or foe? Understanding the implications of unified memory on managed heaps. In *Proc. ACM SIGPLAN MPLR*.

[14] Naiyu Gao, Fei He, Jian Jia, Yanhu Shan, Haoyang Zhang, Xin Zhao, and Kaiqi Huang. 2022. Panopticdepth: A unified framework for depth-aware panoptic segmentation. In *Proc. IEEE/CVF CVPR*.

[15] Yuan Gao, Kaifeng Yang, Yibing Yue, and Yunfeng Wu. 2025. A vehicle trajectory prediction model that integrates spatial interaction and multiscale temporal features. *Sci. Rep.* 15, 1 (2025), 8217.

[16] Hadi Ghahremannezhad, Hang Shi, and Chengjun Liu. 2022. Real-time accident detection in traffic surveillance using deep learning. In *Proc. IEEE IST*.

[17] Mahshid Ghasemi, Sofia Kleisarchaki, Thomas Calmant, Levent Gürgen, Javad Ghaderi, Zoran Kostic, and Gil Zussman. 2022. Real-time camera analytics for enhancing traffic intersection safety. In *Proc. ACM MobiSys*.

[18] Mahshid Ghasemi, Sofia Kleisarchaki, Thomas Calmant, Jiawei Lu, Shivam Ojha, Zoran Kostic, Levent Gürgen, Gil Zussman, and Javad Ghaderi. 2023. Real-time multi-camera analytics for traffic information extraction and visualization. In *Proc. IEEE PerCom Workshops*.

[19] M. Ghasemi, Z. Kostic, J. Ghaderi, and G. Zussman. 2021. Auto-SDA: Automated video-based social distancing analyzer. In *Proc. ACM HotEdgeVideo*.

[20] Mahshid Ghasemi, Zoran Kostic, Javad Ghaderi, and Gil Zussman. 2024. EdgeCloudAI: Edge-cloud distributed video analytics. In *Proc. ACM MobiCom*.

[21] Mahshid Ghasemi, Zoran Kostic, Javad Ghaderi, and Gil Zussman. 2025. Systems and Methods for Adaptive Streaming for Real-Time Video Analytics (ASTRA). U.S. Patent Application No. 19/179,836. Patent pending.

[22] Mahshid Ghasemi, Zhengye Yang, Mingfei Sun, Hongzhe Ye, Zihao Xiong, Javad Ghaderi, Zoran Kostic, and Gil Zussman. 2023. Video-based social distancing: Evaluation in the cosmos testbed. *IEEE IoT* 11, 3 (2023), 4987–4997.

[23] Ila Gokarn, Yigong Hu, Tarek Abdelzaher, and Archan Misra. 2024. JIGSAW: Edge-based streaming perception over spatially overlapped multi-camera deployments. In *Proc. IEEE ICME*.

[24] Ila Gokarn, Hemanth Sabbella, Yigong Hu, Tarek Abdelzaher, and Archan Misra. 2023. Mosaic: Spatially-multiplexed edge AI optimization over multiple concurrent video sensing streams. In *Proc. ACM MMSys*.

[25] Fei Guan, Hao Xu, and Yuan Tian. 2023. Evaluation of roadside LiDAR-based and vision-based multi-model all-traffic trajectory data. *Sensors* 23, 12 (2023), 5377.

[26] Jan Hauke and Tomasz Kossowski. 2011. Comparison of values of Pearson's and Spearman's correlation coefficients on the same sets of data. *Quaestiones geographicae* 30, 2 (2011), 87–93.

[27] Junwen He, Yifan Wang, Lijun Wang, Huchuan Lu, Bin Luo, Jun-Yan He, Jin-Peng Lan, Yifeng Geng, and Xuansong Xie. 2023. Towards deeply unified depth-aware panoptic segmentation with bi-directional guidance learning. In *Proc. IEEE/CVF ICCV*.

[28] Jörg Herter. 2014. *Timing-predictable memory allocation in hard real-time systems*.

[29] Matthew Hertz and Emery D Berger. 2005. Quantifying the performance of garbage collection vs. explicit memory management. In *Proc. ACM SIGPLAN OOPSLA*.

[30] Ying Huang, Xiaoying Zheng, and Yongxin Zhu. 2023. Optimized CPU–GPU collaborative acceleration of zero-knowledge proof for confidential transactions. *J. Syst. Arch.* 135 (2023), 102807.

