CityTrac: Precise Camera Selection and Movement Prediction for Object Tracking in Hyperscale Public Security Camera Network

Jiaping Yu, Hongjia Wu, Tongqing Zhou, Zhiping Cai, *Member, IEEE*, Wenyuan Kuang, and Hui Xia

***Abstract*—Using hyperscale surveillance cameras, seamless target tracking can be accomplished in urban security scenarios, significantly enhancing public security and emergency response capabilities. In spite of the advantage of edge computing, tracking multiple targets using multiple cameras would incur prohibitive high computation costs. Based on the deployment of real-world cameras, this research finds that existing tracking scheduling is inefficient as a result of redundant and excessive activation of cameras. As a follow-up, the research proposes a hierarchical tracking framework called CityTrac that leverages fine-grained target movement predictions to provide efficient tracking in hyperscale cameras. First, CityTrac uses a specially designed camera selection strategy that ensures accurate tracking with a minimum number of cameras. After that, CityTrac constructs a probabilistic target movement graph by using historical tempo-spatial correlation information. Using the graph as a model, the tracking scheduling and camera selection problem are formulated as an optimization problem with efficiency-accuracy trade-off constraints. The research addresses this NP-hard problem using greedy optimization. The experiments conducted with the Cityflow and Geolife datasets demonstrate that, compared with two baselines, CityTrac requires significantly fewer computation resources (over 90%) in order to track the same number of targets with the same level of accuracy.**

***Index Terms*—Internet of Things, multicamera multitarget tracking, sensor network scheduling.**

Received 6 November 2024; revised 3 January 2025; accepted 18 January 2025. Date of publication 31 January 2025; date of current version 26 March 2025. This work was supported in part by the National Key Research and Development Program of China under Grant 2024YFB3311802; in part by the National Natural Science Foundation of China (NSFC) under Grant 62172377, Grant 62172155, and Grant 62472434; in part by the Taishan Scholars Program of Shandong Province under Grant tsqn202312102; and in part by the Shandong Provincial Natural Science Foundation under Grant ZR2024QF055. The work of Hui Xia was supported by the Natural Science Foundation of China (NSFC) under Grant 62172377 and Grant 61872205. This article was presented in part at the 2023 IEEE International Conference on Acoustics, Speech and Signal Processing (ICASSP),Rhodes Island, Greece, 2023 [DOI: 10.1109/ICASSP49357.2023]. *(Corresponding authors: Zhiping Cai; Hui Xia.)*

Jiaping Yu and Hui Xia are with the School of Computer Science and Technology, Ocean University of China, Qingdao 266005, China (e-mail: xiahui@ouc.edu.cn).

Hongjia Wu is with the Department of Mathematics and Information Technology, The Education University of Hong Kong, Hong Kong, SAR, China.

Tongqing Zhou and Zhiping Cai are with the College of Computer, National University of Defense Technology, Changsha 410073, Hunan, China (e-mail: zpcai@nudt.edu.cn).

Wenyuan Kuang is with the Cyber Security Innovation Center, 360 Digital Security Group, Beijing 100085, China.

Digital Object Identifier 10.1109/JIOT.2025.3532965

I. Introduction

HYPERSCALE surveillance camera networks play a significant role in enhancing the safety and intelligence of modern cities [1], [2], [3], [4], [5]. There have been reports of thousands of surveillance devices being deployed in megacities, such as New York and Singapore [6]. Applications that require tracking objects in live video streams from distributed cameras are based on the analysis of live video streams from these cameras [7], [8], [9], [10], [11], [12]. For instance, the BriefCam application [13] utilizes surveillance videos for effective crowd management. Although surveillance cameras have been widely deployed, video analysis services have not been provided on a large scale [14], [15], [16], [17], [18], [19]. The primary challenge in providing these services is that they must provide analytical results in real-time, as well as low latency, high throughput, and scalability [20], [21], [22], [23]. Notably, in many Internet of Things (IoT) deployments, large numbers of sensors and devices (such as cameras) continuously generate data at the edge, demanding efficient data transmission and processing to achieve near real-time analytics. Managing these hyperscale sensor networks is crucial for modern smart cities, highlighting the urgency of developing cost-effective yet reliable video analysis frameworks.

Analysis of video generated by hyperscale surveillance cameras is not trivial. As an example, 100 distributed cameras with a resolution of 1080P are capable of generating hundreds of gigabytes of data a minute [24]. The transmission of these data to the remote cloud will put tremendous strain on the networks [25]. As a result, edge computing [26] is typically used via edge servers to perform video analysis computations. A number of edge devices have so far been equipped with DL computing hardware for learning tasks on surveillance videos [27] (e.g., Azure Stack Edge Pro R [28]).

Tracking multiple targets with onboard DL accelerators still poses prohibitively high computation costs for edge servers [29], [30], [31], [32]. Therefore, tracking targets efficiently has become an increasingly important concern [33], [34], [35]. In theory, turning off fewer edge servers and cameras for tracking a target could reduce costs, but could compromise tracking accuracy [36], [37], [38]. Several tracking scheduling methods have recently been proposed that activate only cameras facing in the matched directions or that are in close proximity to the moving targets [39], [40], [41]. Such strategies reflect a key challenge in IoT-driven systems, where managing resource constraints—like bandwidth, battery

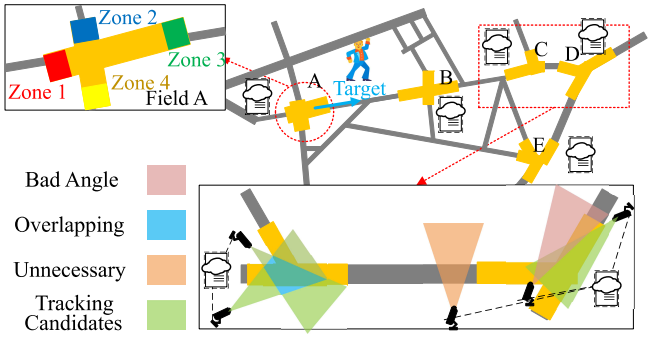

Fig. 1. Toy example on the tracking-inefficient deployment of surveillance cameras under distributed edge servers. Some camera activation is redundant, overused, and unnecessary in existing tracking solutions.

power, or compute capacity—must be balanced against the need for accurate and timely data processing.

This work observes with a real-world surveillance video dataset (CityFlow [24]) that existing scheduling strategies remain cumbersome and provide limited efficiency when facing hyperscale camera deployment. Three essential factors are elaborated for their inefficiency.

1) *Overlapping Surveillance Regions:* There is a tendency in most existing smart tracking strategies to activate all cameras that the target may pass through. Our evaluation demonstrates that skipping some of the cameras will not impact tracking accuracy.
2) *Unnecessarily Involved Cameras:* It is common for smart tracking strategies to simulate only one camera covering an area of interest. Real-world city-scale scenarios, however, would likely require multiple cameras to capture a particular region of interest to minimize the impact of camera location and angle.
3) *Awkward Locations and Angles:* Poor tracking accuracy results from using cameras at awkward angles and locations indiscriminately, resulting in the use of more cameras to achieve acceptable tracking.

Based on current knowledge, the aforementioned facts have not been investigated. Inspired by these findings, CityTrac is proposed as a solution for tracking targets within city-scale surveillance camera networks. As shown in Fig. 1, utilizing hierarchical geographical parsing, it divides an edge server's administration region into multiple *Fields* of Interest, each representing an area covered by a cluster of nearby cameras. Each *Field* is further subdivided into critical *Zones*, denoting entry or exit regions. With this structure, CityTrac calculates the probability of targets' movements as a probabilistic movement graph, encoding tempo-spatial correlations as the number and frequency of trips between *Zones*. Following this, tracking costs are quantified based on movement predictions and integrated with tracking accuracy into a tradeoff inequality. These probabilistic correlations serve as constraints for the efficient scheduling goal, which is ultimately reduced to the linear programming of optimal edges and Field selections. Consequently, our framework also underscores the broader IoT perspective: camera networks are among the largest data generators in urban environments, and CityTrac's design aims to intelligently select and schedule these IoT endpoints in a cost-effective, scalable manner.

Simulations have been conducted on both small-scale (7 *Fields*) and large-scale (103 *Fields*) camera networks based on real-world datasets CityFlow [24] and Geolife [42], [43], [44]. Experimental results indicate that CityTrac could significantly reduce the computation burden (reducing at least 90% of the computation data) while maintaining intact tracking accuracy as compared with Spatula and Anveshak. This demonstrates the high feasibility of CityTrac in practical deployment.

Contribution: We claim the following contributions.

1) We analyze the deployment of surveillance cameras in a real-world scenario and summarize the factors that may cause unnecessary cost raising.
2) To perform the multitarget-multicamera (MTMC) Tracking on a hyperscale scenario with resource-limited edges, we propose CityTrac, an energy-efficient solution for hyperscale multicamera object tracking. It dramatically reduces the video analysis system's workload through target route approximation and optimized spatial-temporal correlation.
3) Using historical accuracy data, we propose a precise camera selection strategy in order to reduce the impact of camera-related factors on tracking accuracy.
4) We illustrate the effectiveness and efficiency of CityTrac by comparing it with tracking logic from two state-of-the-art MTMC tracking systems: Khochare et al. [41] and Jain et al. [40]. The evaluation result shows that CityTrac requires only 10% of surveillance data to achieve the same accuracy as the other two state-of-the-art surveillance systems.

This study expands upon and substantially extends our preliminary findings in [45], the 2023 IEEE International Conference on Acoustics, Speech and Signal Processing (ICASSP). In that paper, we modeled the MTMC scheduling problem as a linear programming task without detailing how cameras should be selected or emphasizing scenarios with different target importance. In contrast, **CityTrac** introduces a novel camera selection strategy (detailed in Section V-A6), significantly enhancing our prior approach. We also provide more comprehensive experiments across two datasets, illustrating the inefficiencies of existing scheduling strategies and validating our new solution at a larger scale.

The remainder of this article is organized as follows. Section II covers the related work, providing an overview of existing video analysis and tracking strategies. Section III presents the preliminary concepts and methodologies underlying our research. Section IV analyzes the inefficiencies in existing hyperscale tracking systems and the prohibitive costs associated with them. Section V introduces the detailed design of CityTrac. Section VI describes the experimental setup and the evaluation results, comparing CityTrac's performance with baseline methods, and demonstrating its efficiency and effectiveness in reducing computation workload and failure rate. Section VII discusses the limitations of our approach and potential future improvements. Section VIII concludes this article, summarizing the key contributions and findings of our research.

II. Related Work

A. Video Analysis Applications

Numerous studies have been published on video analysis applications, most of which require data from cameras at different geographical locations. This section discusses several typical applications.

As part of its work, Liu et al. [46] proposed the use of a near-real-time detection mechanism for complex activity. It allows users to specify complex actions. The system was capable of converting users' specifications into graphs. Using camera information, it was possible to match specification graphs with the underlying data in order to detect activities. The evaluation results demonstrate that this system achieves high precision and recall for all complex activities in the dataset with limited wireless bandwidth and low latency. The server side of Caesar, however, requires three Nvidia RTX 2080 GPUs, which have a Thermal Design Power of 260 watts, making it impractical for use in city-scale surveillance systems. He et al. [47] solved the problem of cross-camera tracklet matching. It generates complete global trajectories for all targets by combining trajectories from all cameras. In real-world deployments, however, it is not realistic to share a tracklet among all the cameras on the network due to limitations in bandwidth and resource availability. For autonomous driving applications, Du et al. [9] proposed a multicamera environment representation fusion solution. Due to the fact that the algorithm is installed in the vehicle, it is difficult to allocate adequate resources to each camera. As a result, resource limitations also affect the performance of such applications. Paul et al. [48] focused on improving the accuracy of video analytics through real-time automated tuning of camera parameters. The proposed system dynamically adjusts settings based on the environment and task requirements, enhancing the efficiency of surveillance systems in urban areas. Liu et al. [49] presented Vi-Fi, a system that associates moving subjects across vision and wireless sensors to improve tracking accuracy. You et al. [50] proposes an online optical-based Pose Association (OPA) method for multitarget multicamera tracking, which addresses occlusion and fast motion issues by utilizing local pose matching and optical flow adjustments to enhance tracking accuracy. The integration of multiple sensor modalities enhances the robustness of tracking systems in complex urban environments.

B. Resource Allocation Solutions for Video Analysis Systems

It has been a long-standing study to determine how to optimize the resources of video analytics systems. There have been proposals for resource allocation solutions for video analysis systems in order to optimize their computing capabilities. However, most of these systems were designed for small-scale scenarios, and they do not take into account the resource allocation problems that sometimes arise with large-scale camera networks uniquely.

A method of fusing information from multiple sensors and estimating the current state of a tracker was proposed by Han et al. [51] in 2010. It uses a combination of sequential Bayesian filters (such as a target filter) for each sensor, with each filter contributing a different level to the estimation of the combined posterior by the model. As part of this framework, multiple sensors interact to determine an appropriate sensor for each target dynamically; each target is assigned to only one sensor for measurement, and the number of targets assigned to each sensor varies accordingly.

