PreActo: Efficient Cross-Camera Object Tracking System in Video Analytics Edge Computing

Thanh-Tung Nguyen
School of Computing, KAIST
Daejeon, South Korea
tungnt@kaist.ac.kr

Si Young Jang*
School of Computing, KAIST
Daejeon, South Korea
sy.jang@kaist.ac.kr

Boyan Kostadinov†
School of Computing, KAIST
Daejeon, South Korea
boyanyk@kaist.ac.kr

Dongman Lee‡
School of Computing, KAIST
Daejeon, South Korea
dlee@kaist.ac.kr

2023 IEEE International Conference on Pervasive Computing and Communications (PerCom) | 978-1-6654-5378-3/23/$31.00 ©2023 IEEE | DOI: 10.1109/PERCOM56429.2023.10099298

***Abstract*—Cross-camera real-time object tracking is one of the important, yet challenging applications of video analytics in edge computing environments. To provide accurate and efficient real-time tracking, a tracking target's future movements need to be predicted. Particularly, the destination camera and travel time of the target object are to be identified so that tracking duties can be handover-ed seamlessly. In this paper, we propose a collaborative cross-camera tracking system, called *PreActo*, with two key features: (1) ResNet-based trajectory learning to exploit the rich spatio-temporal information embedded within objects' moving patterns, which has not been utilized by the existing literature, and (2) collaboration between the edge server and the edge device for real-time trajectory prediction and tracking handover. To prove the validity of our proposed system, we evaluate PreActo on a video dataset leveraging real-world trajectories. Evaluation results show that the proposed system reduces up to $7\times$ the number of processed frames for handover, with $2\times$ lower latency while providing $1.5\times$ tracking precision improvement compared to the state-of-the-art.**

***Index Terms*—Edge Computing, Machine Learning, Object Tracking, Trajectory learning**

I. Introduction

With an ever-increasing number of cameras deployed in urban cities [1], real-time video analytics is definitely an important application at the network edge. It provides various video analytic services such as object detection [2], [3], [4], re-identification [5], [6] and cross-camera object tracking [7], [8], [9], [10], [11]. Among those, tracking objects (e.g., humans or vehicles) across multiple geo-distributed video feeds is challenging as it requires sophisticated edge server-camera collaboration and camera coordination. Regarding the former, it can be largely divided into two categories: edge server-driven approach [12], [13], where the server analyzes video feeds collected from all geo-distributed cameras; and edge device-driven approach [14], where powerful smart AI-enabled cameras filter out large portions of the videos before uploading to the server. Both of these approaches are prone to creating a bottleneck at either the edge server or the cameras, leading to the failure of real-time tracking tasks. Thus, it is essential that the computation workload is effectively divided via sophisticated server-camera collaboration. Moreover, since tracking involves multiple cameras, camera coordination is equally important. Specifically, when a tracking object is leaving the field-of-view (FOV) of *camera A*, a destination *camera B* and the time at which the object arrives at *camera B* need to be identified so that the handover of the tracking duty from *A* to *B* can be done seamlessly. A simple approach, mentioned as a baseline in [9], is to have all cameras search their incoming video feeds until the object is found. However, this brute force approach results in low efficiency and scalability. A more sophisticated line of approaches considers the spatio-temporal correlations [15] among the cameras. The underlying insight is that there exist physical correlations between certain pairs of cameras that can be leveraged to improve tracking efficiency [16]. For instance, Spatula [9] analyzes the distributions of past entries and exits to determine the next camera candidates and arrival time. Kestrel [10] and WatchDog [11] exploit temporal filters obtained from analyzing historical data to determine which frames to be processed during real-time tracking. We argue that existing works have not utilized the rich spatio-temporal information naturally embedded within an object's mobility patterns, which can reveal its moving speed and the direction it is heading towards.

In this paper, we propose *PreActo*, an efficient cross-camera object tracking system. *PreActo* utilizes a lightweight 1-D ResNet model [17] extracting spatio-temporal features from object trajectories to predict the camera target (*where*) and the transition time (*when*) to arrive at the target. *PreActo* was named after **pre**dict and **react**, meaning our system with an ability to make accurate predictions can efficiently react to various tracking events. *PreActo* can be divided into 2 phases. **(1)** Firstly, during the ***offline phase***, deep spatial & temporal trajectory learning is performed at the edge server. The training data is composed of trajectories of objects within the tracking camera's FOV. **(2)** Secondly, during the ***online phase***, to avoid computation bottlenecks, the workload is evenly distributed between the server and cameras. Instead of streaming raw footage to the server, each camera runs an object detection model [2] and only sends cropped bounding boxes to the server for re-identification (re-ID) [5], [18], which is a computationally demanding task. The coordinates of each object within the camera's FOV are recorded as a time series. Once the object departs from its FOV, the camera inputs the recorded time series into the pretrained models to predict the next camera candidates and the transition time; and then

* Currently affiliated with Nokia Bell Labs, Cambridge, UK.
† Currently affiliated with Netlight, Munich, Germany.
‡ Corresponding Author

notifies the candidate with the highest confidence score of the time the target may arrive. At the specified time, the target candidate requests the server for re-ID to identify if the object is indeed found. Otherwise, it handovers to the second candidate in the list in a similar manner, and so on.

To the best of our knowledge, there exist no large-scale video datasets publicly available[1]. Instead, we leverage an 18-camera dataset generated by Carla [25] with trace-driven human mobility patterns and camera locations reproducing a real-world environment, which is described in detail in Subsection VI-A. This approach is similar to that of [11], [9]. Our results show that as a whole, the system ensures that tracking is only performed on necessary cameras at the correct frames, leading to 7× and 2× improvement in terms of the number of processed frames and latency, respectively. The key contributions of our work are as follows:

- We leverage an FOV-trajectory-based deep-learning mechanism using lightweight ResNet models to predict when and where to activate edge cameras. The results show that a target object is mostly found to be the first and second candidate cameras in 93% and 6% of the cases, respectively.
- We evaluate our system using a video dataset reproducing a real-world environment (particularly, Peking University's campus). The proposed system doubles the tracking application's performance in terms of on-time tracking result delivery compared to the state-of-the-art.