[31] Alireza Jafari and Yen-Chen Liu. 2024. Pedestrians' safety using projected time-to-collision to electric scooters. *Nat. commun.* 15, 1 (2024), 5701.

[32] Gaurav Jain, Basel Hindi, Zihao Zhang, Koushik Srinivasula, Mingyu Xie, Mahshid Ghasemi, Daniel Weiner, Sophie Ana Paris, Xin Yi Therese Xu, Michael Malcolm, et al. 2024. StreetNav: Leveraging street cameras to support precise outdoor navigation for blind pedestrians. In *Proc. ACM UIST*.

[33] Jitesh Jain, Jiachen Li, Mang Tik Chiu, Ali Hassani, Nikita Orlov, and Humphrey Shi. 2023. Oneformer: One transformer to rule universal image segmentation. In *Proc. IEEE/CVF CVPR*.

[34] Samvit Jain, Xun Zhang, Yuhao Zhou, Ganesh Ananthanarayanan, Junchen Jiang, Yuanchao Shu, Paramvir Bahl, and Joseph Gonzalez. 2020. Spatula: Efficient cross-camera video analytics on large camera networks. In *Proc. IEEE/ACM SEC*.

[35] Si Young Jang, Boyan Kostadinov, and Dongman Lee. 2021. Microservice-based edge device architecture for video analytics. In *Proc. IEEE/ACM SEC*.

[36] Kasthuri Jayarajah, Dhanuja Wanniarachchige, Tarek Abdelzaher, and Archan Misra. 2022. ComAI: Enabling lightweight, collaborative intelligence by retrofitting vision DNNs. In *Proc. IEEE INFOCOM*.

[37] EunJin Jeong, Jangryul Kim, and Soonhoi Ha. 2022. Tensorrt-based framework and optimization methodology for deep learning inference