To achieve low latency, high throughput, and scalable live video analytics, Zeng et al. [34] proposed a method for adaptively balancing the workload between cameras and edge clusters. The Distream solution takes advantage of the camera's onboard resources more efficiently, but may not be adequate in the hyperscale scenario where some tasks may require more resources than the camera can provide. Lee et al. [52] proposed a human tracking solution based on a camera topology that significantly reduces the number of resources consumed. Similarly, it was designed for campus and does not seem applicable to a larger-scale scenario. Liu et al. [53] presented a multitask learning approach for real-time tracking of smartwatch orientation and location. The method improves the accuracy and efficiency of tracking wearable devices, which can be applied to broader surveillance systems for better integration of personal devices in urban monitoring networks. Yang et al. [54] explored the optimization of placement for UAV-enabled wireless networks in urban environments. The multihop backhaul approach improves the coverage and efficiency of surveillance networks, making it a valuable reference for optimizing large-scale urban surveillance systems. Lin et al. [29] proposed a novel resource allocation solution for video analysis systems by formulating CTU-level bit allocation as a Nash equilibrium problem within the λ-domain rate control, enhancing the efficiency and visual quality of Versatile Video Coding (VVC).

III. Preliminary

A. Decentralized Object Tracking and Video Analyzing System

Instead of connecting all the surveillance cameras to a centralized server, a decentralized MTMC system consists of several edge servers, and each server is connected to a set of surveillance cameras located at specific locations. P Please note that although some smart cameras, like Amazon DeepLens, are equipped with limited neurocomputing capacities, for ease of analysis, it is assumed in this article that all surveillance cameras can only decide whether to upload the surveillance data to the edge server for further analysis based on predefined commands. This decision-making process is called filter control in [41]. Cameras with neurocomputing capacities can be generalized as one camera connected with one edge server.

The general workflow of the MTMC system is shown in Fig. 2. When receiving the surveillance data from the camera, the edge server then performs the object tracking or other video analyzing tasks based on the users' requirements. In this step, the edge server generates a batch of key-value pairs for further processing. Typically, the key is the surveillance camera's ID, and the value is the bounding box of the potential targets in several frames.

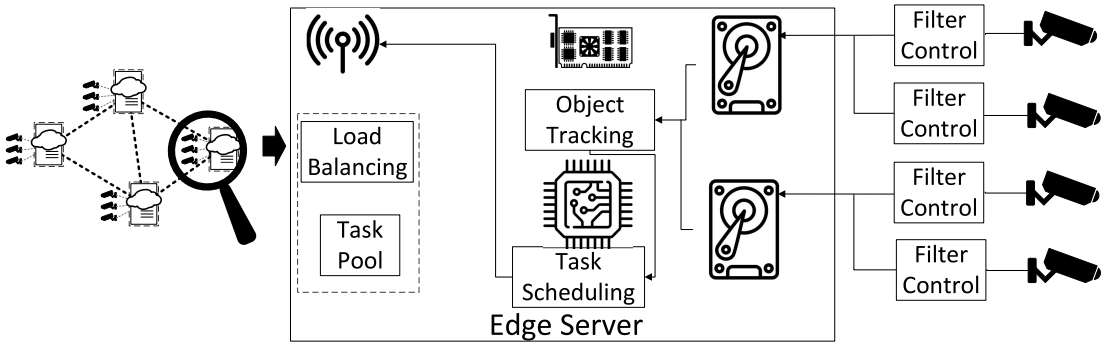

Fig. 2. Genreal Model of an MTMC System.

After generating the analyzing results of the surveillance data, the edge server then shares the results with other edge servers and issues the tracking schedule in the next tracking round. Suppose it receives tracking requirements from other servers before. In that case, the edge server will compare the results with the targets' information and shares the comparison result with the server which sends the tracking data.

Occasionally, the MTMC system may have conflict detection results: (e.g., cameras deployed in different locations detects the same target simultaneously). MTMC system needs to provide a contention resolution mechanism to resolve the conflict detection. Usually, the detection results come with a confidence score to show how likely the tracking result is correct.

Besides the mechanism mentioned above, the MTMC system also needs to decide when to activate or deactivate the cameras to balance the tracking accuracy and resource consumption.

Theoretically, once the targets leave the camera's field of view, the edge server needs to activate all the surrounding cameras to assure the highest tracking accuracy. But in practice, this strategy can cause a huge burden to the edge server.

As an example of a naive approach, we illustrate in Fig. 3 how activating all potentially relevant cameras upon a target's departure can rapidly escalate the resource overhead. This approach highlights the pitfalls of turning on every neighboring camera whenever the target leaves one camera's field of view. Although such a method might ensure coverage, it is neither resource-efficient nor scalable in real-world deployments.

In Fig. 3, assuming that each edge server connects with one camera. Starting from the left side, once the walking target leaves the field of view in edge A, edge A needs to activate the edges B, F, G, and A (since the target may come back) to cover all the possible places the target is likely to go. Meanwhile, there is another vehicle target appeared in the field of view of edge M, then the edge server needs to activate the cameras connected with G, L, and M for a certain period. Then, the camera connected with B and L discovers the walking and vehicle targets simultaneously, then edge server C, F, B, H, K, and L needs to activate their connected cameras and analyze the surveillance data. Once the walking target reaches F and the vehicle target reaches H, then edges A, B, C, E, F, G, H, I, L, and M need to be activated. Edge F and H need to track both targets simultaneously, which could be a huge burden to their processing power.

In contrast, our subsequent approach leverages a probabilistic target movement graph (Section V-A2), which uses historical observations to estimate the likelihood of transitioning between different Zones in a Field. By doing so, we only activate the cameras and Fields with high transition probabilities, significantly reducing unnecessary overhead while maintaining robust tracking accuracy.

B. Tracking Logics From Existing MTMC System

Researchers propose different tracking logic to reduce the processing power the tracking task needs. Taking Khochare et al. [41] and Jain et al. [40] as two examples.

Khochare et al. [41] is a scalable platform for multicamera distributed object tracking. It implements a domain-specific dataflow model. With this model, Anveshak allows functional operators to plug in different function models and tracking strategies to decide which camera to activate dynamically. In [41], based on the Anveshak platform, authors propose an example tracking logic that allows users to tune the accuracy, latency, and scalability by dropping and batching frames.

Jain et al. [40] is designed as a cross-camera analytic system. It leverages the spatio-temporal correlations among different cameras so that the system can evaluate which frames are most likely to contain the query identity. This makes the system can dramatically reduce the network and compute costs.

When tracking an object, Anveshak only activates the surveillance cameras in the direction where the target is last seen. And expand the activation region based on domain knowledge like the road, the speed of the target, or the time the target leaves the field of view. Spatula generates a camera-based spatial-temporal correlation table based on the domain knowledge from previous data. When the target leaves the field of view, it activates the most likely camera during the most likely time period that the target reaches.

IV. Analysis on Hyperscale Tracking

A. Prohibitive Costs

First, a test is conducted to observe the computation costs of distributed edges on tracking objects. The simulation uses an edge server with a workstation equipped with an Intel i5 CPU,

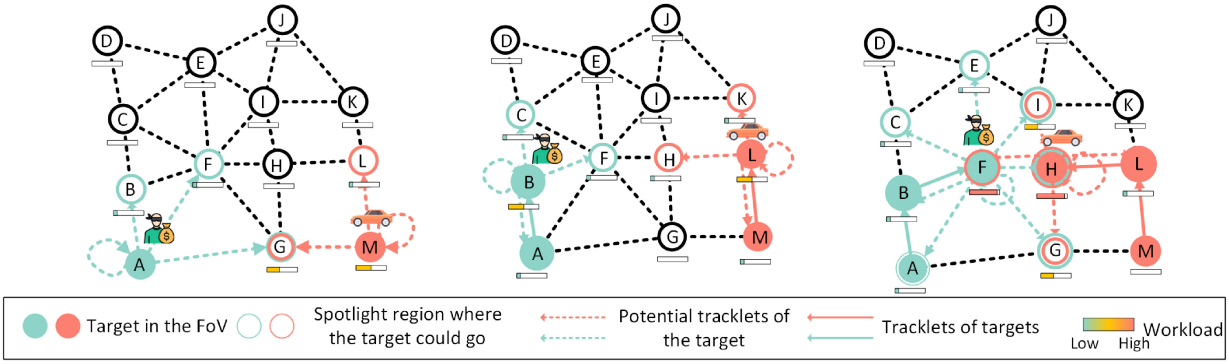

Fig. 3. Workflow of a naive tracking strategy. Involving all the surrounding edges in the tracking strategy may cause a huge burden on the edge resources.

Nvidia RTX 3060 GPU, and 32GB RAM. Surveillance videos are extracted from the Cityflow dataset [24], each containing 200 s of footage from six different cameras. It takes the simulated edge 1163 s to process with such an amount of data. For hyperscale cameras for tracking, the incurred computation costs would be prohibitively high if directly activating all of them for real-time tracking on multiple targets.

B. Observations on Inefficiency

The above scheduling solutions still provide limited efficiency. By examining city-scale camera deployments in practice, three aspects contributing to inefficiency are identified, as shown in Fig. 4. First, numerous cameras have *surveillance region overlapping*, making simultaneous activation of them for tracking redundant. Second, a camera between two cameras with no road detour cannot provide additional tracking tracklets, so is *unnecessary* to be activated. Third, some cameras have awkward *deployment locations and angles* which is useless in inferring targets' moving trajectories.

1) Overlapping: In a real-world deployment, a large number of cameras may have shooting range overlapping. In some cases, the shooting range of one certain camera could be fully covered by another. Fig. 4(a) could be a good example. The camera C21's field of view is fully covered by camera C20. Possible reasons are these cameras are not deployed simultaneously, these cameras are owned by a different institution, or users tend to deploy multiple cameras to assure the reliability of the system. To reduce the burden of the edge server, instead of analyzing all the overlapped videos, in this case, the edge server only needs to evaluate the data from C20. Data from C21 can be used to verify the tracking result when conflict occurs.

2) Unnecessary Deployment: When performing object tracking tasks, the tracing system needs to approximate the tracklets of the target based on the surveillance data. To do this, the tracking logic needs to know if the targets change their moving directions. On the other hand, if cameras cannot provide additional information about the targets, then these cameras are less critical. The main feature of this kind of camera is that although some cameras' shooting range does not overlap with other cameras, their set positions are covered by other cameras before and after.

Take the scenario in Fig. 4(b) as an example. When targets leave from the blue region in camera C24, then three possible cases may happen: 1) it keeps going and enters the crossroad covered by C22; 2) it turns back and enters the C24's field of view again; and 3) it stops between C24 and C22, which is likely to be in the field of view of C23. Previous tracking logic tends to analyze video from C23 to determine if case 3 happens.

However, though camera C23 is not overlapped with any cameras, all the vehicles entering the surveillance field of camera C23 from the upper side are passed from camera C22, and from the downside are passed from camera C24. And there is no sideroad between C24 and C22, the system can know case 3 happens if neither C24 nor C22 captures the target.

3) Bad Angle: In some cases, the camera's deployment location and angle make the camera unable to determine the accurate moving destination of targets. The object tracking system needs to take measures to avoid spending computing power to analyze these surveillance data.

For example, as shown in Fig. 4(c), in C33, the field of view is very easy to be hindered by other vehicles. Because of the bad angle, vehicles coming from S.W. University Avenue and E. University Avenue are likely to block the camera's field of view, and it cannot tell vehicles from E. University Avenue are leaving toward S.W. University Avenue or Ansbury Rd. Camera C35 has a much clearer field of view. Thus, it is much easier to detect the vehicle coming and leaving in the direction of camera C35 than with camera C33.

V. Design of CityTrac

A. Detail of CityTrac Scheduling Strategy

In response to the inefficiency in existing tracking systems, this work presents the design of a movement prediction-based tracking framework, named CityTrac, whose workflow is shown in the right part of Fig. 5. CityTrac builds on a three-tier structure, which contains *scheduling edge (server)*, *tracking edge (servers)*, and *cameras*. The workflow of CityTrac is as follows.

1) *Stage 1:* Based on the predicted tempo-spatial movement of the targets, the scheduling edge constructs the tracking schedule.