II. Related Work

A. Object Trajectory Prediction

Object trajectory prediction can be considered a time-series forecasting task. In [26], Alahi *et al.* introduced the early work for an influential type of model, called Social LSTM, which leveraged the LSTM model with a Social Pooling layer to learn and predict human trajectories. Since then, several similar approaches using increasingly more complex learning models [27], [28], [29], [30] have been proposed. It can be seen that the common approach from these works is using large recurrent models (e.g., encoder-decoder LTSM, Transformer) with complex mechanisms including multi-head attention and multi-modal data embedding in order to predict the trajectories of objects. We argue that these models are not suitable for the scenario under consideration due to the following reasons. First and foremost, to predict the detailed trajectories of objects, the computational complexity and resource demands of these models are beyond what edge devices can handle, which is crucial to real-time tracking. Secondly, these models focus on predicting only at most dozens of points into the future. Once produced, the output could be leveraged as input for further prediction repeatedly until the tracked object is *seen by the next camera*. However, this would require several more times computation resources and cause unbearable delay. Thus, we argue that the trajectory prediction task can be reduced to only predicting *where* and *when* a tracked object shows up next in the camera network, which would vastly reduce the size of the prediction model and its time consumption, allowing the task to be achieved in real-time. We propose to use lightweight ResNet-based models that can be operated on small embedded devices but still can accurately predict the next cameras as well as the transition time to reach those cameras. The details of our models are given in Subsection IV-C.

[1]DukeMTMC [19] and SLP [20] are not publicly available. Virat [21] and LPW [22] are small-scaled. WNMF's videos are too short. Re-ID datasets including Market1501 [23] and RPIField [24] only contain cropped images.

B. Cross-camera tracking systems

WatchDog [11] builds a cross-camera vehicle tracking system with powerful geo-distributed cameras. This work manages to run tracking in real-time by running a lightweight and less re-ID technique for a crowded intersection and delaying the heavy and more accurate re-ID operations until there is an empty intersection to correctly identify the vehicle. Kestrel [10] tracks suspicious vehicles in the road camera network by utilizing cheap visual features for associating objects seen in different cameras (re-ID). Both WatchDog and Kestrel present a solid method for optimizing the Re-ID pipeline. However, their tracking algorithms desire much improvement. Firstly, they rely only on the moving directions of the objects to infer to next camera target. In reality, this may not be the case, especially for dense camera networks. Secondly, in terms of transition time from one camera to another, they both set fixed lower and upper limits based on historical data without considering each individual object's characteristics. We argue that the two above-mentioned aspects are the most important in cross-camera tracking and they could be better determined by analyzing the mobility patterns of objects. Caesar [8] proposes a cross-camera activity recognition system that requires tracking through multiple cameras. However, next camera prediction and travel time estimation were not discussed. The authors instead only cite Kestrel [10] as their method for cross-camera tracking.

Spatula [9] analyzes a multi-camera object tracking dataset to generate spatio-temporal correlations between cameras, which depends on the statistical distributions of exit-entry samples. They then exploit the spatio-temporal correlations through a statistical approach to reduce the search space and devise a forward and backward search upon missing targets. However, we believe this approach has the following inherent drawbacks. Firstly, the performance of Spatula heavily depends on manually-set temporal and spatial thresholds to filter the search space. It can be found in the original paper that there are noticeable discrepancies between the results of different thresholds. Secondly, Spatula falls short on tracking precision due to wrong travel time predictions as they do not consider the mobility patterns of the object under tracking. For instance, given two objects leaving camera A for camera B with different speeds and mobility patterns, Spatula would behave in the exact same way for both cases. This weakness can also be seen in Kestrel and WatchDog.

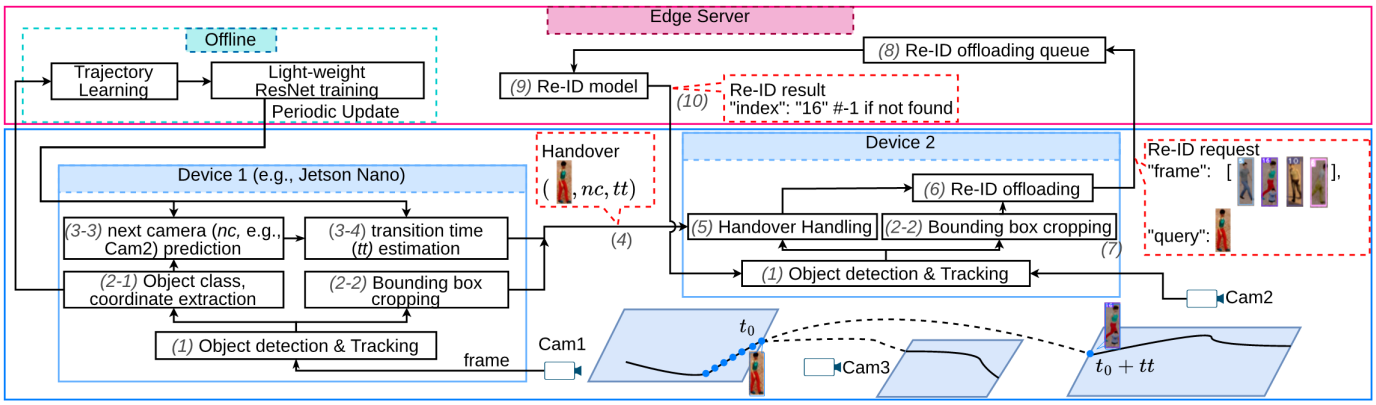


Fig. 1: The overall architecture of our proposed cross-camera object tracking system, *PreActo*. When an object disappears, Device 1 uses the object's extracted trajectory to predict the next target (*nc*) and the transition time (*tt*). This information and the last seen image of the object are forwarded to the predicted next target, Device 2.

In this paper, we propose a tracking system that leverages deep learning models to predict the destination and the arrival time of an object accurately. We will further discuss our advantages over Kestrel, WatchDog, and Kestrel along with experiment results in Section VI.

III. System Overview

Fig. 1 shows the overall architecture of PreActo to support cross-camera object tracking, which essentially associates objects seen in one camera to ones in another. The challenge is that there are many cameras, each with numerous frames, which makes the spatio-temporal search space become exponentially larger as time goes. It is impossible to achieve real-time cross-camera tracking without efficiently establishing a spatio-temporal model across the camera network. We roughly divided PreActo's operation into two phases *offline* and *online*.

