WatchDog: Real-time Vehicle Tracking on Geo-distributed Edge Nodes

ZHENG DONG, Wayne State University
YAN LU, New York University
GUANGMO TONG, University of Delaware
YUANCHAO SHU, Microsoft Research Redmond
SHUAI WANG, Southeast University
WEISONG SHI, Wayne State University

Vehicle tracking, a core application to smart city video analytics, is becoming more widely deployed than ever before thanks to the increasing number of traffic cameras and recent advances in computer vision and machine-learning. Due to the constraints of bandwidth, latency, and privacy concerns, tracking tasks are more preferable to run on edge devices sitting close to the cameras. However, edge devices are provisioned with a fixed amount of computing budget, making them incompetent to adapt to time-varying and imbalanced tracking workloads caused by traffic dynamics. In coping with this challenge, we propose WatchDog, a real-time vehicle tracking system that fully utilizes edge nodes across the road network. WatchDog leverages computer vision tasks with different resource-accuracy tradeoffs, and decomposes and schedules tracking tasks judiciously across edge devices based on the current workload to maximize the number of tasks while ensuring a provable response time-bound at each edge device. Extensive evaluations have been conducted using real-world city-wide vehicle trajectory datasets, achieving exceptional tracking performance with a real-time guarantee.

CCS Concepts: • **Computer systems organization → Embedded systems**; *Real-time system architecture;*

Additional Key Words and Phrases: Edge computing, neural networks, real-time system, road network

ACM Reference format:
Zheng Dong, Yan Lu, Guangmo Tong, Yuanchao Shu, Shuai Wang, and Weisong Shi. 2023. WatchDog: Real-time Vehicle Tracking on Geo-distributed Edge Nodes. *ACM Trans. Internet Things* 4, 1, Article 2 (February 2023), 23 pages.
https://doi.org/10.1145/3549551

This work was supported in part by the U.S. National Science Foundation under Grants CNS-2140346, CNS-2103604, and in part by a start-up Grant from Wayne State University.

Authors' addresses: Z. Dong and W. Shi, Wayne State University, 42 W Warren Ave, Detroit, Michigan, 48202, USA; emails: {dong, weisong}@wayne.edu; Y. Lu, New York University, 27 West Fourth Street, New York, New York, 10012, USA; email: jasonengineer@hotmail.com; G. Tong, University of Delaware, 210 South College Ave. Newark, Delaware, 19716, USA; email: amotong@udel.edu; Y. Shu, Microsoft Research Redmond, 14820 NE 36th St, Redmond, Washington, 98052, USA; email: yuanchao.shu@microsoft.com; S. Wang, Southeast University, Dongnandaxue Road 2, Nanjing City, Jiangsu, 211189, China; email: shuaiwang@seu.edu.cn.

2577-6207/2023/02-ART2 $15.00
https://doi.org/10.1145/3549551

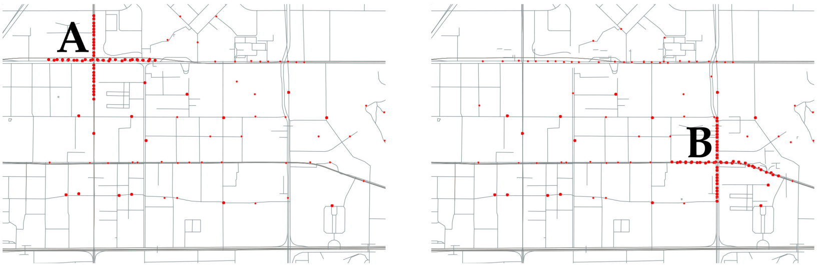
1 INTRODUCTION

Smart city traffic safety initiatives are springing up across the world as more cities embrace big data and video analytics. A straightforward solution of smart city data analytics is to aggregate data and conduct centralized processing in the cloud. This paradigm, however, has several downsides. First, video data uploading requires a substantial amount of network bandwidth, especially under the increasing number of high-resolution cameras. Second, cloud-based processing adds up latency, which could be prohibitively high for smart city applications such as amber alerts or traffic light control. Third, privacy becomes more of an issue when public information is uploaded and stored in the cloud.

In coping with such challenges, an increasing number of smart edge devices, such as Azure Stack Edge [1], Argonne Waggle, and Intel Fog Reference, are being developed and deployed in cities around the world. Such smart edge devices enable fast local computation and have benefited security surveillance systems such as induction coil system [32], traffic surveillance system [6], and the suspicious object monitoring system [23]. The shift in computing paradigm from the cloud to the edge has also necessitated the adoption of new programming models, algorithms, and analytics methods to fully exploit the computing capacity of multi-core chips deployed on the edge. Edge node manufacturers and application developers are starting to discover ways to multiplex tasks and share resources across nodes in an edge cluster [10, 21, 25]. These advanced edge computing approaches provide new possibilities for designing real-time video analytics systems, which leverage machine-learning for tasks like object detection and re-identification. In this article, we focus on multi-camera vehicle tracking, a core application of the smart city video analytics system, and study how to leverage geo-distributed edge nodes to build a reliable real-time tracking system.

Motivation: Edge nodes are provisioned with a fixed compute budget. For instance, Azure Stack Edge [1] node has 2 × 10 core CPUs and 128 GB memory. Even though they are beefy enough to handle computational demands for real-time data processing in most cases, in some corner cases, when the number of vehicles appearing in the monitored areas is too large, the corresponding data processing may not be able to complete in time, leading to a fatal failure for the whole system. Figure 1 shows the snapshots of traffic conditions during different rush hours in Shenzhen. Suppose that a real-time tracking system is implemented on the edge node at each intersection to track hit-and-run vehicles (Assuming that the hit-and-run accident is detected and reported instantly in smart cities, and the tracking system can obtain the location of the accident and the information of the VoI immediately). As seen in Figure 1(a), at most of the intersections, only one or two vehicles appear in the monitored areas and, hence, the tracking system can run complex vision algorithms on each of the vehicles and identify **Vehicle-of-Interest** (**VoI**) easily. However, during morning rush hours, the number of vehicles at intersection A grows substantially. Thus, identifying VoI from all vehicles traveling through intersection A in real-time requires far more computing resources, which may exceed the computing capacity of the edge node. If the VoI cannot be identified at intersection A, the tracking system will lose the VoI. Similar patterns are observed at intersection B, where the corresponding edge node falls short during evening rush hours. On the other hand, computing budget provision on all edge nodes based on the worst-case scenario is also a non-starter due to the high upfront investment and low resource utilization. In light of the tension between traffic variations and the fixed amount of edge computing resources, this article aims at answering a simple question: Can we collaboratively utilize the existing geo-distributed edge nodes deployed in the city to provide a reliable tracking system without "tracking loss" at crowded intersections?

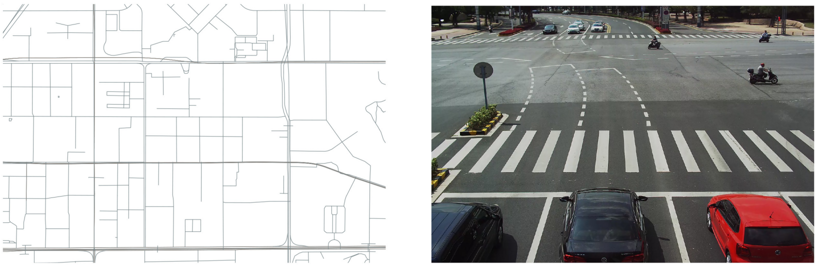
Inspired by the idea of collaborative tracking (i.e., the tacking task will be decomposed and scheduled judiciously across the distributed edge nodes based on real-time traffic conditions), in



(a) Morning rush hour

(b) Evening rush hour

Fig. 1. Snapshots of traffics during different rush hours in the city of Shenzhen, China. Red dots denote vehicles traveling in this area.



(a) Road Network

(b) Real-time traffic information.

Fig. 2. A road network and surveillance video at an intersection.

this article, we propose the design of an intelligent real-time tracking system in smart cities, named WatchDog, to track vehicles across intersections. There are two major components in WatchDog: *a real-time admission control policy* and *a novel dynamic Vehicle—ReID framework*. WatchDog models video analytics executed on an edge node to ReID the VoI as a real-time system: each real-time task corresponds to a vehicle re-identification module performed on a detected vehicle. The execution time of a real-time task depends on the vehicle re-identification module chosen from the dynamic Vehicle—ReID framework, and whether or not each real-time task can complete in real-time is verified by the admission control policy. The longer the tasks execute, the less chance the real-time task system has to pass the admission control. By combining the real-time admission control policy with the dynamic Vehicle—ReID framework together, WatchDog builds a reliable tracking system without "tracking loss" at crowded intersections.

Example. Suppose a Mercedes silver GLB-SUV (which is the VoI) is involved in a hit-and-run accident. When it enters a crowded intersection, the computation capacity on the edge node is not enough to perform the most fine-grained re-identification method to check each vehicle and identify VoI. Then the Vehicle—ReID module is downgraded to a coarse but more lightweight method to identify the vehicles' colors and models, which is determined by the admission control policy. The tracking system detects all silver SUVs at the crowded intersection, which are traveling to different neighboring intersections. The tracking system informs the edge nodes to ReID the VoI at corresponding intersections. When the silver SUVs enter the intersections where the traffic is light, the VoI will be identified through advanced matching methods, and the other silver SUVs will be eliminated. Following this method, the "tracking loss" issue is solved. Our specific contributions are listed as follows:

- We propose a simple yet effective real-time system for tracking hit-and-run vehicles in smart cities, which for the first time enables us to collaboratively utilize the distributed edge resources in the road network to enhance the performance of the whole tracking system.
- To the best of our knowledge, this is the first in-depth work to investigate the combined effect of video processing latency and real-time traffic conditions, which are not discussed in the existing solutions. A real-time admission control policy and a novel dynamic Vehicle—ReID framework are proposed to resolve the "tracking loss" issue.
- We have extensively evaluated WatchDog using our accessible real-world vehicle system-wide datasets. Experimental results show that WatchDog can achieve exceptional tracking performance in real-time without "tracking loss".