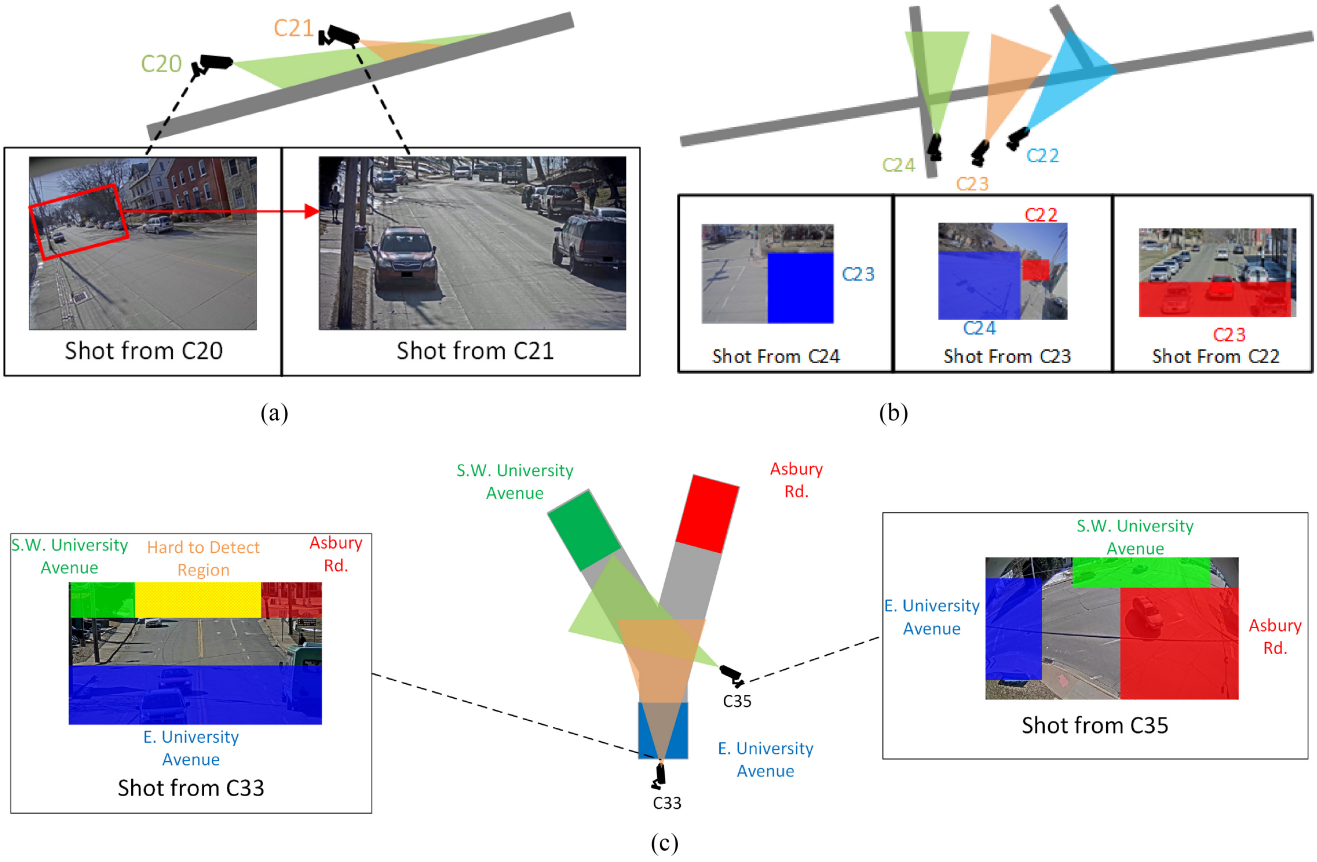

Fig. 4. Example of inefficiency in the real-world camera deployment. (a) *Example of Overlapping:* C21's field of view is fully covered by C20. (b) *Example of Unnecessary Deployment:* Whether analyzing the surveillance data from C23 has no significant impact on the target tracking result. (c) *Example of Bad Angle:* Hard to determine the targets' leaving destination in C33.

2) *Stage 2:* The scheduling edge distributes the tracking schedule to a specific tracking edge group.
3) *Stage 3:* Based on the targets' appearance probabilities in the cameras' monitored Zones, the selected tracking edges assign target tracking tasks to some of their local cameras.
4) *Stage 4:* The selected camera will report target visuals when they are detected.

The following scheduling edge is designated as the tracking edge that successfully detected the target in the previous round.

The scheduling strategy is at the core of CityTrac. In Table I, we shows the summary of symbols and their meanings. Formally, the scheduling strategy of CityTrac is denoted as the following minimization problem.

1) Hyperscale Tracking Costs: Assuming that edge $S \in \mathbf{E}$ is the scheduling edge, responsible for tracking the $|\mathbf{J}|$ targets, and the tracking time period for the jth target in the kth edge is from t^j_{ak} to t^j_{bk} with $\Delta t^j_k = t^j_{bk} - t^j_{ak}$. Next, the amount of data that needs to be processed in the kth edge when tracking target j can be calculated as follows:

$$l^j_k = M \cdot D^j_k \cdot \Delta t^j_k \tag{1}$$

where D^j_k is the amount of tracking data generated by one camera per time unit and M is the number of activated cameras when tracking the jth target. Let indicator $x^j_k = 1$ when edge S selects the kth edge as one of j's tracking edges in its schedule. The computation overhead CityTrac needs to minimize when tracking target set $\mathbf{J}$ among edge set $\mathbf{E}$ is

$$W = \sum_{j=1}^{|\mathbf{J}|} \sum_{k=1}^{|\mathbf{E}|} x^j_k \cdot l^j_k. \tag{2}$$

Intuitively asking all $|\mathbf{E}|$ edges to activate all of their cameras for tracking incurs prohibitive overheads in hyperscale cameras. Based on the observations on bad angles, overlapping, and unnecessary tracking, we note that a target's movement trajectory is limited to a certain tempo-spatial pattern, which provides opportunities to effectively reduce the searching space. Specifically, these movement patterns could be elaborated from objects' historical movement statistics.

2) Spatial Correlation and Movement Graph: To yield fine-grained spatial movement pattern discovery, two spatial levels of cameras' monitored areas are first introduced.

Definition 1 (Surveillance Field): The joint surveillance region is covered by a cluster of nearby cameras that share surveillance intersections and are controlled by the same edge server. *Field* for short.

Definition 2 (Transition Zone): The regions that a target may enter or leave a *Field*. *Zone* for short.

As shown in Fig. 1, a *Field* consists of several *Zones*, each corresponding to one moving direction in the covered roads. Objects move between *Zones*, so their spatial movement pattern lies in the activity of links between *Zones* in different

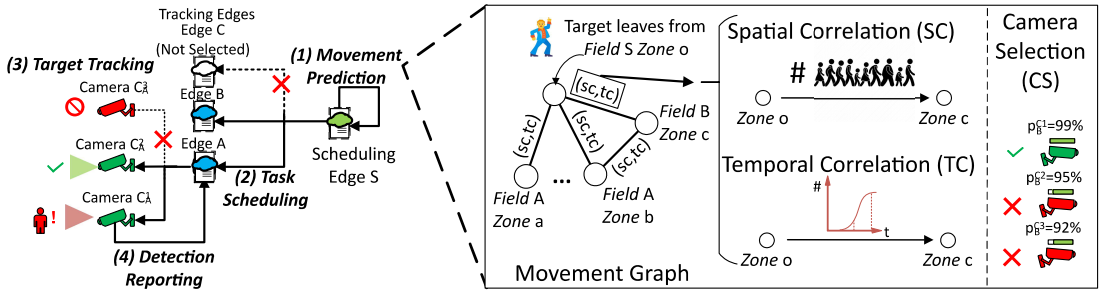

Fig. 5. CityTrac's 4-stage workflow based on a hierarchical structure of scheduling edge, tracking edges, and cameras.

TABLE I
SUMMARY OF SYMBOLS AND THEIR MEANINGS

Symbol	Meaning
$\mathbf{E}$	Set of edges in the network.
S	Scheduling edge responsible for tracking.
$\mathbf{J}$	Set of targets being tracked.
t_{ak}^j	Start time of tracking the j-th target at the k-th edge.
t_{bk}^j	End time of tracking the j-th target at the k-th edge.
Δt_k^j	Tracking time period for the j-th target at the k-th edge.
l_k^j	Amount of data to be processed at the k-th edge when tracking target j.
D_k^j	Amount of tracking data generated by one camera per time unit at the k-th edge for target j.
M	Number of activated cameras when tracking the j-th target.
x_k^j	Indicator variable, 1 if the k-th edge is selected as one of j's tracking edges, 0 otherwise.
W	Computation overhead.
$p_{sc}^{So\to m}$	Spatial correlation between *Zone* o and Zone m.
p_{So}^{exit}	Possibility of targets historically losing surveillance from *Zone* o.
$p_{sc}^{So\to exit}$	Sum of spatial correlations from *Zone* o to exit and all connected zones.
$p_{tc}^{a,b,\Delta t}$	Temporal correlation between *Zone* a and *Zone* b within time interval Δt.
$p_{Z,ZoneAcc}^c$	Empirical detection accuracy of the c-th camera for targets leaving or entering Zone Z.
$p_{reliability}$	Predefined reliability threshold for camera coverage.
$p_{F,FieldAcc}^c$	Overall detection accuracy within the camera's location *Field* F.
$Count(So, i)$	Number of objects monitored leaving Field S Zone o for Zone i.
$Count(So, exit)$	Number of objects leaving Field S Zone o without entering any other surveillance regions.
N	Total number of cameras activated.
p_{re}^Z	Overall reliability of *Zone* Z.
$\mathbf{C}^A$	Set of cameras located in *Field* A.
Z_A	Set of *Zones* in *Field* A.
cov_i^{A-j}	Indicator variable, 1 if camera i covers Zone j in *Field* A, 0 otherwise.
sel_i	Indicator variable, 1 if camera i is selected as the tracking camera, 0 otherwise.
p_{track}^j	Probability of successfully tracking target j.
F_k^i	A *Field* under the k-th edge.
$x_{F_k^i}^j$	Indicator variable, 1 if cameras in F_k^i are selected for tracking in the schedule, 0 otherwise.
v_j	Predefined importance of target j given by the stakeholder.

Fields, which is referred to as *spatial correlation*. For this, we build a *probabilistic target movement graph* (shown in Fig. 5) with *Zones* as its nodes, connections as the edges, and correlations as the weights. In this graph, each Zone is represented as a node, and edges connect Zones that have been historically observed as successive transit points for moving targets. The weight on each edge corresponds to the observed probability of transitioning between the two connected Zones. If an object is monitored moving from *Field* B *Zone* b to *Field* A *Zone* a, these two *Zones* are connected in the graph.

Formally, consider a target leaving a known Field S from a specific Zone o. Let $\mathbf{Z}_o$ be the set of all Zones directly reachable from Zone o based on historical observations. Each Count(So, m) denotes how many objects were observed to move from Zone o of Field S to Zone m, and Count$(So,$ exit$)$ represents how many objects left Zone o without entering any other surveillance zone (indicating they have effectively exited the monitored region). The spatial correlation between Zone o and Zone $m \in \mathbf{Z}_o$ is defined as an empirically derived probability

$$p_{sc}^{So\to m} = \frac{\text{Count}(So, m)}{\text{Count}(So, \text{exit}) + \sum_{i\in\mathbf{Z}_o} \text{Count}(So, i)}. \tag{3}$$

This ratio can be interpreted as a conditional probability that a target leaving Zone o of Field S will subsequently appear in Zone m instead of exiting the monitored network entirely. By definition, the probabilities over all connected Zones plus the exit scenario sum to 1

$$p_{sc}^{So\to \text{exit}} + \sum_{m=1}^{|\mathbf{Z}o|} p^{So\to m} sc = 1 \tag{4}$$

where $p^{So\to \text{exit}} sc = ([\text{Count}(So, \text{exit})]/[\text{Count}(So, \text{exit}) + \sum i \in \mathbf{Z}_o \text{Count}(So, i)])$ is the historical probability that a target is not observed in any subsequent Zone after leaving Zone o.

In practice, we estimate these probabilities using collected historical trajectories of targets. The resulting spatial correlation metrics can be visualized, for example, as a heatmap that highlights the likelihood of the target's future locations, given its current observed Zone.

3) Temporal Correlation: Targets do not only transition between Zones; they often do so within characteristic time intervals. Incorporating temporal dynamics refines the scheduling of tracking tasks by prioritizing Fields (and their respective Zones) that are likely to be reached by the target sooner.

For two connected Zones a and b in the movement graph, let Count$(a, b, -)$ be the total number of observed transitions from a to b and Count$(a, b, \Delta t)$ be the number of those transitions completed within a specified time interval Δt. The *temporal correlation* is then defined as

$$p_{tc}^{a,b,\Delta t} = \frac{\text{Count}(a, b, \Delta t)}{\text{Count}(a, b, -)}. \tag{5}$$

This probability quantifies how rapidly targets typically move from Zone a to Zone b. A higher value of $p_{tc}^{a,b,\Delta t}$ indicates that, historically, targets usually reach Zone b from Zone a within time Δt, suggesting that if the target has just left Zone a, deploying tracking resources at Zone b promptly might increase the likelihood of successful detection.

4) Camera Selection: Surveillance cameras unavoidably have detection defects for working under diverse contexts and imperfect algorithms. This may cause *false positive* situations where the target passes the camera in the monitoring interval while not being detected, as well as *false negative* situations where a camera reports a detection on a target that has not entered its *Zones*.

We denote the cth camera's empirical detection accuracy of targets leaving or entering Zone Z as $p_{Z,\text{ZoneAcc}}^{c}$. In CityTrac, we consider the camera *covers* the Zone Z only if $p_{Z,\text{ZoneAcc}}^{c} > p_{\text{reliability}}$. The value $p_{\text{reliability}}$ is predefined by stakeholders. In this article, $p_{\text{reliability}}$ is defined as 50%. The overall detection accuracy within the camera's location *Field* F is $p_{F,\text{FieldAcc}}^{c}$. If totally N cameras are activated to monitor *Zone* Z, then the overall reliability of *Zone* Z in this tracking task can be estimated with

$$p_{re}^{Z} = 1 - \prod_{c=1}^{N} \left(1 - p_{\text{ZoneAcc}}^{c}\right). \tag{6}$$

5) Tracking Optimization: Based on the above formulation, the possibility for CityTrac's scheduling to successfully track target j is given as

$$p_{\text{track}}^{j} = \sum_{k=1}^{|\mathbf{E}|} x_k^j \cdot \sum_{F_k^i \in \text{Field of } (k)} x_{F_k^i}^j \cdot p_{sc}^{So \to F_k^i p} \cdot p_{tc}^{o,p,\Delta t} \cdot p_{re}^{F_k^i} \tag{7}$$

where F_k^i represents a *Field* under the kth edge and $x_{F_k^i}^j$ is an indicator variable that equals 1 if cameras in F_k^i are selected for tracking in the schedule. Obviously, assigning tasks to fewer cameras could reduce computations, but at the cost of tracking failures. To maintain sufficient effectiveness, CityTrac requires the probability of losing target to retain in a controlled threshold

$$1 - p_{\text{track}}^{j} - p_{\text{exit}} < p_{\text{exit}} \cdot 10^{v_j}. \tag{8}$$

v_j is the predefined importance of target j, given by the stakeholder issuing the tracking task. Finally, the optimization problem for hyperscale tracking efficiency in CityTrac can be formulated as scheduling the indicator variables

$$\underset{x_k^j, x_{F_k^i}^j}{\text{Minimize}} \quad W = \sum_{j\in}^{|\mathbf{J}|} \sum_{k=1}^{|\mathbf{E}|} x_k^j \cdot l_k^j \tag{9a}$$

$$\text{s.t.} \quad \text{Eqs. (1), (3), (4), (5), (6), (7), (8).} \tag{9b}$$

Since $p_{sc}^{So \to F_k^i}$, $p_{tc}^{o,p,\Delta t}$, $p_{re}^{F_k^i}$, and p_{exit} are derived from historical data and remain fixed for a given scheduling scenario, they can be treated as constants. Therefore, p_{track}^{j} is a linear combination of the binary decision variables x_k^j and $x_{F_k^i}^j$.