A. Offline Phase

To establish spatio-temporal relationships amongst cameras, the edge server uses a historical video dataset recorded from multiple edge cameras during the *offline phase* and conducts the following operations:

1) Firstly, in order to train models that can predict the destination that a target object is traveling to and approximate their arrival time, we first need to construct and label the traces. Each labeled trace consists of a sequence of coordinates of bounding box centers (or **Points**), denoted as (x, y), the next destination camera, denoted as nc, on which a target object appears next, and its transition time, denoted as tt, to reach the destination (Subsection IV-A).
2) Secondly, we train two lightweight models to predict the destination camera and the object's transition time. We devise a lightweight 1-D convolutional ResNet-based classification model, for *next camera prediction* and a regression version of the same model for *transition time estimation* using the same labeled trace (Subsection IV-C).

B. Online Phase

A multiple object tracking pipeline typically consists of two components: (1) an object detection model to find objects (e.g., people) in a frame, and a re-identification, or re-ID, model to determine if the objects-of-interest are still present in the current frame. Running the full pipeline is a challenge for low-capability edge devices. The reason is that these devices are low on available memory for computation, which cannot support both object detection and re-ID models. Even if they do, the latency is unbearable. For instance, we run an object detection model Yolo-v5 [4] and a re-ID model OSNet [31] on Nvidia Jetson Nano, which take 193 ms and 560 ms for a single frame, respectively. To achieve *real-time* cross-camera tracking with high precision, it is crucial that the workload is evenly divided between the server and edge cameras to avoid computation bottlenecks. Thus, during the *online* phase, the real-time operation of PreActo is designed as followings:

1) Each camera uses a state-of-the-art object detector to detect objects in the frame. Then, cropped bounding boxes of all object-of-interests are transmitted to the edge server for re-ID periodically (re-ID offload). The re-ID index is then returned to the camera, which starts a tracking operation (Subsection V-A).
2) After finding the target object, a camera tracks the object. Once its leaves the camera's FOV, the camera inputs a sequence of coordinates sampled from obtained traces into pretrained models for predicting a list of next camera candidates and the transition time, as shown in Fig. 1.
3) The next camera with the highest confidence score is chosen as the candidate camera, nc. A handover message, which includes a cropped image of the object-of-interest and the time it will take to arrive (tt), is sent to the candidate (e.g., Fig. 1's Device 2). After tt has passed, the candidate sends a non-periodic re-ID request to the server to confirm that the object-of-interest is found in its frame. Upon false predictions, a span-out recovery is performed to search for the target in the rest of the candidates (Subsection V-B).

In Section IV, we explain in detail how the proposed system realizes cross-camera handover by modeling the spatio-temporal relationship amongst video traces. In Section V, we take a closer look into the object tracking pipeline containing

object detection, re-ID, tracking, and handover operations.

IV. Spatial & Temporal Locality Driven Trajectory Learning

This section explains the extraction of learnable traces and the learning mechanisms for spatial and temporal knowledge between cameras. This information is then adopted by the cameras at run-time.

A. Trajectory Labeling

In PreActo, the center coordinates (x, y) of the bounding box containing each detected object in a camera's FOV are recorded as a sequence, or a time series, using frame indices as timestamps. We call this sequence a **Trajectory** (Tr) as it depicts a full set of coordinate points of a target object within a camera's FOV, from the first appearance until disappearance. The transition time is obtained by subtracting the time of the object's last detection in the current camera from the time of the first appearance in the next camera. We denote this **trajectory-label** as $(Tr_i^s : (nc, tt))$ where i, s, nc, and tt denote the current camera id, the sequence number, the next camera's id, and the transition time, respectively.

B. Sampling methods

At runtime, a camera has to decide the next camera candidates and the transition time to such candidates. Objects' trajectories in a camera's view are inherently different from one another in terms of shape and length due to mobility patterns (e.g., walking fast *v.* slowly). This implies that a camera may have to predict with traces (Tr), whose length may not be the same as the original trajectories from the training dataset. Such occasions may happen 1) when object detection fails for a number of frames, 2) re-ID request is finished late, 3) transition from the previous camera takes longer than expected, etc.

To handle such situations, we decide to sample all trajectories using three sampling methods: *last, evenly distributed, and overlapped sliding window* to guarantee a uniform length on all trajectories. We define a formal description of a trace as follows:

Every point is composed of two coordinates: x and y. Let $P_n = (x_n, y_n)$ denote a point at frame n, where $0 \leq x_n < frame_{width}$ and $0 \leq y_n < frame_{height}$. Then, a trace Tr of length n represents the following sequence:

$$Tr_n = (P_0, P_1, P_2, \dots P_{n-1})$$

The definitions of the sampling methods are as follows:

- *Last*: takes the last k points from a trace. This method aims to sample the most discriminative part of the trajectory, which is related to the exiting point of the trajectory before the object's disappearance.

$$Last(Tr, k) = (P_{n-(k)}, P_{n-(k-1)} \dots P_{n-1})$$

- *Evenly distributed (ED)*: takes k elements that are evenly spread out in the trace [32]. This prevents regions of higher sample density from contributing more points, causing skewed distribution in the data.

$$ED(Tr, k) = (P_0, P_g, \dots P_{g*(k-1)}, P_{g*k}), g = \lfloor n/k \rfloor$$

- *overlapped sliding window (SW-O)*: samples k elements, using a sliding window, starting from the first element and until reaching the end. Similar to the linear resampling technique [33], this returns all of the extracted samples.

$$SWO_0(Tr, k) = (P_0, P_1 \dots P_{k-1})$$
$$\dots$$
$$SWO_{n-(k+1)}(Tr, k) = (P_{n-(k-1)}, P_{n-(k-2)} \dots P_{n-1})$$

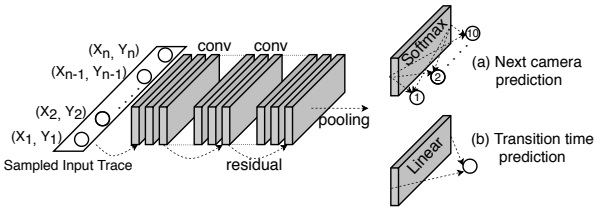


Fig. 2: An illustration of the ResNet network model used for training (a) next camera prediction and (b) transition time estimation.