on jetson boards. *ACM TECS* 21, 5 (2022), 1–26.
[38] Junchen Jiang, Ganesh Ananthanarayanan, Peter Bodik, Siddhartha Sen, and Ion Stoica. 2018. Chameleon: Scalable adaptation of video analytics. In *Proc. ACM SIGCOMM*.
[39] Rui Jiang, Hongyun Xu, Gelian Gong, Yong Kuang, and Zhikang Liu. 2022. Spatial-temporal attentive LSTM for vehicle-trajectory prediction. *ISPRS Int. J. Geo-Inf.* 11, 7 (2022), 354.
[40] Linyi Jin, Jianming Zhang, Yannick Hold-Geoffroy, Oliver Wang, Kevin Blackburn-Matzen, Matthew Sticha, and David F Fouhey. 2023. Perspective fields for single image camera calibration. In *Proce. IEEE/CVF CVPR*.
[41] R. E. Kalman. 1960. A New Approach to Linear Filtering and Prediction Problems. *J. Basic Eng.* 82, 1 (1960), 35–45.
[42] Katarzyna Kosek-Szott, Szymon Szott, Wojciech Ciezobka, Maksymilian Wojnar, Krzysztof Rusek, and Jonathan Segev. 2025. Indoor Positioning with Wi-Fi Location: A Survey of IEEE 802.11 mc/az/bk Fine Timing Measurement Research. *arXiv preprint arXiv:2509.03901* (2025).
[43] R. A. Light. 2017. Mosquitto: Server and client implementation of the MQTT protocol. *J. Open Source Software* 2, 13 (2017), 265.
[44] Mao Lin, Keren Zhou, and Pengfei Su. 2023. Drgpum: Guiding memory optimization for GPU-accelerated applications. In *Proc. ACM ASPLOS*.
[45] LinuxTV.org. 2025. Video4Linux buffer API documentation. https://linuxtv.org/downloads/v4l-dvb-apis/uapi/v4l/buffer.html.
[46] Hansi Liu, Hongsheng Lu, Kristin Data, and Marco Gruteser. 2023. ViFi-Loc: Multi-modal pedestrian localization using GAN with camera-phone correspondences. In *Proc. ICMI*.
[47] Shengzhong Liu, Xinzhe Fu, Maggie Wigness, Philip David, Shuochao Yao, Lui Sha, and Tarek Abdelzaher. 2022. Self-cueing real-time attention scheduling in criticality-aware visual machine perception. In *Proc. IEEE RTAS*.
[48] Shengzhong Liu, Tianshi Wang, Jinyang Li, Dachun Sun, Mani Srivastava, and Tarek Abdelzaher. 2022. Adamask: Enabling machine-centric video streaming with adaptive frame masking for DNN inference offloading. In *Proc. ACM MM*.
[49] Rui Lu, Chuang Hu, Dan Wang, and Jin Zhang. 2022. GEMINI: A real-time video analytics system with dual computing resource control. In *Proc. IEEE/ACM SEC*.
[50] Daniel Ma, Ren Zhong, and Weisong Shi. 2025. ICanC: Improving camera-based object detection and energy consumption in low-illumination environments. *arXiv preprint arXiv:2503.00709* (2025).
[51] A Masiero, P Dabove, V Di Pietra, M Piragnolo, A Vettore, A Guarnieri, C Toth, V Gikas, H Perakis, K-W Chiang, et al. 2022. A Comparison Between UWB and Laser-based Pedestrian Tracking. *Int. Arch. Photogramm. Remote Sens. Spatial Inf. Sci. (ISPRS)* 43 (2022), 839–844.
[52] Mind Mingles. 2025. Complete Guide to How Often Google Maps Updates: Frequency, Statistics, and Processes. https://www.mindmingles.com/how-often-google-maps-updates/.
[53] Biswajeeban Mishra and Attila Kertesz. 2020. The use of MQTT in M2M and IoT systems: A survey. *IEEE Access* 8 (2020), 201071–201086.
[54] Payal Mittal. 2024. A comprehensive survey of deep learning-based lightweight object detection models for edge devices. *Artif. Intell. Rev.* 57, 9 (2024), 242.
[55] Keivan Nalaie, Renjie Xu, and Rong Zheng. 2022. Deepscale: Online frame size adaptation for multi-object tracking on smart cameras and edge servers. In *Proc. IEEE/ACM IoTDI*.
[56] New York City Department of Transportation. 2025. Advanced traveler information systems (ATIS). https://www.nyc.gov/html/dot/html/motorist/atis.shtml.
[57] Bingbing Nie, Quan Li, Shun Gan, Bobin Xing, Yuan Huang, and Shengbo Eben Li. 2021. Safety envelope of pedestrians upon motor vehicle conflicts identified via active avoidance behaviour. *Sci. Rep.* 11, 1 (2021), 3996.
[58] Wei Niu, Jiexiong Guan, Yanzhi Wang, Gagan Agrawal, and Bin Ren. 2021. DNNFusion: Accelerating deep neural networks execution with advanced operator fusion. In *Proc. ACM SIGPLAN PLDI*.
[59] NVIDIA Corporation. 2025. NVIDIA TensorRT documentation. https://docs.nvidia.com/deeplearning/tensorrt/.
[60] Jiwoong Park and Young-Bae Ko. 2024. Pedloc: UWB-based pedestrian localization for autonomous vehicles. *Internet Things* (2024), 101194.
[61] Pozyx. 2023. UWB versus other tracking technologies in 2024. https://www.pozyx.io/newsroom/uwb-versus-other-technologies. Industry report.
[62] Jiaming Qiu, Ruiqi Wang, Brooks Hu, Roch Guérin, and Chenyang Lu. 2024. Optimizing edge offloading decisions for object detection. In *Proc. IEEE/ACM SEC*.
[63] Qorvo, Inc. 2022. *DWM3001CDK Quick Start Guide (Rev. B, May 2022)*. https://www.qorvo.com/products/p/DWM3001CDK Document number: DWM3001CDK QSG Rev. B.
[64] Zillur Rahman, Amit Mazumder Ami, and Muhammad Ahsan Ullah. 2020. A real-time wrong-way vehicle detection based on YOLO and centroid tracking. In *Proc. IEEE TENSYMP*.
[65] Scott Schaefer, Travis McPhail, and Joe Warren. 2006. Image deformation using moving least squares. In *ACM SIGGRAPH*. 533–540.
[66] Philip Sedgwick. 2014. Spearman's rank correlation coefficient. *Bmj* 349 (2014).
[67] Neda Seifi and Abdullah Al-Mamun. 2024. Optimizing memory access efficiency in CUDA kernel via data layout technique. *J. Comput. Commun.* 12, 5 (2024), 124–139.
[68] Kiyoung Shin, Ryan McConville, Oussama Metatla, Minhye Chang, Chiyoung Han, Junhaeng Lee, and Anne Roudaut. 2022. Outdoor localization using BLE RSSI and accessible pedestrian signals for the visually impaired at intersections. *Sensors* 22, 1 (2022), 371.
[69] Manavjeet Singh, Sri Pramodh Rachuri, Bryan Bo Cao, Abhinav Sharma, Venkata Bhumireddy, Francesco Bronzino, Samir R Das, Anshul Gandhi, and Shubham Jain. 2024. OVIDA: Orchestrator for video analytics on disaggregated architecture. In *Proc. IEEE/ACM SEC*.
[70] Wim Taymans, Steve Baker, Andy Wingo, Rondald S Bultje, and Stefan Kost. 2013. Gstreamer application development manual (1.2. 3). *Publicado en la Web* 72 (2013).
[71] Technical City. 2025. Jetson AGX Orin 64 GB vs A100 PCIe. https://technical.city/en/video/A100-PCIe-vs-Jetson-AGX-Orin-64-GB. Accessed: 2025-10-21.
[72] Juan Terven and Diana Cordova-Esparza. 2023. A comprehensive review of YOLO: From YOLOv1 to YOLOv8 and beyond. *arXiv preprint arXiv:2304.00501* (2023).
[73] TomTom International BV. 2025. TomTom definitions of the different traffic condition values in the flow feed. https://developer.tomtom.com/traffic-api/documentation/traffic-flow/flow-segment-data.
[74] TomTom International BV. 2025. TomTom maps APIs. https://developer.tomtom.com/maps-api.
[75] TomTom International BV. 2025. TomTom Real time historical traffic. https://download.tomtom.com/open/crm/lib/docs/licensing/RTTHT.EN.pdf.
[76] TomTom International BV. 2025. TomTom traffic APIs. https://www.tomtom.com/products/traffic-apis/.
[77] Duojie Weng, Wu Chen, Mengyu Ding, Simin Liu, and Jingxian Wang. 2025. Sidewalk matching: a smartphone-based GNSS positioning technique for pedestrians in urban canyons. *Satell. Navig.* 6, 1 (2025), 4.
[78] Baofu Wu, Yuankai He, Zheng Dong, Jian Wan, Jilin Zhang, and Weisong Shi. 2022. To turn or not to turn, SafeCross is the answer. In *Proc. IEEE ICDCS*.