Algorithm 1 HyMOT Scheduling Algorithm

Require: Zones $Z = \{Z_1, Z_2, \ldots, Z_n\}$, Targets $J = \{j_1, j_2, \ldots, j_t\}$, Edge Set $E = \{E_1, E_2, \ldots, E_m\}$, Spatial Correlation p_{sc}, Temporal Correlation p_{tc}, Reliability Threshold $p_{\text{reliability}}$, Probability of Losing Target p_{exit}

Ensure: Selected Zones and Edges for Tracking

```
Initialize Selected_Zones = ∅
Initialize Unselected_Zones = Z
for each target j ∈ J do
    Identify the target's leaving Zone Z_leave
    while constraints (1), (3), (4), (5), (6), (7) are not satisfied do
        Select Zone p ∈ Unselected_Zones with the largest spatial correlation p_sc with Z_leave
        Add p to Selected_Zones
        Set x_k^j = 1 and x_{F_k^i}^j = 1 for the corresponding edge k and Field F_k^i that covers p
        Call the Camera Selection Strategy in Algorithm 2
        Remove p from Unselected_Zones
    end while
end for
Return Selected_Zones
```

This is a typical linear programming problem, which is proved to be NP-Hard in computation. We use a greedy strategy named hyperscale multicamera object tracking (HyMOT) to solve it, that is, for each issued tracking, unselected *Zone* p with the largest spatial correlation with the target's leaving *Zone* is added to the scheduling pool (i.e., $x_k^j = 1$ and $x_{F_k^i}^j = 1$ with p a *Zone* in F_k^i) one by one. For each Zone, CityTrac selects the cameras with special designed *camera selection strategy* (Proposed in Algorithm 2) until the constraint in (6) is satisfied. The Pseudo Code of the Scheduling is given in Algorithm 1.

6) Camera Selection Strategy in CityTrac: It is not easy for Edge S to determine the optimal camera clusters to track the target. Taking *Field* A as an example, edge S needs to assure that each Zone is covered by at least one camera, and the reliability of each *Field* is maximized. Additionally, to meet the requirement of tracking cost minimization, the overall number of activated cameras must be minimized. Denote the set $\mathbf{C}^A$ as the cameras located in the *Field* covered by edge A (*Field* A for simplicity), Z_A as the set of *Zones* in *Field* A, $\text{cov}_i^{A-j} = 1$ if camera i covers the Zone j in the *Field* A, and $\text{sel}_i = 1$ if camera i is selected as the tracking camera in the next tracking round, then we have

$$\text{Minimize} \quad \sum_{i=1}^{|\mathbf{C^A}|} \text{sel}_i \tag{10a}$$

$$\text{Maximize} \quad p_{A,\text{FieldAcc}}^{i} \forall i \ \text{sel}_i = 1 \tag{10b}$$

$$\text{s.t.} \quad \sum_{i=1}^{|\mathbf{C^A}|} \text{sel}_i \text{cov}_i^{A-j} > 0 \ \forall j \in Z_A. \tag{10c}$$

This is a multiobject optimization problem. To address this problem, a camera selection strategy is proposed. Taking

Algorithm 2 Camera Selection Strategy

Require: The list of cameras within *Field* A C_a,
the reliability requirement of each *Zones* p_{reBond},
the minimal acceptable reliability $p_{reliability}$,
list of not *well addressed* Zone Z_{orig}

Ensure: The list of chosen camera C_{chosen}

```
Sort all the cameras within the target Field in decrement order based on the number of Zones they covered. If two cameras cover the same amount of Zones, then cameras with higher reliability values are sorted first.
Add the first camera c1 to C_chosen and remove it from C_a.
New zone array Z_wa.
for Each Zone z covered by camera c1 do
    if p^z_re > p_reBond then
        Add z to Z_wa and remove it from Z_orig
    end if
end for
if 1 − (1 − p_reliability)^|Z_orig| < p_reBond then
    for Each camera c within C_a do
        if c covers at least one Zone within Z_orig then
            Add c to C_chosen and remove it from C_a.
            for Each Zone z covered by camera c do
                if p^z_re > p_reBond then
                    Add z to Z_wa and remove it from Z_orig
                end if
            end for
            if Z contains all Zones within the Field then
                break;
            end if
        end if
    end for
else
    Resort the Z_orig according to the Field detection accuracy.
    for Each camera c within C_a do
        if c covers at least one Zone within Z_orig then
            Add c to C_chosen and remove it from C_a.
            for Each Zone z covered by camera c do
                if p^z_re > p_reBond then
                    Add z to Z_wa and remove it from Z_orig
                end if
            end for
            if Z contains all Zones within the Field then
                Break;
            end if
        else
            Continue;
        end if
    end for
end if
Return C_a
```

Field A as an example, assume that the overall reliability of each Zone should be higher than p_{reBond}. the main idea of the camera selection strategy is as follows: First, edge S sorts the camera list within *Field* A based on the number of *Zones* the camera covers. Cameras covering more *Zones* are sorted first. If there are multiple cameras covering the same amount of *Zones*, then cameras with higher $p^c_{A,\text{FieldAcc}}$ are sorted first. The edge S adds the first camera to the selected batch. That means the camera is selected to track the target in the next round. Then, edge S checks if the camera covers all the *Zones* in the target *Field* and if the reliability of each Zone meets the requirement. If Zone e is covered and the reliability p^e_{re} is higher than the requirement, then Zone e is *well addressed*.

Assuming that there are u_{total} *Zones* in the *Field* A, and u *Zones* are not *well addressed*. Then, Edge S checks if $1-(1-p_{\text{reliability}})^u < p_{\text{reBond}}$. If $1-(1-p_{\text{reliability}})^u > p_{\text{reBond}}$, then keep selecting the camera within the sorted camera list that covers at least one not *well addressed* Zone until all the *Zones* are well addressed. In this case, the proposed strategy activates at most $\lceil \log^{p_{\text{relyBond}}}_{1-(1-p_{\text{reliability}})^u_{\text{total}}} \rceil - 1$ more cameras than the optimal solution. If $1-(1-p_{\text{reliability}})^u < p_{\text{reBond}}$, then we resort the camera list by the $p^c_{A,\text{FieldAcc}}$. Then, edge S selects the camera that covers the not *well addressed* Zone with the highest reliability until all the *Zones* are *well addressed*. In this case, the proposed strategy activates at most $u-1$ more cameras than the optimal solution. The pseudo-code of the Camera Selection Strategy is shown in Algorithm 2.

To calculate the approximate ratio, first, assume that there are u_{total} *Zones* within *Field* A. Considering the worst case, with the Camera Selection Strategy, the selected camera is no more than $\lceil \log^{p_{\text{relyBond}}}_{1-(1-p_{\text{reliability}})^u_{\text{total}}} \rceil + 1$, and the optimal solution is to select 2 cameras. Thus, the approximate ratio of the Camera Selection Strategy is no more than $2/(\lceil \log^{p_{\text{relyBond}}}_{1-(1-p_{\text{reliability}})^u_{\text{total}}} \rceil + 1)$.

B. Analysis

1) Time Complexity Analysis: Our greedy optimization strategy involves several key steps, each contributing to the overall time complexity.

1) *Zone Sorting and Selection (Based on Algorithm 1):* We sort all unselected Zones based on their spatial correlation with the target Zone. This sorting operation has a time complexity of $O(|Z|\log|Z|)$, where $|Z|$ represents the total number of Zones.
2) *Camera Selection (Based on Algorithm 2):* For each selection of cameras, we traverse the list of available cameras and perform sorting based on coverage and reliability. This step has a time complexity of $O(|C|\log|C|)$, where $|C|$ is the total number of cameras.
3) *Overall Time Complexity:* In the worst-case scenario, where all Zones need to be processed, the cumulative time complexity becomes $O(|Z|^2\log|Z|+|Z||C|\log|C|)$. This reflects the nested nature of Zone and camera processing within our greedy approach.

2) Scalability Analysis: Our greedy algorithm is designed with scalability in mind, ensuring efficient performance even as the number of Zones and cameras increases. Our system operates in a fully decentralized manner during the execution of target tracking tasks. This decentralization ensures that no matter how many edge servers and cameras are deployed

across the network, the performance of a specific scheduling server is only affected by the number of cameras and Zones within its corresponding Field.

1) *Zone-Based Scalability:* The number of Zones typically depends on geographical factors, such as intersections and their connections within the deployment area of an edge server. In our experimental datasets, a Field typically contains 3–4 Zones. Consequently, the number of Zones tends to be a small fixed value. As the number of Zones increases, the time complexity grows as $O(n^2)$.
2) *Camera-Based Scalability:* In practice, it is more common to deploy additional cameras within the same Field. CityTrac efficiently handles this scenario. When the number of cameras increases within a Field, the time complexity grows as $O(n)$. This linear growth ensures that the system remains efficient even when more cameras are added.

This scalability ensures that our approach remains practical for large-scale deployments. Additionally, the decentralized architecture allows for parallel processing of different Fields across multiple edge servers, further enhancing performance in hyperscale environments.

3) Closeness to the Optimal Solution: In CityTrac, the overall scheduling strategy comprises two key algorithms: 1) the HyMOT scheduling strategy and 2) the camera selection strategy. In the following sections, we analyze the approximation ratio of each algorithm individually and then provide a combined analysis.

1) *Scheduling Strategy (Algorithm 1):* The scheduling algorithm selects Zones based on the highest spatial correlation. This problem is structurally similar to the *Set Cover Problem*, which is known to have a greedy approximation ratio of $\ln n + 1$, where n is the number of Zones.
2) *Camera Selection Strategy (Algorithm 2):* The camera selection strategy aims to cover the selected Zones while meeting reliability constraints. This can be modeled as a *Weighted Set Cover Problem*, also solvable with a greedy approach that provides an approximation ratio of $\ln m + 1$, where m is the number of cameras.
3) *Combined Approximation Ratio:* Since the scheduling and camera selection are sequential processes, the overall approximation ratio can be expressed as the product of the two individual ratios

$$\frac{A_{\text{combined}}}{\text{OPT}_{\text{combined}}} \leq (\ln n + 1) \cdot (\ln m + 1)$$

where: A_{combined} is the total cost achieved by our algorithm, $\text{OPT}_{\text{combined}}$ is the total cost of the optimal solution, n is the number of Zones. and m is the number of cameras.

This analysis indicates that the deviation from the optimal solution grows logarithmically with the problem size (n and m). In large-scale scenarios, this tradeoff between computational efficiency and solution quality is practically acceptable. Our algorithm efficiently balances performance and complexity, making it suitable for hyperscale tracking systems.

4) Detailed Explanation of the Linear Programming Transformation: In our formulation, the tracking success probability p^j_{track} is defined as

$$p^j_{\text{track}} = \sum_{k=1}^{|\mathbf{E}|} x^j_k \cdot \sum_{F^i_k \in \text{Field of } (k)} x^j_{F^i_k} \cdot p_{sc}^{So \to F^i_k} \cdot p_{tc}^{o,p,\Delta t} \cdot p_{re}^{F^i_k} \quad (11)$$

where:

- x^j_k: A binary decision variable indicating whether edge k is selected for tracking target j.
- $x^j_{F^i_k}$: A binary decision variable indicating whether Field F^i_k under edge k is selected for tracking target j.
- $p_{sc}^{So \to F^i_k}$: The spatial correlation probability, representing the likelihood of a target moving from Zone So to Field F^i_k. This is derived from historical movement data.
- $p_{tc}^{o,p,\Delta t}$: The temporal correlation probability, representing the likelihood of a target moving from Zone o to Zone p within a time interval Δt. This is also derived from historical movement data.
- $p_{re}^{F^i_k}$: The reliability probability of Field F^i_k, representing the likelihood that cameras in F^i_k successfully detect a target. This is determined based on historical camera detection performance.
- p_{exit}: The probability of losing a target, i.e., the likelihood that a target leaves a Zone and does not re-enter any surveillance region. This is precomputed from historical statistics.

Since $p_{sc}^{So \to F^i_k}$, $p_{tc}^{o,p,\Delta t}$, $p_{re}^{F^i_k}$, and p_{exit} are derived from historical data and remain fixed for a given scheduling scenario, they can be treated as constants. Therefore, p^j_{track} is a linear combination of the binary decision variables x^j_k and $x^j_{F^i_k}$.

Given the constraint on tracking success probability

$$1 - p^j_{\text{track}} - p_{\text{exit}} < p_{\text{exit}} \cdot 10^{v_j} \quad (12)$$

substituting p^j_{track} results in the following linear inequality:

$$1 - \sum_{k=1}^{|\mathbf{E}|} x^j_k \cdot \sum_{F^i_k \in \text{Field of}(k)} x^j_{F^i_k} \cdot p_{sc}^{So \to F^i_k} \cdot p_{tc}^{o,p,\Delta t} \cdot p_{re}^{F^i_k} - p_{\text{exit}} < p_{\text{exit}} \cdot 10^{v_j}. \quad (13)$$

This constraint remains linear because it consists solely of constants and binary decision variables multiplied together.