2 RELATED WORK

Not surprisingly, a large number of works have been conducted to understand intelligent urban edge computing with public security surveillance systems, including intelligent transportation system realization [38, 53, 59], traffic flow analytics [3, 34, 48], and driving routes optimization [17, 32, 46]. However, in our work, we consider building a real-time vehicle tracking system, which fully utilizes edge nodes across the road network. Since the topic is novel, similar studies on real-time vehicle tracking with video surveillance cameras are few, but include [7, 36, 54].

Especially, [54] designs and implements a self-adapting hit-and-run vehicle tracking algorithm with distributed sparse video surveillance cameras and mobile taxicabs, leveraging time-varying characteristics of road traffic flow patterns. By mining the massive trajectory dataset of taxicabs and a video dataset of surveillance cameras, the travel time-cost of a road segment during a specific time period is modeled using a Logarithmic Normal Distribution, which calculates the time-cost of an urban trip during a specific time period with a Log Skew Normal Distribution approximately. This work trains the model offline and does not consider resource management while tracking a VoI in real-time. With the emergence of vehicle-mounted GPS navigation systems and vehicular networks, a collection of series of GPS coordinates became available for vehicle tracking. Based on the GPS records, [36] proposes a novel global map-matching algorithm utilizing the spatial geometric/topological structures of the road network as well as the temporal/speed constraints of the trajectories. The basic assumption, true paths of vehicles tend to be direct rather than roundabout, is used to enhance the accuracy of computing actual vehicle trajectories. But for VoIs, if they choose abnormal paths to avoid the tracking system, it will increase the difficulty of trajectory recovery. Reference [7] develops an efficient machine learning-based method for vehicle detection and motion analysis in the low-altitude airborne platform. This work has not considered cooperative computing with multiple video cameras, since the deployments of fixed video cameras are still distributed, sparse, and cannot support the seamless monitoring of the VoI in nature.

Specific to the problem investigated in this article, existing studies on recovering detailed trajectories of vehicles with GPS coordinates are designed for normal vehicles, which cannot be used to track VoIs. Moreover, in practice, we cannot even obtain any GPS information of VoIs and videos from the taxicabs, and the information from the public security surveillance system and the computing capacity on the edge nodes are the only resources we can leverage.

3 BASIC SETUP AND SYSTEM MODEL

3.1 Basic Setup

Given the road network in the urban area of a smart city, this article aims at finding a method to track the VoI (e.g., hit-and-run vehicles) by combining the information from fixed video surveillance cameras and the road network, thereby helping the tracking system track the VoI in real-time using minimum edge resources. The basic settings of this article are outlined as follows:

- **Road Network:** Figure 2 shows a road network in the urban area of Shenzhen with intersections and road segments. Surveillance cameras are deployed at the intersections to capture real-time traffic information.
- **Edge nodes:** Consistent with existing deployments, our focus is on "edge" computation of video analytics. In our setup, one edge node is deployed at each intersection, which consists of a surveillance camera and a computing platform. A video captured by the camera is streamed to this edge box and the pipeline modules including object detection and re-identification algorithms are run on this edge node.
- **Cloud:** Each edge node only captures and processes local information at the intersections. In order to get a global view of the entire monitored area, the cloud collects the processing results from all the edge nodes. Hence, the tracking system includes both edge nodes and the cloud.

3.2 System Model

In this subsection, we define the road network and vehicle trajectory used in our system model.

Definition 1 (Road Network). The road network consists of intersections and road segments between intersections, which can be modeled as a graph $G(V, E)$ where the vertex set V denotes all intersections and the edge set E corresponds to all road segments. Intersection $I_i \in V$ and $e_{i,j} \in E$ if there exists a road segment from intersection I_i to intersection I_j.

Definition 2 (Vehicle Trajectory). For an arbitrary vehicle, which travels in the road network, its trajectory in one specific day d can be formulated by $T = \{I_o(t_1, t_2), I_{o+1}(t_3, t_4), \ldots\ldots, I_e(t_x, t_{x+1})\}$, which means this vehicle is firstly detected at intersection I_o, then ends at intersection I_e. For one element $I_y(t_k, t_{k+1})$ in this trajectory, t_k is the time instant that the vehicle enters intersection I_y and time instant t_{k+1} corresponds to the time it leaves this intersection.

Definition 3 (The Tracking System). The tracking system includes both the edge nodes and the cloud. The object recognition algorithms are implemented on the edge nodes. At the very beginning, when a hit-and-run accident is reported to the tracking system, the first edge node is activated instantly according to the location of the accident. During the *active period* of the edge node, it identifies the VoI and the next intersection on the VoI's trajectory based on real-time video analytics. The analysis results are uploaded to the cloud to activate the edge node deployed at the next intersection and stop the previous one. From then on, the next activated edge node and its active period depend on the analysis results performed at the previous intersection and the historical traffic information (which will be discussed in Section 7). The communication between edge nodes is enabled through the cloud.

Intuitively, if the tracking system knows when the VoI will arrive at which intersections in advance, the corresponding edge nodes can be activated during specific periods to track the VoI in real-time. However, in practice, the implementation of real-time tracking is very challenging.

3.3 Tracking Loss Issue

Figure 3 shows an example of a road network consisting of 16 intersections. Suppose a hit-and-run vehicle has a trajectory of $T = \{I_1(t_1, t_2), I_2(t_3, t_4), I_6(t_5, t_6), I_7(t_7, t_8), I_{11}(t_9, t_{10}), I_{12}(t_{11}, t_{12})\}$ in the monitored area. Then, the edge nodes deployed at $I_1, I_2, I_6, I_7, I_{11}, I_{12}$ are involved in tracking the VoI. If we ignore the time periods taken by the vehicle at the intersections, the trajectory T can

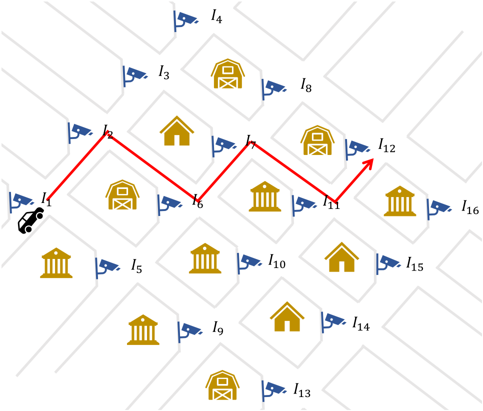


Fig. 3. An example of vehicle real-time tracking.

be considered as a path in the road network, i.e., $I_1 \to I_2 \to I_6 \to I_7 \to I_{11} \to I_{12}$. In this article, the tracking system aims at tracking the VoI in real-time: If the VoI enters I_i at time instant t_p, then

- The edge node at I_i must be activated before t_p. Otherwise, the VoI may not be captured by the camera and get lost at I_i.
- The VoI and the next intersection I_j on the VoI's trajectory must be identified before the time instant t_q when the VoI enters I_j. Otherwise, the tracking system cannot activate the edge node at I_j before t_q.

Example 1. For the example in Figure 3, a Mercedes silver GLB-SUV (the VoI) is reported to be involved in a hit-and-run accident at intersection I_1 at time instant t_1, and it arrives at I_2 at t_3. The edge node at I_1 is activated at t_1: The camera captures videos from I_1, and the videos are processed frame-by-frame to identify the VoI and the next intersection on the VoI's trajectory. The processing results must be obtained and sent to the cloud before t_3. Then, the tracking system can activate the edge node at I_2 before t_3; otherwise, the VoI may already depart from I_2 before the edge node starts tracking.

Note that the next intersection on the VoI's trajectory can be easily determined by the VoI's position and the road network based on a series of video frame processing.

If the tracking system tracks the VoI in real-time successfully, the VoI will be located at either an intersection or a road segment between two intersections at any specific time instant when it travels in the monitored area. The tracking problem becomes rather trivial to resolve if only a few vehicles travel in the smart city. It means that whenever the VoI appears at an intersection, it will be recognized by the object identification algorithms instantly due to sufficient computing resources on the edge node. However, the practical scenario is always not the case.

Example 2. In Figure 3, the VoI arrives at intersection I_2 at time instant t_3 and travels to I_6. Assuming that I_2 is a crowded intersection. In this case, it takes a very long time to identify the VoI from dozens of cars traveling through the intersection. The edge node at I_2 may fail to complete the video analytics when the VoI arrives at I_6 at t_5. Then, the tracking system does not know which edge nodes should be activated afterward to carry on the tracking task, and the VoI is lost.

Identified Issue. The computing capacity of an edge node depends on the computing platform used to implement it, which is determined by the hardware manufacturer. For example, an Azure Data Box Edge [1] node is equipped with 2 × 10 core CPUs for data processing. Thus, at the crowded intersections, the tracking system cannot identify the VoI in real-time due to the limited computing capacity of the edge node and the "tracking loss" occurs.

Key Idea of Our Proposed Method. To resolve this "tracking loss" issue, we seek to develop a smart tracking method to make up for the limited computing capacity of a single edge node. At a crowded intersection, since the full-fledged object identification algorithm needs an unaffordable amount of time to precisely find the VoI, the tracking system can use a "coarse" object identification algorithm to save time. Multiple suspected VoIs may be identified by the "coarse" algorithm and travel to different intersections. The tracking system can track all the suspected VoIs simultaneously by utilizing the edge nodes at different intersections and identifying the VoI at the uncrowded intersections. Intuitively, this idea is feasible, because we find that almost 95% of the intersections in a smart city are uncrowded intersections (See the statistics in Section 9.1.2).