C. Next camera and transition time predictions

As established in a significant body of works [34], [26], [30], [35], [36], there exists spatiotemporal information embedded within each trajectory that can be leveraged to predict the future movements of an object. However, as explained in Subsection II-A, unlike these works, we are not interested in short-term detailed predictions of future trajectory points, but in the next camera (*where*) and the time to arrive at that camera (*when)*, which enables seamless handovers of tracking duties from one camera to another. Moreover, running the above-mentioned models requires more computation capability than edge devices can provide. Thus, in this paper, we instead leverage a lightweight 1D ResNet model [17] as shown in Fig. 2. This model uses convolutional filters with 1D kernels to extract spatiotemporal features from time series data. It is constructed from three blocks, each of which has 3 convolutional filters. Between the blocks, a shortcut connection is added to aid the gradient flow. The number of channels of filters in the blocks is $\{64, 128, 128\}$.

Based on the visualization in [17] and our experiments, it can be seen that the low-level filters learn individual features such as relative positions of the trajectory with a camera FOV and walking stride. On the other hand, the high-level filters learn more complex features such as the general direction, walking style (straight lines *v.* contours), and average speed. Using such features allows PreActo to accurately make the following predictions:

Next camera prediction: To predict the next camera candidates (nc) for handover, we attach a softmax classification head to the ResNet backbone. This model is trained using categorical cross-entropy loss and an Adam optimizer with a learning rate of 0.001. The rate is reduced when on a plateau.

TABLE I: Performance (in frames per second) of object detection models on Jetson Nano.

Model	Efficientdet-d0[2]	YOLOv4 [3]	YOLOv5 [4]
FPS	5.17	2.35	5

Transition time estimation: To predict the time it takes for the pedestrian to travel, we replace the softmax layer with a fully-connected layer (dense). This model is trained with Root Mean Square Error (RMSE) and an Adam optimizer with a learning rate of 0.001.

It is worth noting that the results of using the same ResNet backbone for both predictions lead to significantly lower results than using two separate models. We consider that this is because they look for different sets of features such as relative positions and directions for camera prediction; and speed and patterns for time prediction. Moreover, each model is remarkably lightweight and this setting causes little computation overhead.

V. Runtime cross camera tracking

A. Detection & Tracking

Object tracking is comprised of object detection, re-ID, and continuous tracking of the correct re-ID index. This section describes how these three operations are synchronized to support timely and accurate frame processing on an edge camera, in collaboration with the edge server.

Object Detection (line 2 in Algorithm 1) Numerous object detection models (e.g., YOLO families [3], [4]) have been proposed to achieve high detection precision. However, many of them are too computationally expensive for embedded devices such as Nvidia Jetson Nano [37]. Table I shows benchmarking results of a variety of object detection models using our dataset (Section VI-A). The table depicts how many frames per second (FPS) each model can process as well as their accuracy. The accuracy score is calculated based on the number of correct detection relative to the ground truth in a frame. Based on the results, we leverage the state-of-the-art YOLOv5 [4] as it provides a good balance between FPS and accuracy. While its performance does not exceed 15 FPS, which is considered the real-time threshold, we do not need detection for every frame as the objects would display virtually no change in terms of positions in consecutive frames.

Offloading re-ID (line 3 in Algorithm 1) In a crowded scene, object detectors return multiple objects Obj_n of the same class (e.g. person). To perform state-of-the-art deep-learning based re-ID methods for all these objects is too computationally intensive for embedded devices as re-ID operations have to be iterating through cropped images, extract deep features [31] from each, and then match every set of features against the target's features. Thus, in our system, an edge camera sends requests with a gallery of cropped objects to a local edge server for re-ID, as shown in Fig. 1.

Lightweight Object Tracking (lines 6-7 in Algorithm 1) Once the exact target object index is verified, the tracker compares the following consecutive frames. The Tracker validates

Algorithm 1: Operation of PreActo on camera

```
   Input: videostream, sm
1  while frame f_i in videostream do
2      Obj_n ← DETECTION (f_i, det_threshold)
3      REQUEST-REID (Obj_n)
4      TRACKING:
5      if no missing then
6          TRACKER.UPDATE(Obj_Max(idx).coordinates)
7          Tr ⌢ Obj_Max(idx).coordinates
8      else
9          sm_Tr ← sm(Tr)
10         nc_list ← PREDICT-NC(sm_Tr)
11         tt_list ← PREDICT-TT(sm_Tr)
12         outOfViewTimer−=1
13         if outOfViewTimer < 0 then
14             nc ← nc_list.pop()
15             HANDOVER-MSG(dest = nc, HO_Obj,
                nc_list, tt_list)
16             DEACTIVATE
17     HANDOVER-HANDLING:
18     RECV-HANDOVER-MSG(HO_Obj, nc_list, tt_list)
19     REQUEST-REID (HO_Obj)
20     tt ← tt_list.pop()
21     WAIT(tt)
22     if notfound then
23         START(Recovery_timer)
24         if Recovery_timer < 0 then
25             nc ← nc_list.pop()
26             HANDOVER-MSG(dest = nc, HO_Obj,
                nc_list, tt_list)
```

two things. Firstly, it initializes new coordinate positions for a newly found target. Second, it checks if a target is still in the following frame. A target is regarded as missing if its previously registered coordinate is no longer in correlation with a couple of frames.

Camera prediction and transition time estimation (lines 10, 11 in Algorithm 1) Whenever a tracked object obj is missing from its view, a camera collects coordinates of P_i, the center of its detection bounding box, into a trace Tr_n of length l, where $0 \leq i < l$. This sequence only contains the coordinates starting from the time the object enters the view up to the current time slice. Then, next camera candidates nc_list and transition time tt_list are inferred using the models with sampling method sm and a minimum threshold. The minimum threshold is set to eliminate predictions that are too low for possible transition, such as 0.1%. For next camera prediction, a list of camera indices sorted by the confidence scores is returned. Time prediction returns the time it would take for the current target object to appear in each camera in the list.