Thus, the optimization problem can be formulated as a linear programming problem with the decision variables x^j_k and $x^j_{F^i_k}$, while all correlation and reliability terms are treated as constants derived from historical data. We summarize the variables and constants in the LP Model in Table II.

TABLE II
SUMMARY OF VARIABLES AND CONSTANTS IN THE LP MODEL

Symbol	Description	Type	Source
x^j_k	Edge k selected for tracking j	Binary	Decision
$x^j_{F^i_k}$	Field F^i_k selected for tracking j	Binary	Decision
$p_{sc}^{So \to F^i_k}$	Spatial correlation probability	Constant	Historical
$p_{tc}^{o,p,\Delta t}$	Temporal correlation probability	Constant	Historical
$p_{re}^{F^i_k}$	Reliability probability of Field F^i_k	Constant	Historical
p_{exit}	Probability of losing the target	Constant	Historical

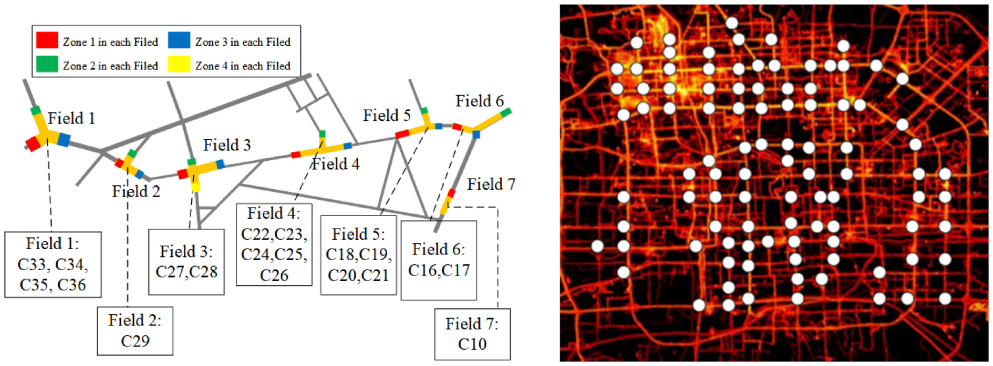

Fig. 6. Examples of dataset preprocessing. (a) Division of *Field* in CityFlow dataset. (b) Example of the manually selected *Field* in the region of interest. Each *Field* covers at least one trajectory.

VI. Experiments

A. Methodology

1) Baselines: For comparison, we use the direction-aware scheduling method Khochare et al. [41] and coarse-grained spatial-correlation-based scheduling method Jain et al. [40]. These two state-of-the-art tracking strategies are introduced in Section III-B.

2) Surveillance Simulation: Simulations are conducted for both small-scale and large-scale camera deployments based on the datasets *Cityflow* and *Geolife*, respectively. A brief description of the two datasets is provided as follows.

Cityflow: The CityFlow dataset is a comprehensive urban traffic camera dataset with over 3 h of synchronized HD video from 40 cameras at 10 intersections. It includes 200K annotated bounding boxes and provides camera geometry and calibration details. This dataset supports tasks like MTMC tracking, MTSC tracking, object detection, and image-based vehicle reidentification, offering valuable data for spatiotemporal analysis in urban environments.

Geolife: The Geolife dataset, from Microsoft Research Asia, comprises GPS trajectory data of 182 users in Beijing and nearby areas, collected from 2007 to 2012. It includes detailed movement patterns, timestamps, and semantic annotations, gathered using GPS recorders and Windows Mobile applications (Table III). This dataset is valuable for research in urban planning, transportation management, and mobility behavior analysis.

3) Preprocessing: Due to the datasets not fully aligning with our experimental setup, we performed specific preprocessing steps. As shown in Fig. 6(a), for *Cityflow*, we estimate tempo-spatial correlations with scenario S05 and test performance with scenario S04. A total of 7 *Fields* are constructed for S04, each containing at least 1 camera and 1 *Zone*.

The correlation among cameras, *Fields*, and *Zones* is shown in Table IV.

For *Geolife Dataset*, 10561 labeled tracklets are picked from it for testing. We simulate a city-scale of 103 *Fields* with 4 *Zones* and 4 cameras in each *Field*. 25% of the trajectory data is used to build the table in Spatula and the movement graph in CityTrac, and the rest is used for tracking evaluation (i.e., around 8000 targets).

TABLE III
Types and Amounts of Data Included in the Geolife Dataset

Transportation Mode	Distance (km)	Time (hours)
Walking	10,123	5,460
Bicycling	6,495	2,410
Bus	20,281	1,507
Car and Taxi	32,866	2,384
Train	36,253	745
Airplane	24,789	40
Other	9,493	404
Total	140,304	12,950

The Geolife dataset, which contains only GPS trajectories, is not directly suitable for object tracking simulations. To address this, we preprocessed the data using over 2300 lines of Java code.

Due to the large geographical span of the dataset, most trajectories are confined to a smaller region. Therefore, we focused on a region between longitudes 116.039894 and 116.8001529 and latitudes 39.680027 and 40.295658, dividing it into 1000×1000 clusters. This ensures that even the shortest trajectories cover at least two units. In this region, 103 *Fields* were manually selected, each containing 4 *Zones*, as illustrated in Fig. 6(b). The simulation ensures that each *Field* has at least one trajectory passing through it. Finally, four cameras were placed in each *Field*, ensuring that each *Zone* is covered by at least one camera.

4) Metrics: We measure tracking efficiency using *computation workload* (i.e., the number of processed frames/days), *activations* (i.e., the number of activated cameras during tracking), and *tracking rounds* (i.e., the number of rounds needed for performing tracking tasks). We measure the tracking accuracy with *failure rate*, which is the number of detection failures in three tracking methods. Please note that here we consider it a failure when none of the activated cameras capture the target, and the target passes through other inactivated cameras.

B. Evaluation Results

The performance of these three strategies is first evaluated using the Cityflow dataset. In the initial round, the v_j of all

TABLE IV
CORRELATION AMONG *Fields* AND *Zones* IN CITYFLOW DATASET

Field ID	Field 1			Field 2			Field 3			Field 4				Field 5			Field 6			Field 7
Location	42.49969, -90.69633			42.49809, -90.69092			42.49831, -90.68822			42.49922, -90.68089				42.50007, -90.67357			42.49966, -90.67224			42.49825, -90.67307
Cameras	C33,C34,C35,C36			C29			C27, C28			C22,C23,C24,C25,C26				C18,C19,C20,C21			C16, C17			C10
Zones	Zone 1	Zone 2	Zone 3	Zone 1	Zone 2	Zone 3	Zone 1	Zone 2	Zone 3	Zone 4	Zone 1	Zone 2	Zone 3	Zone 1	Zone 2	Zone 3	Zone 1	Zone 2	Zone 3	Zone 1
Covered By	C33, C35, C36	C33, C35	C33, C34	C29	C29	C29	C28	C28	C28	C27, C28	C25, C26	C24	C22, C23, C24	C19, C20, C21	C19	C18, C19	C16, C17	C16, C17	C16, C17	C10

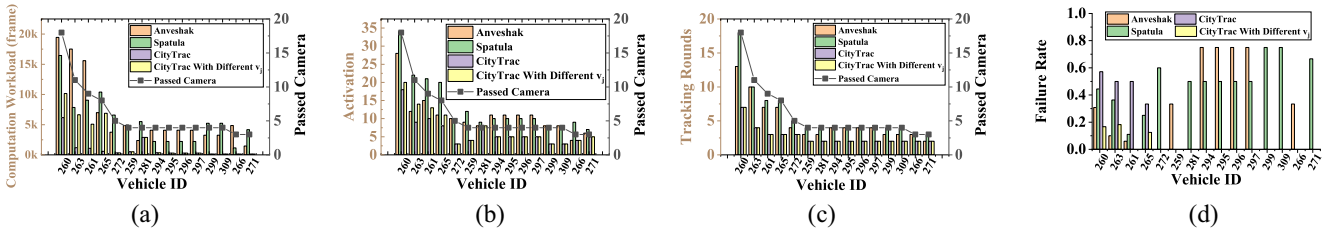

Fig. 7. *Evaluation results* of three tracking scheduling methods on *Cityflow*. (a) Computation workload. (b) Activation number. (c) Tracking rounds. (d) Failure rate.

TABLE V
ABLATION STUDY RESULT ON TARGET 261

Passed Field	Field 5			Field 4					Field 3
Passed Camera	C19	C20	C21	C22	C23	C24	C25	C26	C27
Activated Cameras	Spatula	Round 1 C20,C17	Round 2 C21,C19,C18	Round 3 C22,C20,C19	Round 4 C21,C24	Round 5 C22 Failed	Round 6 C23,C27,C26,C25	Round 7 C23,C26	Round 8 C24 Failed
	Anveshak	Round 1 C20,C22,C23,C24	Round 2 C21	Round 3 C20,C19,C22	Round 4 C24	Round 5 C24,C26	Round 6 C27	Round 7 C25,C26,C28	Round 8 null Failed
	CityTrac	**Round 1** C20		**Round 2** C24,C25					**Round 3** C28

targets is set to 1. CityTrac's tracking results in this round are shown in purple. As shown in Fig. 7, the total computation workload, the number of cameras activated, and the needed tracking round of CityTrac are much smaller than those in Spatula and Anveshak in all the tested cases, demonstrating superior efficiency in this small-scale surveillance scenario. From Fig. 7(c), it is observed that for targets passing through more than 8 cameras, the failure rate of CityTrac is relatively higher compared to the baselines. However, for targets appearing in fewer than 5 cameras, CityTrac achieves 100% accuracy, with no failures. This is because the amount of data used to construct the spatial-temporal correlations is relatively small, making the correlation statistics needed for long-range tracking (i.e., > 8 cameras) not sufficiently revealed.

To enhance CityTrac's performance, we assigned higher v_j values to targets 260, 263, 261, and 265. The yellow bar in Fig. 7 represents the result when v_j of them are set to 3. The results indicate that with higher v_j, the number of activated cameras and the overall computational workload increase significantly. However, there is a substantial reduction in the failure rate.

As for the tradeoffs between accuracy and resource usage, the balance between these two factors often depends on the deployment of devices in different Fields. For example, during the tracking of Target 260, increasing the workload by 80% (frames analyzed) reduced the failure rate from 58% to 17%. In contrast, for Target 261, achieving a similar reduction in failure rate required increasing the workload by a factor of 5. This demonstrates that the relationship between accuracy and resource usage varies significantly depending on the complexity and characteristics of the target's trajectory and the camera deployment in each Field.

To further evaluate the details of the three system performances, an ablation study is conducted on target 261. The result is shown in Table V. Target 261 enters the monitor *Field* from C19. According to the spatio-tempo correlation among cameras, the target leaving C19 has more than 56% of possibility reaching C17, and the possibility of reaching C20 is more than 33%. Thus, it activates both C17 and C20. This mechanism works well until the target leaves C26. Because C24 and C26 are deployed next to each other, more than 94% of the target leaves C26 and reaches C24. Thus, the Spatula only activates the C24. However, this is true only if the target is moving inside *Field* 4. But target 261 leaves *Field* 4 and reaches *Field* 3. The Spatula failed to activate the camera within *Field* 3. The performance of Anvishark is much better. However, the results show that cameras are activated repeatedly and redundantly, leading to wasted resources.

A large-scale evaluation is then carried out, aiming to reveal the performance of different methods in hyperscale tracking. As shown with the results in Table VI, CityTrac requires 94% fewer computations and 91% fewer computations for tracking

TABLE VI
THREE METHODS' PERFORMANCE ON GEOLIFE

	Computation Workload(frames)	Maximum Activations	Tracking Round	Average Failure Rate			
Spatula	6.696×10^9	85	27	0.147			
Anveshak	4.28×10^9	78	26	0.162			
CityTrac	0.383×10^9	**60**	**21**	**Overall** 0.140	**Level 3** 0.127	**Level 2** 0.136	**Level 1** 0.174

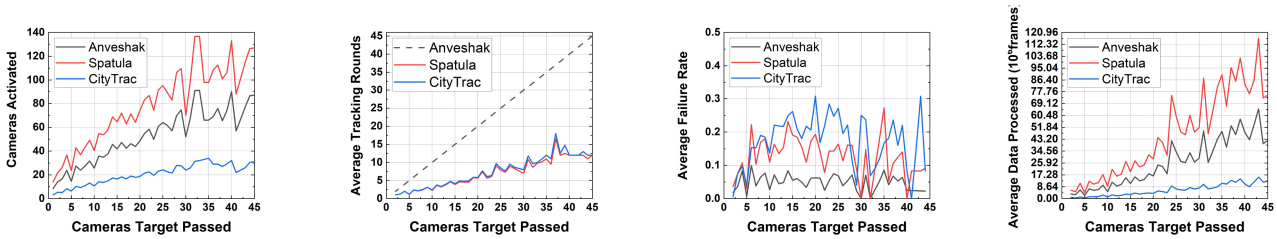

Fig. 8. *Evaluation results* of three tracking scheduling methods on *Geolife*. (a) Activation number. (b) Tracking rounds. (c) Failure rate. (d) Computation workload.

around 8000 targets, compared with Spatula and Anveshak, respectively. Meanwhile, CityTrac introduces a similar, even smaller, failure rate than the baselines while requiring less camera activation and tracking rounds. Among them, the target with a higher importance value has a lower failure rate. Even with level 2 importance level can assure a much lower failure rate than both Anveshak and Spatula. But the resource consumption is also higher.