In order to implement the real-time tracking system, first, we propose a dynamic **Vehicle re-Identification** (**V-ReID**) framework to realize the Re-ID algorithm at different granularity levels. Then, we introduce a real-time admission control module on each edge node to decide which Re-ID algorithm will be performed to identify the VoI according to the number of vehicles detected at the corresponding intersection. Finally, we will discuss how to obtain the active period for each edge node involved in a hit-and-run tracking event, and how the proposed tracking system, named *WatchDog*, works to track the VoI in real-time.

4 VISION-BASED VEHICLE TRACKING

Tracking in WatchDog relies on computer vision-based machine learning algorithms. Query input is a target vehicle (e.g., from an accident report) with information such as make, model, color, and plate number. Once WatchDog receives a tracking query, the corresponding edge node starts running a video analytics pipeline to analyze traffic videos in real-time. At a high level, the processing pipeline of each frame consists of two modules: vehicle detection and vehicle re-identification.

4.1 Vehicle Detection

For each video frame, vehicle detection targets to find all vehicles and assign a class to each one of them. Unlike classification networks, traditional vehicle detection networks [15, 27, 31, 47] have three components: a CNN-based feature extractor, **Region Proposal Network** (**RPN**), and a classifier. They usually use a pre-trained classification network (e.g., ResNet) as a feature extractor to a generate feature map of an input image. After that, they utilize RPN [15] to generate all candidate bounding boxes of vehicles, and finally assign labels for each bounding box. As those detectors extract all vehicles first and classify each bounding box later, they are called two-stage vehicle detectors. Although two-stage vehicle detectors achieve superior performance on many public benchmarks [2, 4, 39], the speed suffers. To tradeoff performance and latency, researchers seek to design efficient vehicle detectors [29, 30, 51, 55], which use one CNN network to solve localization and classification simultaneously. Albeit marginal performance drop, one-stage detectors largely reduce latency, and, hence, have been widely implemented on today's edge nodes for real-time tracking systems.

4.2 Vehicle Re-Identification

V-ReID determines whether two bounding boxes belong to the same vehicle. The most popular deep learning approach for V-ReID is to build a CNN-based feature extractor and differentiate bounding boxes based on the similarity (e.g., cosine distance) between discriminative feature vectors. Unlike **person re-identification** (**P-ReID**), V-ReID [22, 24, 43, 50, 57] often uses many CNN-based networks to extract different features (e.g., global features, region features, and key point features) and concatenate them for a reliable comparison. For example, researchers often set features of a vehicle's shape as a general feature and window screen as a region feature. As a result, V-ReID models are often very large, and the end-to-end compute process is prohibitively costly, making them not amenable to real-time tasks. For example, V-ReID each one of a large set of vehicle bounding boxes during the rush hours could end up causing "tracking loss" issue. Thus, a V-ReID method that can dynamically tradeoff accuracy with inference time in real-time is desired. To this end, in what follows that we propose a dynamic V-ReID framework called D-V-ReID.

4.3 D-V-ReID Pipeline

D-V-ReID adopts the idea of divisive clustering where V-ReID on each frame follows a multi-layer framework where upper layers correspond to coarse but efficient classifications, and lower layers are in charge of powerful but complex detection. While we need to slightly modify the existing learning algorithm for the purpose of real-time tracking, we do not intend to propose any new learning algorithm in this article but focus on building a flexible V-ReID pipeline. The cascaded pipeline we propose includes: color matching, model and make matching, and the full-fledged deep feature-based re-identification.

4.3.1 Color Matching. As a basic filter, color matching measures similarity between bounding boxes by color distribution in spaces such as RGB or HSI [16]. Given RGB or HSI features, we use **K-Nearest Neighbors** (**KNN**) [12] to decide the similarity of two images. A car is classified by a majority vote of its neighbors, with the car being assigned to the class, which is most common amongst its KNN. The complexity of the inference process in this step is linear to the input frame size, and it can be further reduced by frame down-sizing, which provides a tradeoff between matching quality and inference time where the inference time can be made arbitrarily small. As the most coarse V-ReID step, a matching confirm result does not provide much confidence that the target vehicle is detected because different vehicles may have similar color, but a matching reject serves as strong evidence for the fact that the target vehicle is not in the current frame.

4.3.2 Model and Make Matching. Our second layer aims at detecting if the vehicle in the bounding box has the same model and make as the target vehicle. Model and make of a vehicle are more distinctive, and such information is commonly used for vehicle identification in security systems such as amber alerts. Traditional matching algorithms use **Scale Invariant Feature Transform** (**SIFT**) [37], SURF [20], and HOG [40] feature extraction techniques to extract models' features. Based on these features, Support Vector Machine [41, 44] and Random Forest [52] can matching two cars accurately, but these methods assume that the input image has the whole shape of the vehicle. In practical environments, many cars are occluded by other objects. To resolve this issue, many works [13, 28] seek to use deep learning approaches to generate discriminative features because of the ability to extract features automatically. Unlike general image classification or object detection, model and make matching on vehicle is done by training a light convolutional neural network (ResNet50, MobileNet, ShuffleNet, and EfficientNet [19, 45, 49, 60]). For instance, we could adopt a subset of ResNet for such a purpose. The goal of model and make matching in our framework is still to provide a fairly accurate V-ReID method with moderate inference overhead.

4.3.3 Full-Fledged V-ReID. Deep feature-based V-ReID algorithm serves as the final layer in D-V-ReID. In light of ensemble learning [9], state-of-the-art V-ReID methods [5, 14, 42, 50, 61] are designed with a set of convolutional neural networks, and these neural networks are responsible for different feature extractions. To further improve ReID accuracy, researchers assign unique lose functions [57] to different feature extractors and train them independently. After pre-training, a unified loss function is utilized to train all feature extractors together. This step is also called multitask learning [14, 50, 56, 58]. In particular, these methods always classify features as global, region,

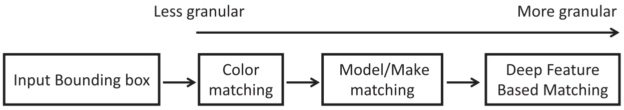


Fig. 4. Layered modules.

Table 1. Cost of Deep Feature Extractors (Global, Region, and Key-Point Features)

	Global	Region	Key-point
Flops	341.5 M	7868.4 M	11392 M
Inference Time	41.1 ms	96.7 ms	172.3 ms
CPU Usage	2.4%	9.8%	15.0%

Table 2. Cost of Different Modules (Color Matching, Model Matching, and Full-Fledged V-ReID)

	Color	Model	Full-fledged V-ReID
Flops	/	341.5 M	19602.3 M
Inference Time	0.5 ms	40.6 ms	310.1 ms
CPU Usage	0.3%	2.2%	27.3%

and key-point features. The global feature is used to describe the overall appearance of the vehicle, which is the linear transformation of the pooling feature of the last convolution module in global feature extractor. Due to the limited discriminative ability of global features, many researchers seek to use multiple granularities network [18] to extract features from the multiple semantic parts of vehicles. Inspired by [61], scientists use another CNN network to predict several key points sit on key parts of the vehicle (e.g., the vehicle license plate), and then extract features around those key parts with the assistance of a heat-map generated from key point network. In our design, we combine features from color matching and model/make matching, and set it as the global feature. The reason for doing so is to provide early exit and reuse computation from previous matchings.

4.4 D-V-ReID Framework

D-V-ReID framework is built upon three V-ReID algorithms with different granularity levels, which is illustrated in Figure 4. As discussed above, we set KNN classifier [12] as our color filter and the architecture of model matching to MobileNetV2 [45]. In terms of the full-blown V-ReID, we adopt the best method [57] in AI City Challenge 2019. They utilized three different convolution neural networks to extract features from the same vehicle and concatenate them as the final feature. As discussed in Section 4.3.3, we combine features from color matching and model/make matching as the global feature. To see the computation cost of region and key-point feature extractors, we set their architectures to ResNet101 [19] and SE-ResNet152 [26], respectively. After re-scaling each bounding box to $224 * 224$, we report **floating point operations per second** (**flops**), inference time, and cpu usage on one bounding box of different feature extractors from deep feature-based matching in Table 1 and different modules in Table 2. The test environment is AMD Ryzen 7 3700x (CPU) with 4G memory. Because KNN classifier is not a deep neural network, we do not record the flops for it.

According to the real-time requirements of the tracking system, it could select the V-ReID modules with a proper complexity to identify the VoI at specific intersections. Intuitively, in order to guarantee that the V-ReID module can complete in real-time, a less granular module will be triggered if a large number of vehicles appear at a crowded intersection. On the contrary, if the intersection is uncrowded, a more granular module will be performed to identify VoI. Note that under our proposed D-V-ReID framework, if a more granular module is selected, all the less granular modules are executed implicitly. It is evident that the granularity selection depends on the number of vehicles captured on each video frame, the inference time of different V-ReID modules, and the computing capacity of the edge node. In next section, we introduce a real-time admission control method, which is implemented on each edge node to select proper V-ReID modules for the tracking system in real-time.

5 REAL-TIME ADMISSION CONTROL

Before discussing the details of our design, we introduce some preliminary results on real-time admission control, which determines the granularity level of the V-ReID modules selected to identify the VoI.

5.1 Real-Time Task Scheduling Framework

In this section, we will introduce a classic soft real-time schedulability test, which can be used to calculate the completion time bounds for real-time tasks scheduled in the real-time system. Based on the completion time-bound of each real-time task, we can perform admission control on each edge node.