[79] Hao Wu, Xuejin Tian, Minghao Li, Yunxin Liu, Ganesh Ananthanarayanan, Fengyuan Xu, and Sheng Zhong. 2021. PECAM: Privacy-enhanced video streaming and analytics via securely-reversible transformation. In *Proc. ACM MobiCom*.

[80] Zhujun Xiao, Zhengxu Xia, Haitao Zheng, Ben Y Zhao, and Junchen Jiang. 2021. Towards performance clarity of edge video analytics. In *Proc. IEEE/ACM SEC*.

[81] Lihe Yang, Bingyi Kang, Zilong Huang, Xiaogang Xu, Jiashi Feng, and Hengshuang Zhao. 2024. Depth anything: Unleashing the power of large-scale unlabeled data. In *Proc. IEEE/CVF CVPR*.

[82] S. Yang, E. Bailey, Z. Yang, J. Ostrometzky, G. Zussman, I. Seskar, and Z. Kostic. 2020. COSMOS smart intersection: Edge compute and communications for bird's eye object tracking. In *Proc. IEEE PerCom SmartEdge*.

[83] Shuochao Yao, Jinyang Li, Dongxin Liu, Tianshi Wang, Shengzhong Liu, Huajie Shao, and Tarek Abdelzaher. 2020. Deep compressive offloading: Speeding up neural network inference by trading edge computation for network latency. In *Proc. ACM SenSys*.

[84] Wei Yin, Chi Zhang, Hao Chen, Zhipeng Cai, Gang Yu, Kaixuan Wang, Xiaozhi Chen, and Chunhua Shen. 2023. Metric3d: Towards zero-shot metric 3d prediction from a single image. In *Proc. IEEE/CVF ICCV*.

[85] Miao Zhang, Fangxin Wang, and Jiangchuan Liu. 2022. CASVA: Configuration-adaptive streaming for live video analytics. In *Proc. IEEE INFOCOM*.

[86] Pengfei Zhu, Longyin Wen, Dawei Du, Xiao Bian, Heng Fan, Qinghua Hu, and Haibin Ling. 2021. Detection and tracking meet drones challenge. *IEEE Trans. Pattern Anal. Mach. Intell.* 44, 11 (2021), 7380–7399.