To provide a more detailed comparison of the performance of the three methods on targets moving different distances, Fig. 8 compares the performance of the three methods on the Geolife dataset when targets pass through different numbers of cameras. Fig. 8(a) shows the average number of cameras activated by the three strategies as targets pass through different numbers of cameras. It can be observed that in all scenarios, *CityTrac* consistently activates significantly fewer cameras compared to Spatula and Anveshak. Spatula activates substantially more cameras than Anveshak. This is because when cameras are clustered, the movement of a target often causes it to enter the monitoring range of multiple cameras simultaneously. Due to the spatiotemporal correlation of each camera, these overlapping coverage areas result in multiple cameras being activated, leading to computational resources in the Spatula strategy being wasted on analyzing video streams from the same area, causing unnecessary redundancy. Fig. 8(b) compares the average number of tracking rounds for the three strategies as targets pass through different numbers of cameras. Notably, Anveshak, which does not consider tracking rounds under different scenarios by design, would trigger a tracking round each time a camera is activated. Thus, its curve is represented by the black dashed line in the figure. Compared to the number of activated cameras, *CityTrac* consistently shows a significantly lower number of tracking rounds than Anveshak, while Spatula's number of activated cameras is not vastly different from *CityTrac*. Fig. 8(c) illustrates the average failure rate of the three strategies as targets pass through different numbers of cameras. It can be seen that Anveshak's failure rate is much lower than the other two methods, while *CityTrac* and Spatula do not exhibit significant differences in failure rates. CityTrac's failure rate is relatively higher when activating 15-25 cameras, while Spatula is more likely to fail when targets pass more than 30 cameras. This is because Anveshak's conservative scheduling strategy tends to call on more cameras to track the target, leading to a much lower average failure rate compared to the other two strategies. Fig. 8(d) compares the average amount of data processed by the three strategies as targets pass through different numbers of cameras. From the figure, it is evident that *CityTrac* processes far less data compared to Anveshak and Spatula, with Spatula requiring slightly more data processing than Anveshak.

C. Comparative Analysis in Challenging Scenarios

This section provides an in-depth comparative analysis of CityTrac, Spatula, and Anveshak under two primary challenges: 1) occlusions and 2) dynamic trajectory changes.

1) Occlusions: Occlusions present significant challenges in tracking systems, often resulting in increased camera activations, higher system loads, and more frequent tracking failures. To demonstrate how each method handles occlusions, we analyze Target 260 from the CityFlow dataset. As depicted in Fig. 9, Target 260 was selected for two main reasons.

1) It traverses five different *Fields* in our experiments, providing an opportunity to compare camera scheduling strategies for long-distance target movements and highlighting differences in tracking rounds and system load.
2) Among the *Fields* traversed by Target 260, *Field* 5 (Camera C16) and *Field* 1 (Camera C33) present significant occlusion challenges. Citytrac successfully tracked the target in both *Fields*. By analyzing how the three systems manage tracking in these *Fields*, we can better illustrate the tradeoffs between accuracy and resource usage.

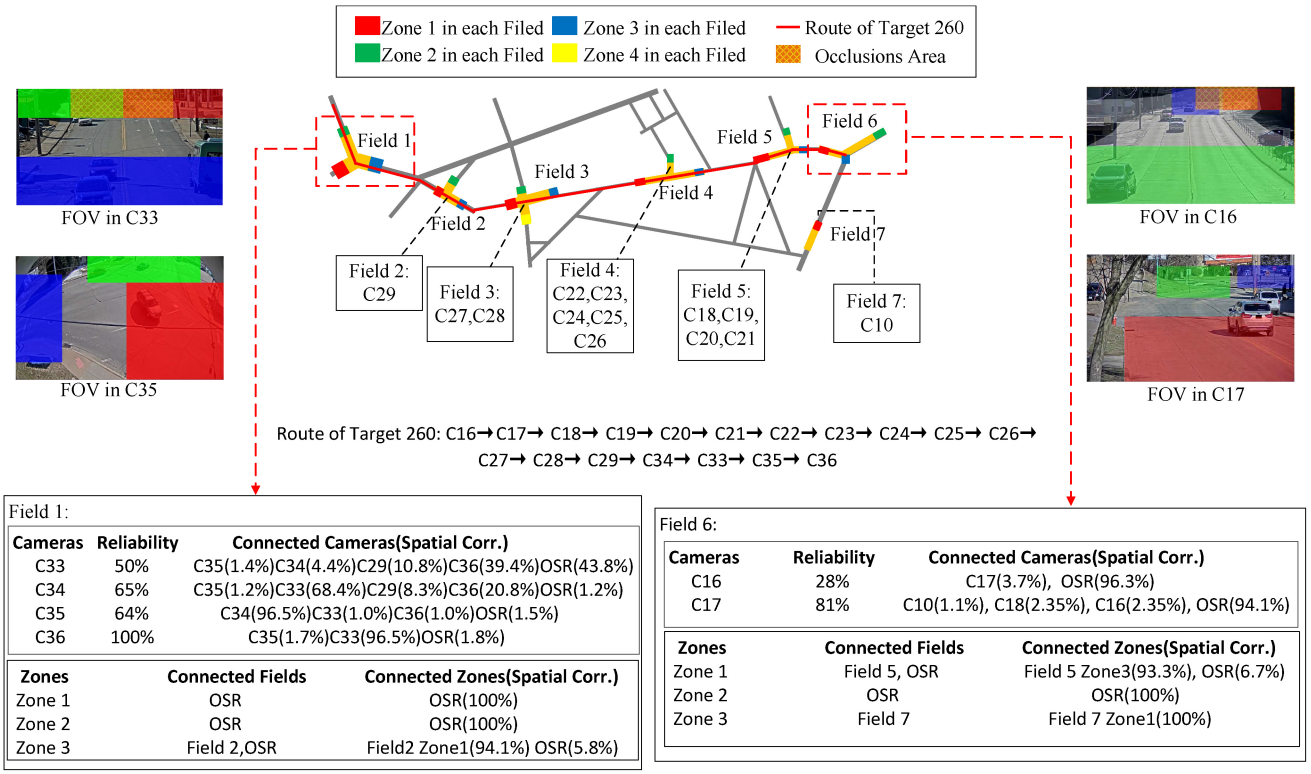

Fig. 9. Route analysis and zone reliability for target 260. OSR stands for out of the surveillance region.

TABLE VII
QUANTITATIVE COMPARISON OF TRACKING METHODS

Method	Tracking Rounds	Frames Analyzed	Failures
Spatula	18	16,464	8
Anveshak	14	19,458	4
CityTrac	6	6,136	2

Anveshak:
C16→C17,C19,C20→C18→C20,C22,C23,C24→C22,C20,C19,C23,C24→C24→C24,C26→C27→C29→C33→C33,C34,C35→C35,C33,C36→C36,C33,C34

Spatula:
C16→C17→C10,C16,C18→C20,C17→C21,C19,C18→C22,C20,C19→C21,C24→C22→C23,C27,C26,C25→C23,C26→C28,C26→C29→C33,C28→C29,C36→C33,C29,C36→C34

CityTrac:
C16→C17→C19,C20→C20,C24,C25→C25,C28→C28,C29,C36→C29,C35

Fig. 10. Camera activation sequences for target 260 across three methods. Arrows represent individual tracking rounds.

Comparison in Field 6
Anveshak: C16→C17,C19,C20 2 rounds of Analysis, 4 Video Streams Analyzed

Spatula: C16→C17→C10,C16,C18 3 rounds of Analysis, 5 Video Streams Analyzed

CityTrak: C16→C17→C19,C20 3 rounds of Analysis, 4 Video Streams Analyzed

Fig. 11. Detailed comparative analysis of three methods in Field 6. Arrows represent individual tracking rounds.

The comparison evaluates CityTrac's performance against Spatula and Anveshak along the route involving cameras C16 to C36. This route includes challenging regions, such as *Field* 6 (C16/C17) and *Field* 1 (C33/C34/C35/C36), which are particularly prone to occlusions.

Table VII presents a quantitative comparison of the three tracking methods. During the tracking of Target 260, Spatula required 18 tracking rounds and analyzed 16 464 video frames, while Anveshak required 14 tracking rounds and analyzed 19 458 video frames. In contrast, CityTrac only needed 6 tracking rounds and analyzed 6136 video frames. Additionally, Spatula experienced 8 tracking failures (instances where the target was not detected in the analyzed video frames), Anveshak had 4 failures, and CityTrac had 2 failures. The camera activation strategies for each method are illustrated in Fig. 10.

The results indicate that Spatula activates cameras conservatively (e.g., starting with C17), leading to multiple rounds of analysis and a higher system load. Anveshak activates multiple cameras based on trajectory predictions (e.g., C16, C19, and C20), resulting in redundant video analysis. Conversely, CityTrac strategically selects reliable zones, optimizing camera activation and reducing redundancy.

We further analyzed the scheduling patterns of the three strategies in *Field* 5 (Fig. 11) and *Field* 1 (Fig. 12). In *Field* 6, Anveshak tends to activate C16 first and predicts the target's next movement direction based on the detected trajectory. This approach requires Anveshak to analyze three video streams—C19, C20, and C17—in subsequent tracking

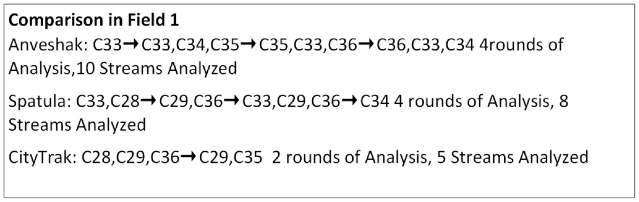

Fig. 12. Detailed comparative analysis of three methods in Field 1. Arrows represent individual tracking rounds.

rounds. In contrast, Spatula activates C17 initially regardless of the target's direction. While this conservative strategy is reliable, it imposes a greater load on the system. Specifically, Spatula requires three rounds of video analysis in this area: C16 in the first round, C17 in the second round, and C10, C19, and C20 in the third round.

For CityTrac, due to the low reliability of C16 and the overlapping monitoring coverage between C17 and C16, no further analysis was conducted based on C16 after the initial target detection. The tracking of Target 260's subsequent movements relied entirely on the analysis results from C17. As a result, while both CityTrac and Spatula conducted three rounds of analysis in *Field* 6, the video data analysis resources consumed by CityTrac at C16 were significantly lower compared to Anveshak and Spatula.

In *Field* 1 (C34, C33, C35, and C36), the differences become even more pronounced. After the target exits C29, Anveshak conducts four rounds of video analysis: C33 in the first round; C33 and C34 in the second; C35, C33, and C36 in the third; and C33, C35, and C34 in the fourth round. Spatula undergoes five rounds: C33 and C28 in the first round; C29 and C36 in the second; C33, C29, and C36 in the third; C34 in the fourth; and C33 in the fifth round. In comparison, CityTrac completes the tracking process in just two rounds: C29, C28, and C36 in the first round, followed by C29 and C35 in the second round.

C16 and C17, as well as C33 and C35, are two pairs of cameras with overlapping monitoring coverage. However, since C35 and C17 effectively overcome the occlusion issues present in C33 and C16, respectively, CityTrac was able to successfully track the target in these Fields despite activating fewer cameras compared to other methods. This demonstrates that CityTrac's mechanism effectively mitigates the impact of occlusions on video analysis results.

2) Dynamic Trajectory Changes: Dynamic trajectory changes—frequent turnarounds, abrupt directional shifts, or previously unseen movement patterns—can invalidate trajectory-based predictions and correlation-based lookups. Since neither CityFlow nor our secondary dataset contained sufficient real-world examples, we simulated 10 targets with random dynamic movements in a Geolife-like setting.

Table VIII shows the results for the 10 simulated dynamic targets. Each row summarizes the number of cameras and Fields passed, followed by whether each system successfully tracked the target.

In this simulation, Anveshak succeeded with only 1 out of 10 targets. Spatula and CityTrac tracked significantly more targets (50% and 60% success rates, respectively), primarily because both methods can leverage existing spatial correlation data for partial reidentification. However, as the target's path becomes longer or more erratic, the likelihood of encountering a route not captured by the correlation data increases, leading to eventual tracking failures in both systems. From the result, it is easy to conclude that Anveshak forecasts the future trajectory based on historical data. Under highly dynamic movements, these predictions become unreliable, resulting in frequent missed targets. Spatula and CityTrac rely on previously recorded spatial correlation data, which helps to compensate for unforeseen movement changes—provided that the target's new path has some overlap with existing correlation records.

In a real-world setting, these results imply that if a target moves along a completely new route (i.e., one not previously observed), all three systems would struggle. One potential mitigation strategy is to operate a subset of cameras continuously, ensuring partial coverage even for novel routes. Addressing this strategy in full detail, however, lies beyond the scope of this article.

D. Performance and Scalability Analysis

We further focus on analyzing latency and scalability in a real-time scenario in this part. Here, we compare the performance of CityTrac with Anveshak and Spatula during the tracking of Target 260 in Cityflow dataset, highlighting the system's capability to achieve large-scale, decentralized target tracking even with limited hardware resources.

CityTrac operates in a fully decentralized manner for target tracking tasks. This decentralized design ensures that the performance of a specific scheduling server is influenced only by the number of cameras and *Zones* within its corresponding *Field*, regardless of the total number of edge servers and cameras deployed across the network.