5.1.1 Real-Time Task Model. At an arbitrary intersection I_x, the surveillance camera captures frames from the intersection periodically and the *period*, denoted by p, depends on the camera's frequency. For example, if the camera captures 24 frames every second, we would say the video is 24 fps, and its period is $\frac{1}{24}s$. Usually, one or more vehicles may be detected on each frame. The V-ReID machine learning algorithm is performed on all the detected vehicles to identify the VoI, and each identification process corresponds to *a real-time machine learning task*. Let e_i denote the *processing time* of task τ_i performed on vehicle i. Tabel 2 gives the processing time for different V-ReID modules. Let e^c, e^m, e^d denote the processing time for color matching module, model/make a matching module, and Deep Feature-Based Matching module, respectively. Under our proposed D-V-ReID pipeline framework, if a more granular module is selected, all the less granular modules has already been selected implicitly. Thus, e_i has three different options under the D-V-ReID framework: $e_i^1 = e^c, e_i^2 = e_i^1 + e^m, e^3 = e_i^2 + e^d$. As we discussed in Section 7.1, in order to avoid "tracking loss", the VoI must be identified before it arrives at the next intersection (it is illustrated by Example 2).

Definition 4. Let $t_{x,y}$ denote the traveling time of the VoI between two neighboring intersections (i.e., I_x and I_y). I_x may have multiple neighboring intersections, and we define the shortest $t_{x,y}$, denoted by D_x, as the *relative deadline* of the real-time tasks from the edge node deployed at I_x.

Based on the above discussion, we use the periodic hard real-time task model to describe the execution behaviors of real-time workloads in the tracking system on an edge node deployed at I_x. We consider the problem of scheduling n periodic real-time tasks on M processors. That means n vehicles are detected from each image, and the edge node has M processors (note that we use "processor" to denote the minimum schedulable processing unit). A task τ_i is characterized by two parameters—a processing requirement e_i and a period p with the interpretation that the task generates a job (i.e., the camera captures a frame) in every p time units— and each such job $\tau_{i,j}$ has a processing requirement of e_i execution units, which should be met by a deadline $d_{i,j}$. Let $r_{i,j}$ denote the generation time of $\tau_{i,j}$, then $d_{i,j} = r_{i,j} + D_x$. We further let u_i denote the utilization of

τ_i, where $u_i = \frac{e_i}{p}$, and the utilization of the task system τ is defined as $U_{sum} = \sum_{i=1}^{n} u_i$. Successive jobs of the same task are required to execute in sequence. We require $u_i \leq 1$, and $U_{sum} \leq M$; otherwise, deadlines will be missed.

5.1.2 Real-Time Scheduling Algorithm. If the number of tasks is no greater than the number of processors, each identification task can be executed on a dedicated processor. In this case, the computing capacity on the edge node is sufficient to execute the real-time tasks, and the "tracking loss" issue will not happen. However, when the traffic is heavy, the number of vehicles (corresponding to the real-time tasks) detected at the intersection may be much larger than the number of processors on the edge node. The real-time tasks will compete for the limited computing resources on the edge node and have inferences with each other, which may give rise to a huge delay for frame processing and lead to deadline miss. In this case, a scheduling algorithm is needed to allocate processor time to tasks, i.e., determines the execution-time intervals and processors for each job while taking any restrictions, such as on concurrency, into account. In real-time systems, processor-allocation strategies are driven by the need to meet timing constraints and in our real-time tracking system, we apply the **First-in First-Out** (**FIFO**) policy to schedule the real-time tasks: processors execute jobs in the exact order of job arrival.

5.1.3 Completion Time Analysis. A given set of real-time tasks is said to be schedulable on a given system of processors, if the tasks can be scheduled on these processors in such a manner that all jobs of all the tasks always complete by their deadlines. Physically, if all the tasks can complete by their deadlines, the "tracking loss" will not happen. The schedulability can be verified by using standard schedulability analysis (Theorem 1) for calculating the completion times of real-time tasks.

Theorem 1. *Considering that a real-time task system τ of n periodic tasks are scheduled on M processors under FIFO scheduling policy, the completion time bound for a task τ_i is*

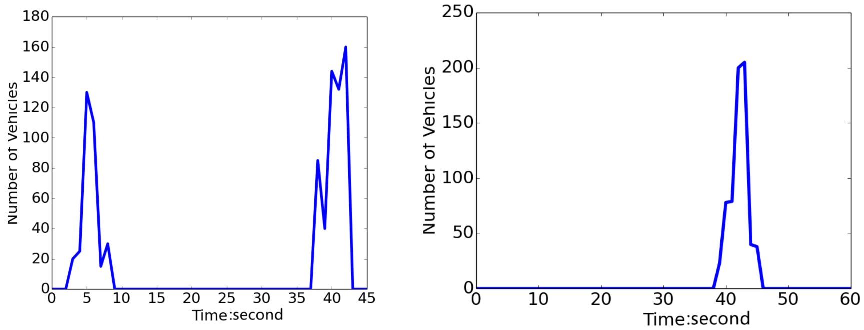
$$R_i = p + e_i + \frac{\sum_{\tau_k \in E(\tau, M-1)} e_k - e_i}{M - \sum_{\tau_j \in U(\tau, M-1)} u_j}, \tag{1}$$

where $E(\tau, M-1)$ denotes the set of at most (M−1) tasks with the highest execution costs from τ, and $U(\tau, M-1)$ denotes the set of at most (M-1) tasks of highest utilization from the task set τ [33] .

The proof of Theorem 1 is given in [33], and the same result can be derived from [11], because the periodic task model is a special case of the stochastic task model discussed in [11]. In light of the real-time task model and the completion time analysis, a formal definition of "tracking loss" is that if the completion time bounds of the real-time tracking tasks exceed their deadlines, the VoI is lost at some intersections.

5.1.4 Real-Time Admission Control. Intuitively, if we can guarantee that the completion time bound R_i for each task τ_i can be no greater than its deadline D_x, the "tracking loss" cannot happen. According to Theorem 1, since p and M are fixed values when the hardware of the edge node is given, the completion time bound R_i of τ_i only depends on the execution times of the machine learning tasks and the number of vehicles detected at the intersection. Therefore, we introduce the following programming to select proper V-ReID modules for the tracking system in real-time.

$$\begin{aligned} \text{maximize} \quad & \sum_{i=1}^{n} e_i \\ \text{subject to} \quad & R_i = p + e_i + \tfrac{\sum_{\tau_k \in E(\tau, M-1)} e_k - e_i}{M - \sum_{\tau_j \in U(\tau, M-1)} u_j} \leq D_x, \\ & e_i \in \left\{ e_i^j, e_i^{j+1} \right\}, \qquad j = 1, 2 \end{aligned} \tag{2}$$



(a) traveling time at an I_x.

(b) traveling time on a road segment.

Fig. 5. Traveling time measurement at different locations. The x-axis denotes the traveling time, and y-axis denotes the number of vehicles traveling through this location.

where e_i denotes the execution time of task i in the jth iteration. According to Tabel 2, the processing time of color matching module is 0.5 ms. We assume that the edge node has enough computing capacity to perform color matching module on all detected vehicles even at the most crowded intersection. Thus, in the first iteration, our objective is to maximize the number of tasks, which will perform the model/make matching module. If all the tasks can meet the deadline, then in the second iteration, we aim at maximizing the number of tasks, which will perform the deep feature matching module. R_i is the response time of task i, which is defined in Theorem 1, and D_x is the relative deadline of every real-time task from I_x, which is defined by Definition 7. We require that each task must complete by its deadline. Since we do not have preferences for the tasks, which will perform in the jth iteration, the time complexity for this programming is $O(n)$ to achieve the optimal solution.

6 ACTIVE PERIODS OF EDGE NODES

In order to track the VoI in real-time, the tracking system should be able to know when the VoI will arrive at which intersections in advance, then the corresponding edge nodes can be activated before the VoI's arrival time and identify the VoI when it appears. In other words, the active period of an involved edge node should cover the time interval when the VoI travels though the corresponding intersection.

If all the intersections are uncrowded, this problem is trivial. For the example in Figure 3, when the hit-and-run accident is reported, the edge node at intersection I_2 is activated to perform the most granular machine learning algorithm on all detected vehicles, and the VoI is identified at time instant t_1. Based on the real-time video analytics, the tracking system finds that the VoI departs from I_2 and travels to I_3 at time instant t_2. Then, the tracking system activates the edge node at I_3 instantly and at the same time stops the edge node at I_2. If the tracking system repeats the same operation on all involved edge nodes, the active period for each edge node can be obtained in real-time.

However, when the VoI enters a crowded intersection, the problem becomes challenging. Assume that intersection I_3 is a crowded intersection in Figure 3, and the VoI (a Mercedes silver GLB SUV) is traveling from I_2 to I_3. The admission control module selects the color matching module to track all the silver vehicles traveling through I_3 based on the number of vehicles detected in the video frames. Multiple silver vehicles may appear in I_3, but the tracking system cannot confirm that whether or not the VoI has arrived at I_3. Because the traveling time on the road segment between I_2 and I_3 is different for different vehicles. Figure 5 shows the traveling time measurement at different locations. On a road segment, the traveling time of vehicles varies within a range from 30 to 50 s (Figure 5(b)). Thus, at a crowded intersection, in order to guarantee that VoI is included in the tracked vehicles (in the example, VoI is one of the tracked silver vehicles), the active period for the edge node is calculated based on historical traffic information.

Definition 5. Let t_x denote the traveling time taken by the VoI at intersection I_x, then $t_x^l \leq t_x \leq t_x^u$, where t_x^l is the lowest value among all the vehicles' traveling time at I_x and t_x^u is the largest value. Both values can be obtained from historical traffic information (For example, Figure 5(a)).