B. Handover messaging & Span-out recovery

Once next camera candidates and transition time are predicted, the current camera sends the next candidate camera nc (the most probable one in the candidate list) a message which contains the target camera id, its transition time tt, and the predicted candidate list and deactivates itself as the

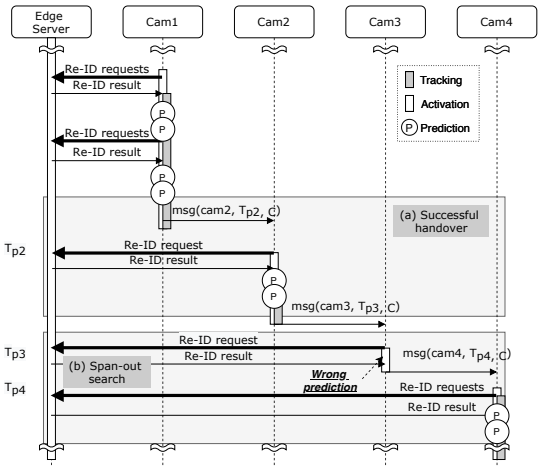


Fig. 3: An example of a span-out recovery sequence

target object leaves its view (lines 14-16 in Algorithm 1). A successful transition is shown in the middle part of Fig. 3.

In the case of a false prediction, we devise a *span-out recovery* where a falsely predicted camera messages the next probable camera in the candidate list to search for the target, instead of alerting all cameras to search at once (lines 22-26 in Algorithm 1. This is shown in the bottom part of Fig. 3 with a wrong prediction from $cam1$ to $cam2$ at T_{p2}. If $cam2$ fails to find the target, it does a span-out search by sending a handover message to the one with the next highest confidence in the list nc_list, $cam3$ at T_{p3}. The top 2 candidates cover 99% of test cases, as shown in Section VI. If the second one is also incorrect, then the third one is queried, and so forth.

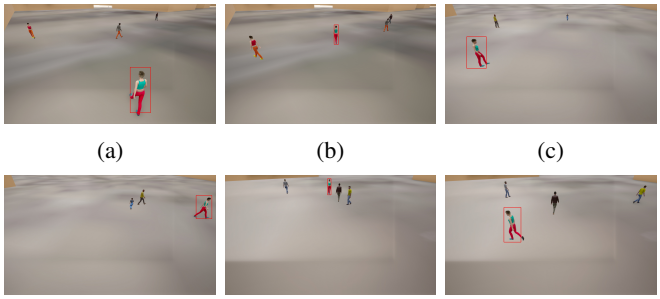
VI. Evaluation

A. Dataset

As stated in Section I, to validate our proposed approach, we leverage an 18-camera dataset generated by Carla [25] with trace-driven human mobility patterns and camera locations reproducing a real-world environment. This approach is similar to that of [11], [9]. The mobility patterns come from Geolife GPS trajectory dataset [38], which was collected over a period of five years, containing millions of GPS coordinates, located mainly in Beijing, China, and their recorded timestamps. We extract walking trajectories within the testing scenario. The reason we choose only pedestrian trajectories for our experiments is that we believe they are sufficient to prove the validity of our system. Even though our system can work for both humans and vehicles, human tracking is significantly more difficult. Vehicles, which generally follow specific lanes and change their directions gradually, predict the next camera and transition time can be done with better accuracy. We choose the southwestern section of Peking University (PU)'s Campus to create our experiment scenario, as shown in Fig. 4a. Using the map data downloaded from *openstreetmap.org*, we are able to construct a replica of a portion of PU's map. Fig. 4 shows that our simulated map is near-identical to the original map in terms of size and scale. The chosen area is

(a) *OpenStreetMap* snapshot (b) Simulated map

Fig. 4: Left figure (a) depicts a section of OpenStreetMap and Right figure (b) depicts the simulated area of (a) in Carla simulator. The sizes and scales of buildings and streets are preserved. The red cameras are ones that track a target as shown in Fig. 5.



(a) (b) (c)

(d) (e) (f)

Fig. 5: Snapshots taken from 3 cameras at 6 different frames. The tracked pedestrian follows the path Cam0 → Cam1 → Cam4: (a) and (b) - 60th and 141st frames of Cam0; (c) and (d) - 993rd and 1129th frames of Cam1; (e) and (f) - 1458th and 1619th frames of Cam4. Refer to the map in Fig. 4b)

most appropriate for our experiments as it contains a wide variety of patterns and routes from one point to another on the map, which can demonstrate the practicality of our proposed approach. Within the selected section, we extract 35881 GPS data points from the Geolife Dataset, which belongs to nearly 300 unique trajectories. By sampling the distributions of point-to-point distance, walking speeds, and oriented angles of the original trajectories, we further generate 2100 more trajectories for generating video data for training and testing the models.

To generate the video data, we employ Carla Simulator [25]. In particular, the constructed map is imported into Carla. We place 18 cameras around the map to cover important areas such as halls, buildings, and pathways. Each camera records 1280 x 720 videos at 15 FPS. Moreover, Carla pedestrian actors are generated so that at any point in time, there are 50-100 pedestrians walking inside the scenario. The length of the dataset used in our evaluation is 18 hours. Snapshots taken while tracking a pedestrian are shown Fig. 5. Here, the pedestrian's path involves Cam0, Cam1, and Cam4 (refer to the map in Fig. 4b).

B. On-device Memory Footprint & Latency

Unlike centralized servers, distributed edge devices are constantly under resource constraints. Thus, to realize our proposed scheme, it is crucial that the object detection model YOLOv5 [4], and our next camera and transition time prediction models are as light and efficient as possible. To prove the feasibility of the proposed system, we measure the memory footprint and latency it causes on a Jetson Nano device [37], which has a humble sum of only 4GB memory for both CPU and GPU calculations. On each device, we load the models into a Docker container. The measured results show that, on average, YOLOv5 inference takes almost 200 millisec process a frame (5 FPS). This makes processing every frame unable to meet real-time requirements. However, we believe it is possible to reduce the processing rate to 5 FPS as objects' appearances change minorly in consecutive frames. Furthermore, we deploy on each device two ResNet models to predict the next cameras and transition time. Each of these models is tiny as it takes up only 60MB of device memory. The weight is only 2MB, which allows periodical updates (as shown in Fig. 1) of models without incurring high communicational overheads. They also work blazing fast, taking 1.6 milliseconds to make predictions for the input of (30, 2)-shape.