To simulate the computational capabilities of cost-constrained, large-scale deployments, we used three resource-constrained Lenovo workstations as edge servers. Each workstation is equipped with an Intel i5 CPU, an NVIDIA RTX 3060 GPU, 32GB RAM, and an 1 TB hard drive. We chose this number because, during the tracking of Target 260, a maximum of three edge servers operate simultaneously. According to the timestamps provided by the Cityflow dataset, the target required approximately 175 s to pass through all monitored regions. the average latency before the next batch of data arrives for processing is approximately 2.87 s. Hence, to ensure near real-time performance, the video analytics pipeline should aim to match this timeframe as closely as possible. The detection algorithm employed in these experiments is the same as that used in [3].

The experimental results are shown in Fig. 13.

According to the result, Anveshak took 23.15 min to complete data processing without load-balancing algorithms, which is unacceptable for real-time scenarios due to significant redundant computation. Spatula performed better, with a processing time of 13.8 min. However, it still experienced substantial delays toward the end of the tracking task. CityTrac

TABLE VIII
TRACKING RESULTS FOR DYNAMIC TARGETS

ID	Passed Cam	Passed Fields	Anveshak Successes	Spatula Successes	CityTrac Successes
1	25	7	0	0	0
2	15	5	0	0	0
3	29	8	0	0	0
4	21	7	0	0	0
5	17	5	0	1	0
6	12	3	0	1	1
7	19	3	0	1	1
8	4	2	1	1	1
9	9	3	0	1	1
10	10	3	0	1	1

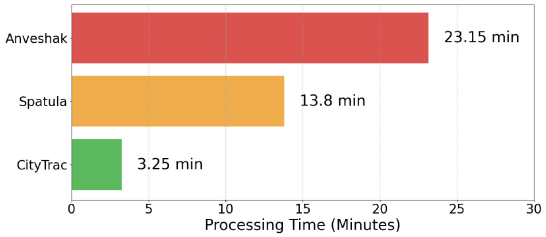

Fig. 13. Comparison of processing times for different strategies.

demonstrated the best performance, achieving a processing delay of just 3.25 min (195.04 s). This level of delay is sufficient for light-weight cross-camera multitarget tracking tasks. Notably, the fact that such low-performance hardware achieved near real-time analysis highlights CityTrac's superior scalability compared to the other two strategies. The results demonstrate the system's ability to efficiently meet real-time performance and scalability requirements, even when operating with limited hardware resources.

To further illustrate the differences in scheduling strategies and to reveal the underlying reasons for the observed performance gaps, we created a Gantt chart comparing the three strategies. Although our actual experiments used only three low-cost workstations as edge servers (due to the maximum concurrency of three edges tracking Target 260), we assume that each Field is managed by its own edge server in this visualization. For instance, CityTrac 1 refers to the edge server managing the video data and generating scheduling decisions for Field 1 under the CityTrac strategy.

Fig. 14 shows the Gantt Chart. For resource-limited edge devices, continuously queuing an excessive volume of incoming tasks without allowing the earlier ones to complete leads to severe task backlog and ultimately degrades overall system efficiency. The Gantt chart clearly shows that:

Spatula experiences significant task accumulation in multiple servers. For example, Spatula 6, 5, 4, and 1 face continuous task assignments over multiple tracking rounds. Field 4, in particular, receives tasks for six consecutive rounds, resulting in a substantial backlog that slows the entire system and contributes to Spatula's longest total processing time.

Anveshak performs somewhat better than Spatula but still encounters a four-round backlog in Fields 5, 4, and 1. This repeated queuing also creates unnecessary delays.

CityTrac exhibits the least task backlog, with at most two consecutive rounds of task queuing. This minimal accumulation of tasks explains why CityTrac achieves the shortest processing time. By avoiding sustained backlog, CityTrac ensures more efficient utilization of each edge server's limited resources and maintains near real-time processing capabilities even under constrained conditions.

VII. LIMITATION AND FUTURE WORKS

This superiority in tracking objects demonstrates CityTrac's high feasibility for practical deployment. However, it remains susceptible to the following limitations that will need to be addressed in the future.

First, it is possible to optimize the scheduling and selection of cameras at a finer level. For example, surveillance cameras within a given *Zone* may capture a target at different times (albeit with a small difference). As a result, the optimal activation time for different cameras within a particular *Zone* may vary.

A second consideration is the uneven distribution of traffic in different *Zones*. A *Field* located in a busy area will likely carry much more traffic than a *Field* located in a peripheral area. Thus, if the resource distribution strategies are tailored according to the traffic flow in different *Fields*, the resources can be used more efficiently.

Third, the current system assumes reliable wired connections between cameras and edge devices, which may not always hold in real-world deployments. Dynamic network changes, such as fluctuating bandwidth, latency variations, and potential connection losses, can adversely affect the performance of CityTrac. For cameras, the tracking strategy is based on *Field* and *Zone* configurations, allowing the system to tolerate the loss of individual camera connections or reduced transmission rates without significantly impacting overall tracking accuracy. This requires timely updates to the status of affected cameras to ensure that the total number of active cameras continues to meet the accuracy requirements of the tracking tasks. Additionally, introducing new cameras can be managed by assigning an initial reliability probability, which is updated in real-time based on subsequent accuracy statistics.

Regarding edge devices, the loss of a single edge server disrupts the synchronization of monitoring data across *Fields*. This scenario can be divided into two cases: 1) the edge device

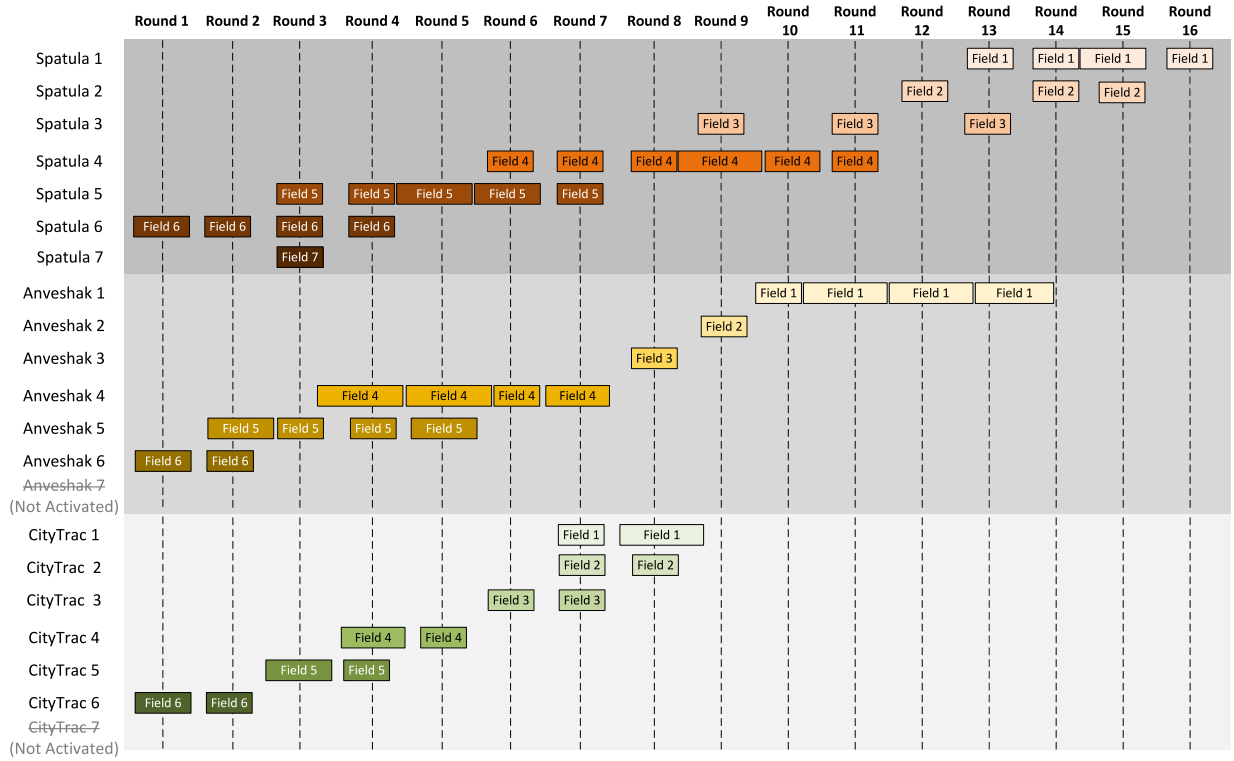

Fig. 14. Gantt chart of task scheduling for different strategies.

remains operational but loses connectivity with other edge servers and 2) the edge server experiences a complete outage. In the first case, the *Field* can continue local processing, but interfield information synchronization is hindered, which can be mitigated by implementing reconnection strategies based on current network conditions. In the second case, the monitoring data from the affected *Field* is entirely lost. To reduce the impact of such outages, future work could explore recording Spatial-Temporal Correlations not only with adjacent edge servers but also with nonadjacent ones, enhancing system robustness. However, this approach would significantly increase the complexity of the scheduling algorithms.

Fourth, high target density scenarios present substantial challenges due to the limited computational capabilities of individual edge devices, especially in supporting large-scale deployments cost-effectively. In high-density environments, the system must address two main aspects: 1) when task queues begin to backlog, tasks should be prioritized based on factors, such as arrival time, importance, and estimated processing duration to optimize the order of task handling and 2) if surrounding edge devices have available capacity, tasks can be offloaded to these less busy servers to enhance resource utilization. This approach must account for transmission delays, processing delays, and anticipated traffic flows to ensure efficient task distribution. These aspects represent promising directions for future research to improve system performance under high-load conditions.

Lastly, when the task volume exceeds the capacity of the Edge, it is also important to prioritize which tasks can ensure that most tasks can be completed in real-time. By integrating adaptive resource management strategies, such as dynamic load balancing, selective task scheduling, or hybrid edge-cloud offloading—future solutions can improve system resilience and efficiency.

VIII. Conclusion

This article presents a summary of the issues that hinder the efficient tracking of targets in hyperscale camera networks. By carefully handling the redundancy and over-activation in existing tracking scheduling strategies, a new tracking framework, CityTrac, is proposed. It accommodates the tracking tasks in a finer grain and profiles the tempo-spatial correlation for target movement prediction, thereby reducing the workload on the edge servers. Compared with the SoTA tracking strategy, CityTrac requires 90 percent fewer computing resources for tracking the same number of targets at a similar level of accuracy.

References

[1] E. Tavanti, P. Nepa, R. Gabbrielli, and M. Pirozzi, "Review on systems combining computational vision and radio frequency identification," *IEEE Internet Things J.*, vol. 12, no. 2, pp. 1291–1319, Jan. 2025.

[2] T. Wang et al., "Energy-efficient relay tracking with multiple mobile camera sensors," *Comput. Netw.*, vol. 133, pp. 130–140, Mar. 2018. [Online]. Available: https://www.sciencedirect.com/science/article/pii/S1389128618300045

[3] C. Liu et al., "City-scale multi-camera vehicle tracking guided by crossroad zones," in *Proc. CVPR Workshop*, 2021, pp. 1–9.

[4] N. M. Chung et al., "Multi-camera multi-vehicle tracking with domain generalization and contextual constraints," in *Proc. IEEE/CVF Conf. Comput. Vis. Pattern Recognit.*, 2022, pp. 3327–3337.

[5] S. U. Khan et al., "Efficient person reidentification for IoT-assisted cyber–physical systems," *IEEE Internet Things J.*, vol. 10, no. 21, pp. 18695–18707, Nov. 2023.

[6] G. Ananthanarayanan et al., "Real-time video analytics: The killer app for edge computing," *Computer*, vol. 50, no. 10, pp. 58–67, Oct. 2017.

[7] X. Zhu and M. Zhou, "Multiobjective optimized deployment of edge-enabled wireless visual sensor networks for target coverage," *IEEE Internet Things J.*, vol. 10, no. 17, pp. 15325–15337, Sep. 2023.

[8] B. Huang, J. Ju, Y. Shu, and Y. Wang, "Simultaneously recovering multi-person meshes and multi-view cameras with human semantics," *IEEE Trans. Circuits Syst. Video Technol.*, vol. 34, no. 6, pp. 4229–4242, Jun. 2024.

[9] H. Du, L. Chen, J. Qian, J. Hou, T. Jung, and X.-Y. Li, "PatronuS: A system for privacy-preserving cloud video surveillance," *IEEE J. Sel. Areas Commun.*, vol. 38, no. 6, pp. 1252–1261, Jun. 2020.

[10] J. Yu, H. Chen, K. Wu, T. Zhou, Z. Cai, and F. Liu, "Centipede: Leveraging the distributed camera crowd for cooperative video data storage," *IEEE Internet Things J.*, vol. 8, no. 22, pp. 16498–16509, Nov. 2021.

[11] X. Wei, M. Zhou, H. Wang, H. Yang, L. Chen, and S. Kwong, "Recent advances in rate control: From optimization to implementation and beyond," *IEEE Trans. Circuits Syst. Video Technol.*, vol. 34, no. 1, pp. 17–33, Jan. 2024.

[12] C. Li et al., "Hybrid-MVS: Robust multi-view reconstruction with hybrid optimization of visual and depth cues," *IEEE Trans. Circuits Syst. Video Technol.*, vol. 33, no. 12, pp. 7630–7644, Dec. 2023.

[13] (BriefCam, Newton, MA, USA). *Transforming Video into Actionable Intelligence*. 2022. [Online]. Available: https://www.briefcam.com/

[14] E. Luna, J. C. SanMiguel, J. M. Martínez, and P. Carballeira, "Graph neural networks for cross-camera data association," *IEEE Trans. Circuits Syst. Video Technol.*, vol. 33, no. 2, pp. 589–601, Feb. 2023.