Definition 6. Suppose I_x and I_y are neighbouring intersections. Let $t_{x,y}$ denote the traveling time taken by the VoI at road segment $e_{x,y}$, then $t_{x,y}^l \leq t_{x,y} \leq t_{x,y}^u$, where $t_{x,y}^l$ is the lowest value among all the vehicles' traveling time at $e_{x,y}$, and $t_{x,y}^u$ is the largest value. Both values can be obtained from historical traffic information (For example, Figure 5(b)).

Definition 7. Note that according to Definitions 4 and 6, I_x may have multiple neighbor intersections, and we use the shortest $t_l^{x,y}$ as the *relative deadline* D_x of the real-time tasks from the edge node deployed at I_x.

Based on Definitions 5–7, we can calculate the active period $[t_x^s, t_x^e]$ for an involved edge node at intersection I_x.

- **Case 1:** If the previous intersection I_p on the VoI's trajectory is an uncrowded intersection, then the VoI is identified at I_p. Let t_p denote its departure time from I_p. Then, the earliest time instant when the VoI can arrive at I_x is $t_x^s = t_p + t_{p,x}^l$. Correspondingly, $t_x' = t_p + t_{p,x}^u$ is the latest time instant when the VoI can arrive at I_x. Thus, the latest time instant when the VoI departs from I_x is $t_x' + t_x^u$, which is the time instant when the last video frame captured from I_x may contain the VoI. Thus, the edge node needs D_x time units to process the last frame and the edge node ends up processing video frames at $t_x^e = t_x' + t_x^u + D_x$.
- **Case 2:** If the previous intersection I_p is a crowded intersection, and we assume that the active period of the edge node at I_p is $[t_p^s, t_p^e]$. According to the definition of active period, the earliest time when the VoI enters I_p is no earlier than t_p^s, then the earliest time when the VoI enters I_x is no earlier than $t_x^s = t_p^s + t_x^l + t_{p,x}^l$; the latest time when the VoI departs from I_p is no later than $t_p^e - D_p$, then the latest time when the VoI departs from I_x is no later than $t_p^e - D_p + t_x^u + t_{p,x}^u$. Again, the edge node needs D_x time units to process the last frame and the edge node ends up processing video frames at $t_x^e = t_p^e - D_p + t_x^u + t_{p,x}^u + D_x$.

In light of the above discussion, the active period for the edge node at intersection I_x is calculated based on the active period of the previous edge node. Thus, we need to define the active period for the first edge node at intersection I_o where the hit-and-run accident happens. Let t_o denote the time instant when the accident is reported. Then, whether or not I_o is a crowded intersection, the active period of the edge node at I_o is $[t_o, t_o + t_o^u + D_o]$. Based on I_o's active period, the active periods for all the involved edge nodes can be calculated one-by-one at run time.

7 WATCHDOG

In this section, we put all the proposed techniques together to build the real-time tracking system —WatchDog.

7.1 System Description

Figure 6 illustrates the architecture of WatchDog implemented on each edge node, which consists of four major components: (i) a live video stream generated by the surveillance camera; (ii) D-V-ReID framework; (iii) the Real-time admission control module; and (iv) the Real-time Vehicle ReID program. On each edge node, each frame of the real-time video stream is firstly processed by a vehicle detection module to detect the vehicles traveling through this intersection, which is introduced

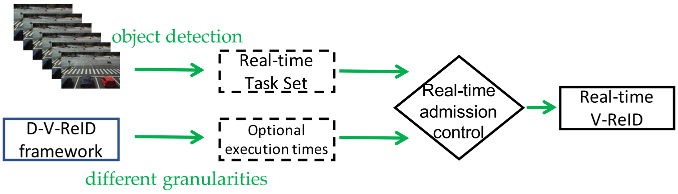


Fig. 6. The architecture of WatchDog implemented on the edge node.

in Section 4.1. A real-time ReID task will be performed on each detected vehicle to identify whether or not the VoI appears. Suppose the system detected n vehicles at this intersection, then the real-time task set contains n tasks. According to the D-V-ReID framework introduced in Section 4.2, each task has several optional execution times, which correspond to different sub-modules, which is introduced in Section 4.3. The longer a real-time task executes, the better ReID performance the real-time task can get. In order to guarantee that each real-time task can complete by its deadline, the real-time admission control module is performed to select the best combination of execution times for real-time tasks to ReID the VoI, according to the optimal solution for the programming, which is given by Equation (2). Intuitively, it is a tradeoff between the ReID performance and the schedulability of the real-time task system. When the modules of ReID program are chosen for each real-time task, the real-time V-ReID starts performing on each frame to track the VoI.

In light of WatchDog's architecture on each edge node, the tracking behavior of WatchDog can be described as follows:

(1) When a hit-and-run accident is reported at time instant t_o^s from intersection I_o, WatchDog activates the first edge node deployed at I_o to track the VoI and its active period is $[t_o^s, t_o^e]$. According to the number of vehicles detected at I_o, the proper machine learning modules are selected. By performing the real-time machine learning tasks on the detected vehicles, all the suspected vehicles and the next intersections on these vehicles' trajectories are identified. If I_o is an uncrowded intersection, the VoI and the next intersection on the VoI's trajectory are identified. Then WatchDog activates all the edge nodes at the next intersections according to their active periods. Again, if I_o is an uncrowded intersection, then only one next edge node will be activated.

(2) The edge node deployed at I_x is activated to track the VoI, if the VoI or some suspected vehicles are identified at its neighboring intersection I_{x-1} and traveling to I_x. The edge node's active period is $[t_x^s, t_x^e]$, which is calculated based on the active period of the previous edge node. The calculation method is introduced in Section 6. Again, similar to the first edge node, the proper machine learning modules are selected and performed to track all suspected vehicles traveling through this intersection during its active period.

(3) Once the VoI is identified at any uncrowded intersection, all the suspected tracking branches are terminated.

Steps 2 and 3 are repeated iteratively to track the VoI in real-time.

Example 3. We use a simple example to illustrate how the whole tracking system works in Figure 7. Suppose a Mercedes silver GLB-SUV is reported to be involved in a hit-and-run accident at intersection I_1 at time instant t_1^s. The edge node at I_1 is activated to track the VoI. According to the number of vehicles detected from each frame, the admission control algorithm selects the most granular algorithm for the real-time ReID tasks to track the VoI and the VoI is found to travel to I_2. Then, the edge node at I_2 is activated during to its active period. I_2 is an uncrowded intersection and the VoI is found to travel to I_6. However, I_6 is a crowded intersection and according

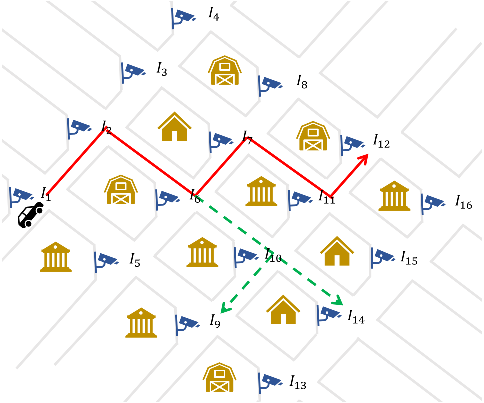


Fig. 7. An illustration example for tracking an VoI.

to the number of vehicles detected from each frame, the admission control model selects color matching module for the real-time ReID tasks to track all the silver vehicles traveling through I_6. At I_6, some silver vehicles are found to travel to I_7, and the others are found to travel to I_{10}. Then, the edge nodes at I_7 and I_{10} are activated to track all the silver vehicles. I_{10} is another crowded intersection, and according to the number of vehicles detected from each frame, the admission control model selects color matching and model/make matching for the ReID tasks to track all suspected vehicles. Fortunately, I_7 is an uncrowded intersection, and the VoI is identified at I_7, then all the other suspected tracking branches are terminated. The following involved edge nodes are activated in the same way to track the VoI in real-time. Note that a corner case is discussed in the captain of Figure 11.

Based on the above discussion, WatchDog can guarantee 100% tracking coverage of the VoI without "tracking loss". We sacrifice the accuracy of ReID algorithms and track all the suspected vehicles at crowded intersections, then identify the VoI by utilizing the computing resource on the edge nodes deployed at uncrowded intersections. In a word, we develop a smart tracking method to make up for the limited computing capacity of a single edge node.

8 EMPIRICAL STUDY

To evaluate the reliance of the proposed image processing methods in realistic scenarios, we conduct extensive experiments based on real-world datasets. We drive a vehicle (VoI) running abnormally in the road network from March 1 to March 30, 2022, then collect captured video information of all fixed video surveillance cameras at the intersections during this month. Next, we filter and select 3,000 trips, each of which contains 20–30 road segments, to run our experiments.

We compare WatchDog with widely used baselines to evaluate the overall performance, which includes two categories: model selection-based baselines and non-model selection-based baselines.

Model selection-baesd baselines:

- **Adadeep** [35] This method leverages a DQN-based strategy to effectively select a combination of compression techniques that balance user-specified performance goals and resource constraints.
- **SkipRec-RL** [8] This work proposes an adaptive model hidden layer selection framework for deep sequential recommender system, which learns to skip inactive hidden layers on a per-user basis.
- **Greedy strategy** A greedy strategy of selecting the most fine-grained model first under the time-bound, based on some prior knowledge.