C. Simulation setup

We run our simulation on a mini-server machine equipped with Intel(R) Xeon(R) Silver 4210R CPU, 188 GB RAM, and an Nvidia GeForce GTX 3090 GPU, using Python version 3.8.10. However, we want to note that on-device measurements in Subsection VI-B are taken into account to make sure the simulation is most realistic.

Model training: For training our custom ResNet models, we use Tensorflow 2.5.0. We train the models with 500 epochs for each camera. Re-ID is implemented using the Torchreid [39] framework version 1.2.5's mobile-oriented model, namely MobileNetv2_x1_0.

D. Approaches for Comparison

We compare the following handover scheme including ours:

- **Kestrel [10]**: uses the moving direction of an object to infer the next camera target. The transition time filter, similar to the original paper, is set as $[0.4t^{AB}, 1.6t^{AB}]$, where t^{AB} is the estimated time travel from $A \rightarrow B$. We set t^{AB} as the mean of instances in the historical data.
- **Watchdog [11]**: also uses the moving direction to infer the next target. The transition time filter, similar to the original paper, is set as $[t^{AB}_{min}, t^{AB}_{max}]$, where t^{AB}_{min} and t^{AB}_{max} are the shortest and longest time taken to move from $A \rightarrow B$ in the historical data.
- **Spatula [9]**: leverages historical trajectories to make predictions for the next camera candidates and the arrival time. For Spatula to function, a spatial-correlation threshold and a temporal-correlation threshold have to be manually set. For fairness, in this paper, we only report the spatial and temporal thresholds of 20% and 10%, respectively which yield the best results.

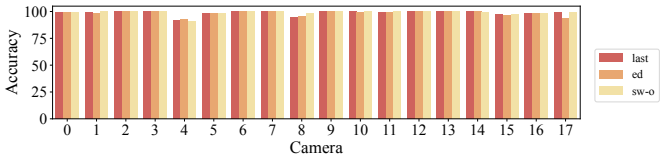


Fig. 6: Accuracy of *PreActo* for next camera prediction with models using different sampling schemes: *last, equally-distributed (eq)*, and *overlapping sliding-window (sw-o)*

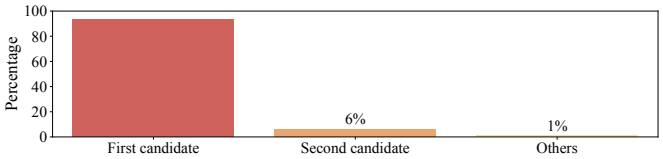


Fig. 7: Overall accuracy of *PreActo* for next camera prediction. Target objects are found to be the first and second candidate cameras in 93% and 6% of all cases.

- **PreActo**: leverages both temporal and spatiotemporal information embedded within each pedestrian trajectory to predict the next camera and the transition time accurately.

E. Simulation Results & Analysis

1) Accuracy of predictions: As *next camera prediction* is a crucial part of our system, we first verify the soundness of the models with different sampling methods (described in Section IV-B). As shown in Fig. 6, our lightweight ResNet models achieve high accuracy (over 90%) for all the sampling methods. The sampling methods *last* and *sw-o* show superior performance for most cameras. The reason is that *last* and *sw-o* use the last portion of the traces as parts of their training and inference data while *ed* uniformly samples the traces. The last portion usually contains the most information about the future direction of the pedestrians, which enables the models to make accurate predictions. We also applied traditional machine learning classification approaches including SVM and random forest. However, they fail to learn meaningful features from the trajectories and achieve low prediction accuracy (60% and 57% on average, respectively). During tracking, in the case where the first predictions are not correct, PreActo relies on the *Span-out Recovery* strategy described in Subsection V-B to recover. Here, the object will be looked for at the second, then the third candidates, and so on. Fig. 7 shows that the first and second candidates account for another 93% and 6% of the cases, which confirm the effectiveness of predictions made by PreActo. In nearly all cases, the object will be found with at most 2 tries.

Next, we evaluate the performances of the *transition time* prediction with the results shown in Fig. 8. During handover, early transition time prediction would result in the activation of cameras at non-essential frames, while late prediction would result in tracking accuracy drop and potentially losing the targets. In Fig. 8, the *x*-axis shows whether the target arrives early, on-time, or late compared to the ground truth labels, with negative values' signaling early arrival and positive values'

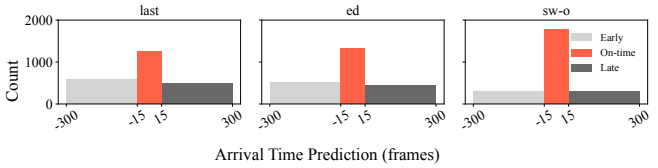


Fig. 8: Distribution of predicted transition time using different sampling schemes. X-axis shows frame differences from the ground truth.

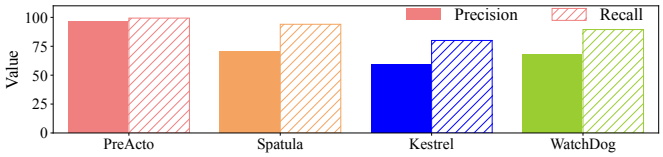


Fig. 9: Ratio of processed valid frames over all processed frames in all cameras (Precision) for different schemes.

showing late arrival. It can be seen that *sw-o* significantly outperforms the two other sampling methods and *last* shows the worst performance in this case. While the *last* portion of a trace contains decisive information about the future direction of a pedestrian, it may not have sufficient information about the walking speed of the pedestrian. With *ed*, on the other hand, the trace is sampled uniformly, which enables the model to learn latent features related to the walking speed. Moreover, with *sw-o*'s sliding windows, the training dataset is effectively extended allowing the prediction model to learn even more about the spatio-temporal relationship. However, during both training and testing, sampling traces with *sw-o* method causes significantly higher latency. Thus, *to reap the most benefits, we train the final model using sw-o and input a series sampled using last during the test time*.

2) Tracking Precision and Recall: We now evaluate how our predictions perform using the test dataset to prove the efficacy of our proposed approach. We first define:

- *True Positive (TP)* as the number of frames in which the tracking target is correctly identified.
- *False Positive (FP)* as the number of frames in which the system tries to track but the target is not present.
- *False Negative (FN)* as the number of frames in which the target is present but the system fails to track.