[15] J. Dai and N. Liu, "The data platform for large-scale video surveillance systems," in *Proc. Int. Conf. Appl. Techn. Cyber Security Intell.*, 2020, pp. 1556–1561.

[16] N. A. Tu, T. Huynh-The, K.-S. Wong, M. F. Demirci, and Y.-K. Lee, "Toward efficient and intelligent video analytics with visual privacy protection for large-scale surveillance," *J. Supercomput.*, vol. 77, no. 12, pp. 14374–14404, 2021.

[17] W. Feng, D. Ji, Y. Wang, S. Chang, H. Ren, and W. Gan, "Challenges on large scale surveillance video analysis," in *Proc. IEEE Conf. Comput. Vis. Pattern Recognit. Workshops*, 2018, pp. 69–76.

[18] A. C. Nazare Jr. and W. R. Schwartz, "A scalable and flexible framework for smart video surveillance," *Comput. Vis. Image Understand.*, vol. 144, pp. 258–275, Mar. 2016.

[19] B. N. Subudhi, D. K. Rout, and A. Ghosh, "Big data analytics for video surveillance," *Multimedia Tools Appl.*, vol. 78, no. 18, pp. 26129–26162, 2019.

[20] M. Hu, L. Zhuang, D. Wu, Z. Huang, and H. Hu, "Edge computing for real-time video stream analytics," in *Encyclopedia Wireless Networks*, X. (Sherman) Shen, X. Lin, and K. Zhang, Eds., Cham, Switzerland: Springer Int. Publ., 2020, pp. 365–369, doi: 10.1007/978-3-319-78262-1_275.

[21] F. Yang, S. Odashima, S. Masui, I. Kusajima, S. Yamao, and S. Jiang, "Enhancing multi-camera gymnast tracking through domain knowledge integration," *IEEE Trans. Circuits Syst. Video Technol.*, vol. 34, no. 12, pp. 13386–13400, Dec. 2024.

[22] Z. Zhang, Z. Liu, Q. Jiang, J. Chen, and H. An, "RDMA-based Apache storm for high-performance stream data processing," *Int. J. Parallel Program.*, vol. 49, no. 5, pp. 671–684, 2021.

[23] C. Patrascu, A. Cosma, and E. Radoi, "Scalable deployments for real-time AI video stream processing," in *Proc. 21st RoEduNet Conf., Netw. Educ. Res. (RoEduNet)*, 2022, pp. 1–6.

[24] Z. Tang et al., "CityFlow: A city-scale benchmark for multi-target multi-camera vehicle tracking and re-identification," in *Proc. CVPR*, 2019, pp. 8797–8806.

[25] P. Hu, J. Im, Z. Asgar, and S. Katti, "Starfish: Resilient image compression for AIoT cameras," in *Proc. 18th Sensys*, 2020, pp. 395–408.

[26] M. Zhang et al., "Deep learning in the era of edge computing: Challenges and opportunities," *Fog Comput., Theory Pract.*, vol. 3, pp. 67–78, May 2020.

[27] J. Liu, M. Zhou, and M. Xiao, "Deformable convolution dense network for compressed video quality enhancement," in *Proc. ICASSP*, 2022, pp. 1930–1934.

[28] M. Azure. "Cloud computing services." 2022. [Online]. Available: https://azure.microsoft.com

[29] J. Lin, A. Huang, T. Zhao, X. Wang, and S. Kwong, "λ-domain VVC rate control based on Nash equilibrium," *IEEE Trans. Circuits Syst. Video Technol.*, vol. 33, no. 7, pp. 3477–3487, Jul. 2023.

[30] K. Nalaie, "Accelerating multi-target visual tracking on smart edge devices," Ph.D. dissertation, Dept. Comput. Softw., McMaster Univ., Hamilton, ON, Canada, 2023.

[31] Z. Wu, L. Li, and Z. Gao, "A fast and efficient vehicle multi-target tracking algorithm for edge computing devices," in *Proc. Int. Conf. Nat. Comput., Fuzzy Syst. Knowl. Discov.*, 2022, pp. 276–285.

[32] H. Sun, Y. Chen, A. Aved, and E. Blasch, "Collaborative multi-object tracking as an edge service using transfer learning," in *Proc. IEEE 22nd Int. Conf. High Perform. Comput. Commun., IEEE 18th Int. Conf. Smart City, IEEE 6th Int. Conf. Data Sci. Syst. (HPCC/SmartCity/DSS)*, 2020, pp. 1112–1119.

[33] S. Li et al., "Towards high-accuracy and real-time two-stage small object detection on FPGA," *IEEE Trans. Circuits Syst. Video Technol.*, vol. 34, no. 9, pp. 8053–8066, Sep. 2024.

[34] X. Zeng, B. Fang, H. Shen, and M. Zhang, "Distream: Scaling live video analytics with workload-adaptive distributed edge intelligence," in *Proc. 18th SenSys*, 2020, pp. 409–421.

[35] A. Poms, W. Crichton, P. Hanrahan, and K. Fatahalian, "Scanner: Efficient video analysis at scale," *ACM Trans. Graph.*, vol. 37, no. 4, pp. 1–13, 2018.

[36] K. Nalaie, R. Xu, and R. Zheng, "DeepScale: Online frame size adaptation for multi-object tracking on smart cameras and edge servers," in *Proc. IEEE/ACM 7th Int. Conf. Internet Things Design Implement. (IoTDI)*, 2022, pp. 67–79.

[37] S. V. Ganesh, Y. Wu, G. Liu, R. Kompella, and L. Liu, "Fast and resource-efficient object tracking on edge devices: A measurement study," 2023, *arXiv:2309.02666*.

[38] H. F. Yang, J. Cai, C. Liu, R. Ke, and Y. Wang, "Cooperative multi-camera vehicle tracking and traffic surveillance with edge artificial intelligence and representation learning," *Transp. Res. Part C, Emerg. Technol.*, vol. 148, Mar. 2023, Art. no. 103982.

[39] Y. Ma, L. Xu, Y. Zhang, T. Zhang, and X. Luo, "Steganalysis feature selection with multidimensional evaluation and dynamic threshold allocation," *IEEE Trans. Circuits Syst. Video Technol.*, vol. 34, no. 3, pp. 1954–1969, Mar. 2024.

[40] S. Jain et al., "Spatula: Efficient cross-camera video analytics on large camera networks," in *Proc. SEC*, 2020, pp. 110–124.

[41] A. Khochare, A. Krishnan, and Y. Simmhan, "A scalable platform for distributed object tracking across a many-camera network," *IEEE Trans. Parallel Distrib. Syst.*, vol. 32, no. 6, pp. 1479–1493, Jun. 2021.

[42] Y. Zheng, L. Zhang, X. Xie, and W.-Y. Ma, "Mining interesting locations and travel sequences from GPS trajectories," in *Proc. 18th WWW*, 2009, pp. 791–800.

[43] Y. Zheng, Q. Li, Y. Chen, X. Xie, and W.-Y. Ma, "Understanding mobility based on GPS data," in *Proc. UbiComp*, 2008, pp. 312–321.

[44] Y. Zheng, X. Xie, and W.-Y. Ma, "Geolife: A collaborative social networking service among user, location and trajectory," *IEEE Data Eng. Bull.*, vol. 33, no. 2, pp. 32–39, Jun. 2010.

[45] J. Yu, T. Zhou, Z. Cai, and W. Kuang, "Tracking targets in hyper-scale cameras using movement predication," in *Proc. IEEE Int. Conf. Acoust., Speech Signal Process. (ICASSP)*, 2023, pp. 1–5.

[46] X. Liu, P. Ghosh, O. Ulutan, B. Manjunath, K. S. Chan, and R. Govindan, "Caesar: Cross-camera complex activity recognition," in *Proc. Sensys*, 2019, pp. 232–244.

[47] Y. He, X. Wei, X. Hong, W. Shi, and Y. Gong, "Multi-target multi-camera tracking by tracklet-to-target assignment," *IEEE Trans. Image Process.*, vol. 29, pp. 5191–5205, 2020.

[48] S. Paul et al., "Enhancing video analytics accuracy via real-time automated camera parameter tuning," in *Proc. 20th ACM Conf. Embed. Netw. Sensor Syst.*, 2022, pp. 291–304.

[49] H. Liu et al., "Vi-Fi: Associating moving subjects across vision and wireless sensors," in *Proc. 21st ACM/IEEE Int. Conf. Inf. Process. Sensor Netw. (IPSN)*, 2022, pp. 208–219.

[50] S. You, H. Yao, and C. Xu, "Multi-target multi-camera tracking with optical-based pose association," *IEEE Trans. Circuits Syst. Video Technol.*, vol. 31, no. 8, pp. 3105–3117, Aug. 2021.

[51] B. Han, S.-W. Joo, and L. S. Davis, "Multi-camera tracking with adaptive resource allocation," *Int. J. Comput. Vis.*, vol. 91, no. 1, pp. 45–58, 2011.

[52] Y.-G. Lee, Z. Tang, and J.-N. Hwang, "Online-learning-based human tracking across non-overlapping cameras," *IEEE Trans. Circuits Syst. Video Technol.*, vol. 28, no. 10, pp. 2870–2883, Oct. 2018.

[53] M. Liu, S. Yang, W. Chomsin, and W. Du, "Real-time tracking of smartwatch orientation and location by multitask learning," in *Proc. 20th ACM Conf. Embed. Netw. Sensor Syst.*, 2022, pp. 120–133.

[54] S. Yang, D. Shi, Y. Peng, S. Yang, B. Zhang, and W. Yang, "Placement optimization for UAV-enabled wireless networks with multi-hop backhauls in urban environments," in *Proc. 21st ACM/IEEE Int. Conf. Inf. Process. Sensor Netw. (IPSN)*, 2022, pp. 54–66.

Jiaping Yu received the M.S. degree in information science and technology from Temple University, Philadelphia, PA, USA, in 2017, and the Ph.D. degree in computer science and technology from the National University of Defense Technology, Changsha, China, in 2023.

He is currently a Lecturer with the College of Computer Science and Technology, Ocean University of China, Qingdao, China. His current research interests include distributed systems, blockchain, and edge computing.

Hongjia Wu received the B.S. degree from Liaoning Technical University, Fuxin, China, in 2016, the M.S. degree from Guangxi University, Nanning, China, in 2018, and the Ph.D. degree in computer science and technology from the National University of Defense Technology, Changsha, China, in 2023.

She is currently a Postdoctoral Fellow with the Department of Mathematics and Information Technology, Education University of Hong Kong, Hong Kong, SAR, China. She was a Visiting Student with Singapore University of Technology and Design, Singapore. Her research interests include computation offloading, game theory, resource allocation, and dispersed computing.

Tongqing Zhou received the bachelor's, master's, and Ph.D. degrees in computer science and technology from the National University of Defense Technology (NUDT), Changsha, China, in 2012, 2014, and 2018, respectively.

He is currently a Postdoctoral Fellow with the College of Computer, NUDT. His main research interests include ubiquitous computing, mobile sensing, and data privacy.

Zhiping Cai (Member, IEEE) received the B.S., M.S., and Ph.D. degrees in computer science and technology from the National University of Defense Technology (NUDT), Changsha, China, in 1996, 2002, and 2005, respectively.

He is currently a Full Professor with the College of Computer, NUDT. His current research interests include network security and big data.

Prof. Cai's Ph.D. dissertation received the Outstanding Dissertation Award of the Chinese PLA. He is a Senior Member of the China Computer Federation.

Wenyuan Kuang received the B.S. degree in computer science and technology from the National University of Defense Technology, Changsha, China, in 1996, and the Ph.D. degree in computer science and technology from Tsinghua University, Beijing, China, in 2008.

From 2014 to 2019, he was a Manager and a Researcher with Forsun Technology, 360 Enterprise Security Group, and Qianxin Group, Beijing. He has applied for five Chinese patents and two U.S. patents. His recent research interests include threat intelligence and protocol security.

Hui Xia received the Ph.D. degree in computer science from the School of Computer Science and Technology, Shandong University, Jinan, China, in June 2013.

From July 2013 to June 2020, he was a Lecturer and an Associate Professor with the College of Computer Science and Technology, Qingdao University, Qingdao, China. Since July 2020, he has been a Full Professor and a Ph.D. Supervisor with the College of Computer Science and Technology, Ocean University of China, Qingdao. He was a Visiting Scholar with the Department of Computer Science, The George Washington University, Washington, DC, USA, from 2017 to 2018. He has published over 50 scientific papers in IEEE TRANSACTIONS ON KNOWLEDGE AND DATA ENGINEERING, *IEEE/ACM Transactions on Networking*, IEEE TRANSACTIONS ON INDUSTRIAL INFORMATICS, IEEE TRANSACTIONS ON VEHICULAR TECHNOLOGY, ComNet, JSA, IEEE INTERNET OF THINGS JOURNAL, IEEE TRANSACTIONS ON NETWORK SCIENCE AND ENGINEERING, IEEE TRANSACTIONS ON SUSTAINABLE COMPUTING, IEEE TRANSACTIONS ON NETWORK AND SERVICE MANAGEMENT, IEEE TRANSACTIONS ON CLOUD COMPUTING, INFOCOM, NDSS, and other international high-grade journals and conferences. His current research interests focus on computer networks, the IoT security, artificial intelligence security, edge computing, crowdsourcing computing, federated learning, and privacy protection.

Prof. Xia is a member of ACM.