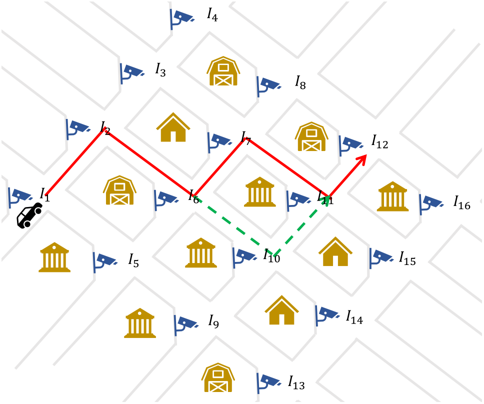


Fig. 8. The edge node at I_{11} is activated to track the VoI by multiple edge nodes deployed at neighboring intersections I_7 and I_{10}. Since the active period of an edge node is calculated based on the active period of its previous edge node, the edge node at I_{11} may have multiple active periods. If these periods overlap with each other, then the edge node's active period is a union of them.

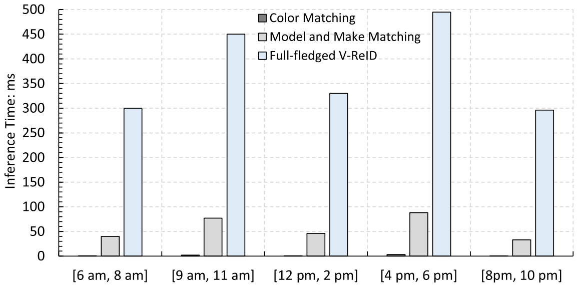


Fig. 9. The average inference time at different granularity levels.

Non-model Selection Based Baselines: On the other hand, in order to verify the effect of the adaptive model selection method, we compare our model with the traditional static fine-grained model approach. Here, we utilize the last fine-grained model mentioned in Section 4.3.3 as the baseline: Full-fledged V-ReID.

Implementation Details and Ground Truths: Our proposed methods and baselines are implemented using TensorFlow 1.14 and Python 3.6 on an edge server with AMD Ryzen 7 3700x (CPU) and one NVIDIA GeForce RTX 2080 Ti (GPU). We obtained the ground truth of the VoI's trajectories through the uploaded GPS data collected by onboard devices periodically.

8.1 Experimental Results

Real-Time Performance at Different Granularity Levels. In this subsection, we evaluate the reliance of the proposed object identification algorithms at different granularity levels (i.e., Color Matching, Model and Make Matching, and Full-fledged V-ReID method). The experimental results are summarized in Figure 9, where the X-axis denotes some typical time slots of a day, and Y-axis is the average inference time of the object identification algorithms. Based on the results, we found that the inference time of the Full-fledged V-ReID method is significantly longer than the other two algorithms' during a day, which follows the intuition in deriving the model selection-based D-V-ReID framework. In addition, the difference between them becomes larger during the rush hours (i.e., from 9 to 11 AM and from 4 to 6 PM). This is due to the fact that as more vehicles appear

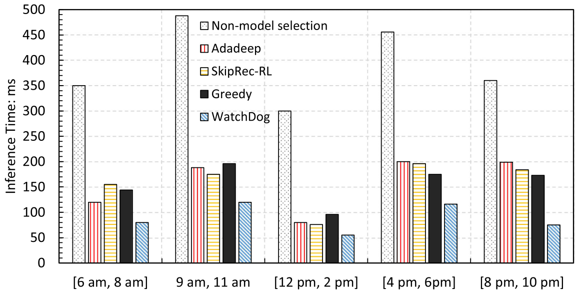


Fig. 10. The average inference time of different tracking methods.

at the intersections per unit time, the Full-fledged V-ReID method will introduce more real-time workload in the embedded computing system, and, hence, the contentions induced by the workload will exacerbate the algorithms' execution time. It suggested that during rush hours, WatchDog should start with the Color Matching module, especially at the busy intersections. Please note our evaluation is based on the real-world datasets. In our setting, a regular vehicle's Color and Make can be accurately identified. However, the plates may be blocked by some other vehicles. We have removed these imperfect data from our original datasets. In addition, we did not consider the night vision and occlusion problems in vehicle detection based on cameras due to a lack of real-world data.

Real-Time Performance of Different Tracking Methods. In this subsection, we evaluate the performance of our WatchDog solution and the Adadeep, SkipRec-RL, Greedy, and Non-model selection strategies during different time periods and traffic periods, and the results are illustrated in Figure 10. In this figure, the dark blue bar with diagonal stripes indicates the performance of our WatchDog solution, the gray bar with dotted diamond grid corresponds to the result of the non-model selection strategy, the red bar with vertical stripes gives the result of the Adadeep strategy, the orange bar with horizontal stripes represents the performance of the SkipRec-RL strategy, and the black bar indicates the performance of the Greedy strategy. And again, in Figure 10, rush periods include 9–11 AM and 4–6 PM, and normal periods are equal to the rest hours. From this figure, it is clear that our WatchDog solution outperforms other alternative approaches. Moreover, we discover that the performance of our solution increases a little bit with the increase of real-time traffic compared with the other methods. This is because with the increase of road traffic, the travel speeds of the VoI decrease, thus their information is easier to be captured by our system even if the number of vehicles showing at each intersection increases. Note that in Figure 10, the performance of the non-model selection method during 9–11 AM is obviously poor, the reason is that during this period the number of vehicles showing at each intersection achieves its peak value, most plates cannot be identified in real-time due to a lack of computing resources on-board, thus the information of VoIs is difficult to be obtained by the tracking system.

9 TRACE-DRIVEN EVALUATION

In addition to the empirical study, to evaluate the efficacy of WatchDog in a larger scale, we conducted extensive experiments based on our accessible real-world GPS datasets.

9.1 Data Description and Time/Traffic Measurement

Table 3 summarizes statistics about vehicle networks studied in this work. To test WatchDog in a real-world scenario, we utilize a real-world dataset of about six months of GPS traces of more than 14,000 vehicles in Shenzhen, a Chinese city with a 10 million population. The dataset is obtained

Table 3. Statistics of Vehicle Network

Dataset Summary	
Collection Period	6 Months
Collection Date	01/01/12–06/30/12
Number of Vehicles	14,453
Total Live Mile	371,269,642 miles

Table 4. A GPS Record

plate ID	**longitude**	**latitude**	**time**	**speed**
TIDXXXX	114.022901	22.532104	08:34:43	22 km/h

by letting every vehicle upload its GPS records (the format as in Table 4) to report its traces to a base station. Based on the dataset of GPS records, we obtain location and time distributions of the vehicles traveling in Shenzhen, which are used to evaluate the performance of WatchDog.

9.1.1 Measurement of Traveling Time. The GPS data are used to measure the traveling time of the VoI at specific locations in the monitored areas, including intersections and road segments. Figure 5(a) shows an example of traveling time taken by vehicles at an intersection. We can see that most of the vehicles take 5 s or 40 s to travel through this intersection. The reason is that if the traffic light at this intersection is red, the traveling time of a vehicle should include the waiting time. Due to different traveling speeds, the shortest time to travel through this intersection is 3 s and the longest time is 42 s according to Figure 5(a). Thus, we assume that the traveling time taken by the VoI at this intersection falls into this time interval, which is from 3 to 42 s. Similarly, according to Figure 5(b), the traveling time taken by the VoI at the road segment belongs to a time interval, which is from 30 to 50 s. By analyzing the GPS dataset, we can obtain the traveling time for the VoI at all intersections and road segments.

9.1.2 Measurement of Traffic Condition. Based on the time and location information in each GPS record, the datasets can be used to measure the traffic conditions in the monitored area. Figure 11 plots the percentage of the number of vehicles traveling through different intersections in one frame/1 min. At almost 16% of the intersections, only nine vehicles appear in 1 min. And for more than 95% of the intersections, less than 21 vehicles travel through these intersections in 1 min. These observations show that at most of the intersections, the traffic light and our proposed method can fully utilize the edge nodes deployed at the uncrowded intersections to track the VoI in real-time.

9.2 Real-time Performance

The key performance metric for WatchDog is the video analytics delay to track the VoI in real-time. We evaluate this metric every 1-h time window of a day. In addition, we investigate the sensitivities of WatchDog's performance on three key parameters, i.e., the tracking delay, the number of involved edged nodes, as well as the tracking cost.

In order to show the impacts of different traffic conditions on WatchDog, we evaluate the performance in three typical areas in Shenzhen: Residential area, Industrial area, and Commercial area, which are denoted by "Residential", "Industrial", and "Commercial" in Figure 12, respectively. For each tracking experiment, we randomly choose a running vehicle from the dataset as the VoI and measure the tracking delay and tracking cost according to its driving circumstance at different intersections on its trajectory according to the GPS records. The ReID modules are selected by each involved edge node automatically, based on the real-time admission control module and

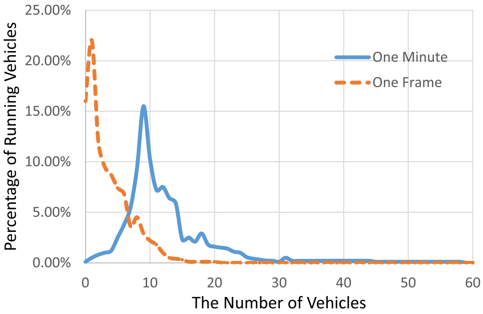
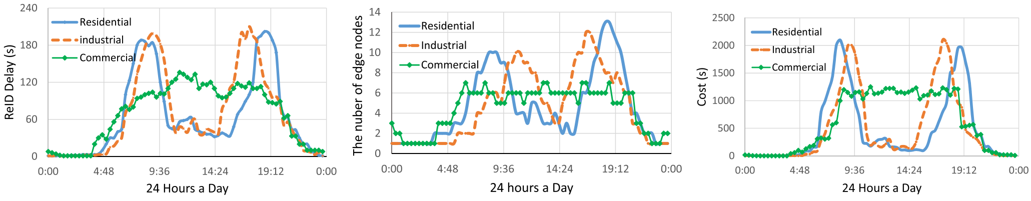


Fig. 11. The statistics of the number of vehicles traveling through an intersection in one minute/appearing on one video frame.