Thus we can define:

- *Precision (%)* as the number of frames in which the tracking target is correctly identified over the total number of frames processed by the system to track the target. *Precision* = $(\frac{TP}{TP+FP})$
- *Recall (%)* as the number of frames in which the tracking target is correctly identified over the total number of frames the target is present in the system. *Recall* = $(\frac{TP}{TP+FN})$

As shown in Fig. 9, our scheme outperforms all of the existing works. Kestrel and WatchDog both rely on the moving direction to infer the next destination. However, this is ineffective since pedestrians can easily change walking directions.

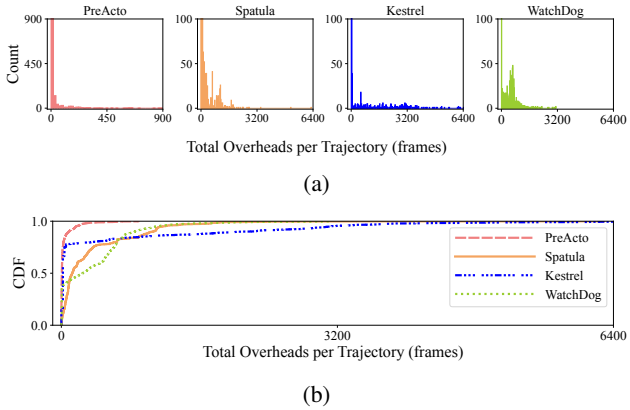


Fig. 10: (a) Histogram of re-ID overheads for all trajectories (b) CDF of re-ID overheads for all trajectories.

Moreover, in real-world systems, cameras are not always organized in well-defined grids, which makes moving direction is even less reliable. Also, both of them cannot point out an accurate point when the object will reappear and expect the transition time to fall into a range. Incorrectly identifying the next camera and transition time leads to lower *TP*s, higher *FP*s, and *FN*s. Spatula applies spatio-temporal thresholds to historical distributions to lower the numbers of cameras and frames to be processed. However, this results in having to process footage from several cameras that all satisfy the spatial threshold, which unnecessarily increases the number of *FP*s. While raising the spatial threshold may reduce the number of candidate cameras, it leads to a higher possibility of losing the target object, which raises the number of *FN*s. This is because Spatula only follows the distributions and is not capable of picking the most probable next camera for the pedestrian target under consideration. Our system, on the other hand, exploits the spatio-temporal relationships embedded within the mobility patterns of objects to perform both next camera prediction and transition time estimation accurately. As a result, only the necessary frames at the correct camera are processed, resulting in less computation waste.

3) Handover Overhead: While performing cross-camera tracking, when a camera receives a handover request, it also has to send a re-ID request to determine if the handovered object has reappeared in its FOV at the predicted time of handover. We consider this the *Handover Overhead*. To reduce the number of re-ID requests, the only way is to select the next camera and estimate the time of arrival accurately.

In Fig. 10a, we show the histograms of per-trajectory total overheads incurred by the handover processes of four schemes. Here, we can see that there are significant differences, especially between PreActo and the existing works. These schemes suffer from very high overheads since they take little advantage of spatio-temporal information for filtering out the frames that need to be processed. For the sake of clarity, we further highlight the difference between PreActo and Spatula in Fig. 10b, which shows the CDF of the total overheads. It can be seen that PreActo incurs nearly 90% of the

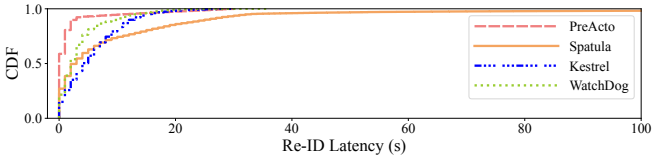


Fig. 11: CDF of all frames which have been delivered late. X axis denotes lag relative to the ground truth.

average handover overheads under 300 frames while Spatula does nearly 60%. The maximum overhead is 897 frames and 6294 frames for PreActo and Spatula, respectively. The reason for this stark difference is that besides choosing more than one camera candidate as explained earlier, Spatula also determines the arrival time from *camera A* to *camera B*, based on the minimum recorded transition time from A to B called f_0. As long as the distribution of transition time from A and B is "dense" at $[f_0, f_{current}]$, Spatula will consider $f_{current}$ potential transition time, without taking into consideration the trajectory itself (It is worth noting that this also the common weakness of Kestrel and WatchDog). PreActo, on the other hand, is able to both predict the next camera and estimate the transition time by employing DL-model-based extraction of spatio-temporal features from trajectories.

4) Re-ID Latency: In a cross-camera tracking system, objects disappear at one camera and after a certain amount of time, reappear at another. Once they reappear, to ensure tracking accuracy, the system needs to confirm their identities with their last image captured before disappearance using re-ID. In this context, re-ID latency refers to the time difference between when the objects physically reappear and when their identities are correctly confirmed. Fig. 11 shows how much lag is incurred when the edge server delivers handover re-ID responses to the cameras. The figure represents the relative lag to the ground truth on the X-axis. An on-time request would have a lag of 0, while any value larger than that signifies how much longer it takes for the request to be completed compared to a perfect computation. A high number of frames where the system unnecessarily tries to look for the object results in high re-ID latency, which in turn causes the tracking cameras' not to be able to recognize the handover-ed objects, leading to low tracking accuracy and precision. We can see that PreActo's edge server feeds approximately 60% of the frames to the camera with no lag, meeting the real-time requirement of our system, and 90% of the frames were delivered with latency lower than 5 seconds.

On the other hand, the existing systems, Spatula, Kestrel, and WatchDog, can deliver less than 30% of the frames on time, respectively. While it is true that Kestrel and WatchDog can effectively reduce the re-ID latency by running a less resource-consuming re-ID pipeline. However, as there are many frames to process, they still suffer from high latency.