(a) Real-time Tracking Delay.

(b) The # of involved edge nodes.

(c) The total execution time to ReID a VoI.

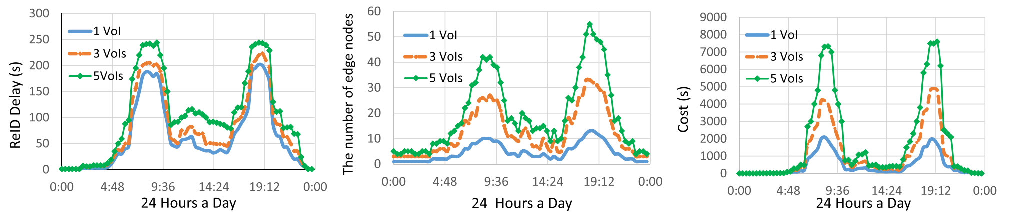
Fig. 12. The performance of Real-time tracking.

the number of vehicles at the intersection. The execution time for each ReID module is given in Table 2.

Figure 12(a) plots the average ReID delay during 24 h of a day. During the rush hour, e.g., 06 : 00–10 : 00, the ReID delay in three different areas reaches the maximal values. The reason is that the number of vehicles traveling through each intersection achieves the maximal value during the rush hour through the whole day. An interesting observation is that the peak of the Residential curve is earlier than the peak of the Industrial curve in the morning rush hour and on the contrary in the evening. This is because the traffic flows from the residential area to industrial area in the morning and reversely in the evening. As seen in this figure, even during the rush hour, the ReID delay is smaller than 4 min, confirming that WatchDog can track the VoI in real-time.

Figure 12(b) plots the average number of edge nodes involved in tracking a VoI during 24 h of a day. As seen in this figure, the maximum number of edge nodes involved in tracking a VoI is smaller than 14 even during the rush hour. This implies that through carefully selecting ReID modules for the edge nodes, WatchDog is able to efficiently reduce the number of suspected VoIs and limit the tracking range into a reasonably small area. Moreover, compared with the residential area and industrial area, the number of involved edge nodes to track a VoI in commercial area in the rush hour is much smaller. This is because the workplaces for residents are distributed in the industrial area, thus the traffic in the commercial area is not as heavy as that in residential/industrial areas during the rush hours.

We have also conducted a set of experiments evaluating the tracking cost in the three areas in terms of the total tracking time of all involved edge nodes to ReID a VoI 24 h a day. This metric can also be used to reflect the effectiveness of WatchDog since a shorter total tracking time to ReID and localize a VoI often results in a lower execution cost for the whole tracking system. Figure 12(c) shows the result using this metric. As seen, WatchDog can track the VoI with very few edge resources during non-rush hours, and the cost for real-time tracking during the rush hour is also reasonable.



(a) Multi-Object Real-time Tracking Delay. (b) The # of involved edge nodes. (c) The total execution time to ReID multiple VoIs.

Fig. 13. The performance of Multi-Object Real-time tracking.

9.3 Multi-Object Tracking

In this set of experiments, we evaluate the efficacy of WatchDog when handling the practical issue of multi-object real-time tracking (i.e., multiple hit-and-run accidents occur at the same time). Figure 13 shows the evaluation results. We randomly choose multiple running vehicles (i.e., 3 and 5) as the VoIs and track them simultaneously according to the GPS datasets. We use the metric "ReID delay" (in seconds) to reflect the tracking latency of WatchDog. Similarly, we use the metrics, "the number of involved edge nodes" and "Cost", to show the total execution time of the edge nodes consumed by WatchDog when tracking the multiple VoIs in real-time.

Figure 13 shows the results w.r.t. the multi-object real-time tracking. As seen in the figure, with increased number of VoIs, all the measured parameters increase during all time intervals. This observation confirms the intuition that as more VoIs are involved in real-time tracking, a larger amount of edge resources is needed to track all the VoIs simultaneously. One interesting observation is that the amount of increased edge resources is not proportional to the number of increased VoIs. For example, in Figure 13(c), the cost for tracking 1 VoI achieves its maximum value at around 8:00 AM, which is about 2,000 s. However, the cost for tracking 5 VoIs at 8:00 AM is less than 8,000 s. This implies that the intersections involved in tracking the 5 VoIs have overlaps with each other, and the video frame processing results are reused to track different VoIs. Another observation in Figure 13(a) is that the ReID Delays for tracking different number of VoIs are very close. The reason is that our proposed WatchDog is a distributed real-time tracking system, which can perform the multi-object tacking simultaneously.

10 CONCLUSION

Recent technology advances in edge computing provide new opportunities to implement a real-time tracking system in smart cities with edge nodes distributed at the intersections of the road network, which consists of both surveillance cameras and embedded computing platforms. We propose a simple yet effective real-time system for tracking hit-and-run vehicles in smart cities, which employs machine learning tasks with different resource-accuracy tradeoffs, and schedule tracking tasks across distributed edge nodes based on the number of detected vehicles to maximize the execution time of tasks while ensuring a provable completion time-bound at each edge node. WatchDog is also designed to be capable of addressing multi-object tracking problems to track multiple VoIs simultaneously in real-time.

REFERENCES

[1] Azure Stack Edge. [online]. https://azure.microsoft.com/en-us/products/azure-stack/edge/#overview.

[2] Geiger Andreas, Lenz Philip, and Urtasun Raquel. 2012. Are we ready for autonomous driving? The KITTI vision benchmark suite. In *Proceedings of the 2012 IEEE Conference on Computer Vision and Pattern Recognition*. IEEE.

[3] Erhan Bas, A. Murat Tekalp, and F. Sibel Salman. 2007. Automatic vehicle counting from video for traffic flow analysis. In *Proceedings of the 2007 IEEE Intelligent Vehicles Symposium*. IEEE, 392–397.

[4] Karsten Behrendt. 2019. Boxy vehicle detection in large images. In *Proceedings of the IEEE/CVF International Conference on Computer Vision Workshops.*

[5] He Bing, Li Jia, Zhao Yifan, and Tian Yonghong. 2019. Part-regularized near-duplicate vehicle re-identification. In *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition.*

[6] Michael Bramberger, Josef Brunner, Bernhard Rinner, and Helmut Schwabach. 2004. Real-time video analysis on an embedded smart camera for traffic surveillance. In *Proceedings of the RTAS 2004 10th IEEE Real-Time and Embedded Technology and Applications Symposium.*

[7] Xianbin Cao, Changxia Wu, Jinhe Lan, Pingkun Yan, and Xuelong Li. 2011. Vehicle detection and motion analysis in low-altitude airborne video under urban environment. *IEEE Transactions on Circuits and Systems for Video Technology* 21, 10 (2011), 1522–1533.

[8] Lei Chen, Fajie Yuan, Jiaxi Yang, Xiang Ao, Chengming Li, and Min Yang. 2021. A user-adaptive layer selection framework for very deep sequential recommender models. In *Proceedings of the AAAI Conference on Artificial Intelligence.* Vol. 35. 3984–3991.

[9] Opitz David and Maclin Richard. 1999. Popular ensemble methods: An empirical study. *Journal of Artificial Intelligence Research* 11 (1999), 169–198.

[10] Zheng Dong, Linghe Kong, Peng Cheng, Liang He, Yu Gu, Lu Fang, Ting Zhu, and Cong Liu. 2014. REPC: Reliable and efficient participatory computing for mobile devices. In *Proceedings of the 2014 11th Annual IEEE International Conference on Sensing, Communication, and Networking*. IEEE, 257–265.

[11] Zheng Dong, Cong Liu, Soroush Bateni, Zelun Kong, Liang He, Lingming Zhang, Ravi Prakash, and Yuqun Zhang. 2018. A general analysis framework for soft real-time tasks. *IEEE Transactions on Parallel and Distributed Systems* 30, 6 (2018), 1222–1237.

[12] D. S. Guru, Y. H. Sharath, and S. Manjunath. 2010. Texture features and KNN in classification of flower images. *International Journal of Computer Applications*, Special Issue on RTIPPR. 1 (2010), 21–29.

[13] Tafazzoli Faezeh, Frigui Hichem, and Nishiyama Keishin. 2017. A large and diverse dataset for improved vehicle make and model recognition. In *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition Workshops.*

[14] Rajamanoharan Georgia, KanacıAytac, Li Minxian, and Gong Shaogang. 2019. Multi-task mutual learning for vehicle re-identification. In *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition.*

[15] Ross Girshick. Fast R-CNN. 2015. In *Proceedings of the IEEE International Conference on Computer Vision.*

[16] Rafael C. Gonzales and Richard E. Woods. 2002. Digital Image Processing. Pearson Education India.

[17] Hector Gonzalez, Jiawei Han, Xiaolei Li, Margaret Myslinska, and John Paul Sondag. 2007. Adaptive fastest path computation on a road network: A traffic mining approach. In *Proceedings of the 33rd International Conference on Very Large Data Bases.* ACM, Inc, 794–805.

[18] Wang Guanshuo, Yuan Yufeng, Chen Xiong, Li Jiwei, and Zhou Xi. 2018. Learning discriminative features with multiple granularities for person re-identification. In *Proceedings of the 26th ACM International Conference on Multimedia.*

[19] Kaiming He, Xiangyu Zhang, Shaoqing Ren, and Jian Sun. 2016. Deep residual learning for image recognition. In *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition.*

[20] Bay Herbert, Ess Andreas, Tuytelaars Tinne, and Van Gool Luc. 2008. Speeded-up robust features (SURF). In *Proceedings of the European Conference on Computer Vision.*

[21] Chien-Chun Hung, Ganesh Ananthanarayanan, Peter Bodík, Leana Golubchik, Minlan Yu, Victor Bahl, and Matthai Philipose. 2018. VideoEdge: Processing camera streams using hierarchical clusters. In *Proceedings of the 2018 IEEE/ACM Symposium on Edge Computing.*

[22] Hsu Hung-Min, Huang Tsung-Wei, Wang Gaoang, Cai Jiarui, Lei Zhichao, and Hwang Jenq-Neng. 2019. Multi-camera tracking of vehicles based on deep features re-ID and trajectory-based camera link models. In *Proceedings of the CVPR Workshops.*

[23] Yusuke Ishii. 2005. Monitor system for monitoring suspicious object. (Dec. 15 2005). US Patent App. 11/150,264.

[24] Spanhel Jakub, Bartl Vojtech, and Herout Adam. 2019. Vehicle re-identifiation and multi-camera tracking in challenging city-scale environment. In *Proceedings of the CVPR Workshops 2019.*

[25] Junchen Jiang, Yuhao Zhou, Ganesh Ananthanarayanan, Yuanchao Shu, and Andrew A. Chien. 2019. Networked cameras are the new big data clusters. In *Proceedings of the 2019 Workshop on Hot Topics in Video Analytics and Intelligent Edges.*

[26] Hu Jie, Shen Li, and Sun Gang. 2018. Squeeze-and-excitation networks. In *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition.*

[27] Dai Jifeng, Li Yi, He Kaiming, and Sun Jian. 2016. R-FCN: Object detection via region-based fully convolutional networks. In *Proceedings of the Advances in Neural Information Processing Systems.*

[28] So Yeon Jo, Namhyun Ahn, Yunsoo Lee, and Suk-Ju Kang. 2018. Transfer learning-based vehicle classification. In *Proceedings of the 2018 International SoC Design Conference.*

[29] Redmon Joseph and Farhadi Ali. 2018. YOLOv3: An incremental improvement. arXiv:1804.02767. Retrieved from https://arxiv.org/abs/1804.02767.

[30] Redmon Joseph, Divvala Santosh, Girshick Ross, and Farhadi Ali. 2016. You only look once: Unified, real-time object detection. In *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition.*

[31] He Kaiming, Georgia, and Dollár Piotr. 2017. Mask R-CNN. In *Proceedings of the IEEE International Conference on Computer Vision.*

[32] Guillaume Leduc. 2008. Road traffic data: Collection methods and applications. *Working Papers on Energy, Transport and Climate Change* 1, 55 (2008), 1–55.

[33] Hennadiy Leontyev and James H. Anderson. 2007. Tardiness bounds for FIFO scheduling on multiprocessors. In *Proceedings of the 19th Euromicro Conference on Real-Time Systems.*

[34] Honghai Liu, Shengyong Chen, and Naoyuki Kubota. 2013. Intelligent video systems and analytics: A survey. *IEEE Transactions on Industrial Informatics* 9, 3 (2013), 1222–1233.

[35] Sicong Liu, Yingyan Lin, Zimu Zhou, Kaiming Nan, Hui Liu, and Junzhao Du. 2018. On-demand deep model compression for mobile devices: A usage-driven model selection framework. In *Proceedings of the 16th Annual International Conference on Mobile Systems, Applications, and Services.* 389–400.

[36] Yin Lou, Chengyang Zhang, Yu Zheng, Xing Xie, Wei Wang, and Yan Huang. 2009. Map-matching for low-sampling-rate GPS trajectories. In *Proceedings of the 17th ACM SIGSPATIAL International Conference on Advances in Geographic Information Systems.* 352–361.

[37] David G. Lowe. Object recognition from local scale-invariant features. 1999. In *Proceedings of the 7th IEEE International Conference on Computer Vision.*

[38] Rongxing Lu, Xiaodong Lin, Haojin Zhu, and Xuemin Shen. 2009. SPARK: A new VANET-based smart parking scheme for large parking lots. In *Proceedings of the IEEE INFOCOM 2009.* IEEE, 1413–1421.

[39] Cordts Marius, Omran Mohamed, Ramos Sebastian, Rehfeld Timo, Enzweiler Markus, Benenson Rodrigo, Franke Uwe, Roth Stefan, and Schiele Bernt. 2016. Feature pyramid networks for object detection. In *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition.*

[40] Dalal Navneet and Triggs Bill. 2005. Histograms of oriented gradients for human detection. In *Proceedings of the 2005 IEEE Computer Society Conference on Computer Vision and Pattern Recognition.*

[41] Boonsim Noppakun and Prakoonwit Simant. 2017. Car make and model recognition under limited lighting conditions at night. *Pattern Analysis and Applications* 20, 4 (2017), 1195–1207.

[42] Khorramshahi Pirazh, Kumar Amit, Peri Neehar, Sai Saketh Rambhatla, Chen Jun-Cheng, and Chellappa Rama. 2019. A dual-path model with adaptive attention for vehicle re-identification. In *Proceedings of the IEEE/CVF International Conference on Computer Vision.*

[43] Khorramshahi Pirazh, Peri Neehar, Kumar Amit, Shah Anshul, and Chellappa Rama. 2019. Attention driven vehicle re-identification and unsupervised anomaly detection for traffic understanding. In *Proceedings of the CVPR.*

[44] Baran Remigiusz, Glowacz Andrzej, and Matiolanski Andrzej. 2015. The efficient real- and non-real-time make and model recognition of cars. *Multimedia Tools and Applications* 74, 12 (2015), 4269–4288.

[45] Mark Sandler, Andrew Howard, Menglong Zhu, Andrey Zhmoginov, and Liang-Chieh Chen. 2018. MobileNetV2: Inverted residuals and linear bottlenecks. In *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition.*

[46] E. Schmitt and Hossein Jula. 2006. Vehicle route guidance systems: Classification and comparison. In *Proceedings of the 2006 IEEE Intelligent Transportation Systems Conference.* IEEE, 242–247.

[47] Ren Shaoqing, He Kaiming, Girshick Ross, and Sun Jian. 2015. Faster R-CNN: Towards real-time object detection with region proposal networks. In *Proceedings of the Advances in Neural Information Processing Systems.*

[48] Kazufumi Suzuki and Hideki Nakamura. 2006. TrafficAnalyzer-the integrated video image processing system for traffic flow analysis. In *Proceedings of the 13th ITS World Congress.*

[49] Mingxing Tan and Quoc V. Le. 2019. EfficientNet: Rethinking model scaling for convolutional neural networks. In *Proceedings of the International Conference on Machine Learning.*

[50] Huang Tsung-Wei, Cai Jiarui, Yang Hao, Hsu Hung-Min, and Hwang Jenq-Neng. 2019. Multi-view vehicle re-identification using temporal attention model and metadata re-ranking. In *Proceedings of the CVPR Workshops.*

[51] Lin Tsung-Yi, Goyal Priya, Girshick Ross, He Kaiming, and Dollár Piotr. 2017. Focal loss for dense object detection. In *Proceedings of the IEEE International Conference on Computer Vision.*

[52] V. F. Rodriguez-Galiano, B. Ghimire, J. Rogan, M. Chica-Olmo, and J. P. Rigol-Sanchez. 2012. An assessment of the effectiveness of a random forest classifier for land-cover classification. *ISPRS Journal of Photogrammetry and Remote Sensing* 67 (2012), 93–104.

[53] Fei-Yue Wang. 2010. Parallel control and management for intelligent transportation systems: Concepts, architectures, and applications. *IEEE Transactions on Intelligent Transportation Systems* 11, 3 (2010), 630–638.

[54] Yang Wang, Wuji Chen, Wei Zheng, He Huang, Wen Zhang, and Hengchang Liu. 2017. Tracking hit-and-run vehicle with sparse video surveillance cameras and mobile taxicabs. In *Proceedings of the 2017 IEEE International Conference on Data Mining*. IEEE, 495–504.

[55] Liu Wei, Anguelov Dragomir, Erhan Dumitru, Szegedy Christian, Reed Scott, Fu Cheng-Yang, and C. Berg Alexander. 2016. SSD: Single shot multibox detector. In *Proceedings of the European Conference on Computer Vision*. Springer.

[56] Chen Weihua, Chen Xiaotang, Zhang Jianguo, and Huang Kaiqi. 2017. A Multi-task deep network for person re-identification. In *Proceedings of the AAAI Conference on Artificial Intelligence*.

[57] Tan Xiao, Wang Zhigang, Jiang Minyue, Yang Xipeng, Wang Jian, Gao Yuan, Su Xiangbo, Ye Xiaoqing, Yuan Yuchen, He Dongliang, Wen Shilei, and Ding Errui. 2019. Multi-camera vehicle tracking and re-identification based on visual and spatial-temporal features. In *Proceedings of the CVPR Workshops*.

[58] Zhang Yu, Wei Ying, and Yang Qiang. 2018. In *Proceedings of the Advances in Neural Information Processing Systems*.

[59] Junping Zhang, Fei-Yue Wang, Kunfeng Wang, Wei-Hua Lin, Xin Xu, and Cheng Chen. 2011. Data-driven intelligent transportation systems: A survey. *IEEE Transactions on Intelligent Transportation Systems* 12, 4 (2011), 1624–1639.

[60] Xiangyu Zhang, Xinyu Zhou, and Jian Sun. 2018. ShuffleNet: An extremely efficient convolutional neural network for mobile devices. In *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*.

[61] Wang Zhongdao, Tang Luming, Liu Xihui, Yao Zhuliang, Yi Shuai, and Jing. 2017. Orientation invariant feature embedding and spatial temporal regularization for vehicle reidentification. In *Proceedings of the IEEE International Conference on Computer Vision*.

Received 15 October 2021; revised 9 May 2022; accepted 7 July 2022