VII. Future Work & Discussion

Our main plan for the future development of PreActo is the deployment in a real-world environment. One of the core objectives when designing PreActo is *scalability*. On the one hand, this has been achieved with an efficient tracking algorithm as presented in this paper. On the other hand, there requires a flexible approach for the implementation of the whole system, to which the tracking algorithm is the heart. To realize PreActo, we first apply the modularization principle of the microservice architecture. The full tracking pipeline is divided and modularized into well-defined components [40]. For instance, PreActo can be divided into [*RSTP Streaming server* $\rightarrow$ *Encoder* $\rightarrow$ *Object Detector* $\rightarrow$ *Re-ID* $\rightarrow$ *Tracking Algorithm*]. This not only allows easy offloading to the server (e.g., Re-ID), but also enables monitoring of each component and the system as a whole for the purpose of optimization and management. For example, a component that is causing a bottleneck can be quickly identified and fixed without affecting the other components. Moreover, the system can be easily imported into an open-source platform such as Kubernetes [41], which enables various scaling models, one of which is horizontal scaling. Once there are new cameras to be added to an existing functioning system, the Re-ID module can simply be "cloned" to increase the overall computation power. This can be done swiftly without causing discontinuation of service of the existing part of the system. Moreover, we address the challenge of rotating cameras. While the physical positions of the cameras do not change, their rotation angles can significantly affect the next camera prediction's accuracy. To tackle such an issue, we plan to add the rotation angles into the trajectories, which will be used as inputs for the prediction models at runtime.

VIII. Conclusion

Object tracking is one of the challenging applications in video analytics edge computing. Existing approaches have either an edge server or edge cameras performing real-time analysis using live video streams generated across multiple cameras. We argue that the former suffers from the overhead of handling video feeds from all the cameras while the latter also causes a computational bottleneck at the edge cameras. In this work, we devise a collaborative tracking system called PreActo. Our system also leverages ResNet-based models to extract spatio-temporal information from object trajectories. Thus, it can accurately select the next camera and predict transition time to minimize the number of cameras that are active at the same time, which in turn reduces the workload incurred on the edge server, as well as lessens the computation overhead at edge cameras. Results show that PreActo generates up to $7\times$ less processed frames, $2\times$ more on-time delivered frames while providing $1.5\times$ tracking precision improvement, compared to state-of-the-art cross-camera tracking systems.

Acknowledgement

This work was supported by Institute of Information & communications Technology Planning & Evaluation (IITP) grant funded by the Korea government (MSIT) (No. 2019-0-01126, Self-learning based Autonomic IoT Edge Computing).

REFERENCES

[1] T. Ricker, "The us, like china, has about one surveillance camera for every four people, says report," Mar. 2006. [Online]. Available: https://www.theverge.com/2019/12/9/21002515/surveillance-cameras-globally-us-china-amount-citizens

[2] M. Tan, R. Pang, and Q. V. Le, "Efficientdet: Scalable and efficient object detection," in *Proceedings of CVPR*, 2020, pp. 10 781–10 790.

[3] A. Bochkovskiy, C.-Y. Wang, and H.-Y. M. Liao, "YOLOv4: Optimal Speed and Accuracy of Object Detection," *arXiv*, apr 2020. [Online]. Available: http://arxiv.org/abs/2004.10934

[4] G. Jocher, "Yolo-v5," 2020. [Online]. Available: https://github.com/ultralytics/yolov5

[5] A. Aich, M. Zheng, S. Karanam, T. Chen, A. K. Roy-Chowdhury, and Z. Wu, "Spatio-Temporal Representation Factorization for Video-based Person Re-Identification," in *Proceedings of ICCV*, 2021, pp. 152–162.

[6] M. Li, "Self-supervised Geometric Features Discovery via Interpretable Attention for Vehicle Re-Identification and Beyond," in *Proceedings of ICCV*, 2021, pp. 194–204.

[7] T. Y.-H. Chen, L. Ravindranath, S. Deng, P. Bahl, and H. Balakrishnan, "Glimpse: Continuous, real-time object recognition on mobile devices," in *Proceedings of SenSys*. ACM, 2015, pp. 155–168.

[8] X. Liu, P. Ghosh, O. Ulutan, B. Manjunath, K. Chan, and R. Govindan, "Caesar: cross-camera complex activity recognition," in *Proceedings of SenSys*, 2019, pp. 232–244.

[9] S. Jain, X. Zhang, Y. Zhou, G. Ananthanarayanan, J. Jiang, Y. Shu, P. Bahl, and J. Gonzalez, "Spatula: Efficient cross-camera video analytics on large camera networks," in *Proceedings of SEC*. IEEE, 2020, pp. 110–124.

[10] H. Qiu, X. Liu, S. Rallapalli, A. J. Bency, K. Chan, R. Urgaonkar, B. Manjunath, and R. Govindan, "Kestrel: Video analytics for augmented multi-camera vehicle tracking," in *Proceedings of IoTDI*. IEEE, 2018, pp. 48–59.

[11] Z. Dong, Y. Lu, G. Tong, Y. Shu, S. Wang, and W. Shi, "Watchdog: Real-time vehicle tracking on geo-distributed edge nodes," *arXiv:2002.04597*, 2020.

[12] M. Satyanarayanan, "The emergence of edge computing," *Computer*, vol. 50, no. 1, pp. 30–39, 2017.

[13] G. Ananthanarayanan, P. Bahl, P. Bodík, K. Chintalapudi, M. Philipose, L. Ravindranath, and S. Sinha, "Real-time video analytics: The killer app for edge computing," *computer*, vol. 50, no. 10, pp. 58–67, 2017.

[14] "Aws deeplens," 2022. [Online]. Available: https://aws.amazon.com/deeplens/

[15] H. B. Pasandi and T. Nadeem, "Collaborative intelligent cross-camera video analytics at edge," in *Proceedings of AIChallengeIoT*, 11 2019, pp. 15–18.

[16] S. Jain, G. Ananthanarayanan, J. Jiang, Y. Shu, and J. Gonzalez, "Scaling video analytics systems to large camera deployments," in *Proceedings of HotMobile*, 2019, pp. 9–14.

[17] Z. Wang, W. Yan, and T. Oates, "Time series classification from scratch with deep neural networks: A strong baseline," in *Proceedings of IJCNN*. IEEE, 2017, pp. 1578–1585.

[18] X. Ning, K. Gong, W. Li, and L. Zhang, "JWSAA: Joint weak saliency and attention aware for person re-identification," *Neurocomputing*, vol. 453, pp. 801–811, 2021.

[19] M. Gou, S. Karanam, W. Liu, O. Camps, and R. J. Radke, "Dukemtmc4reid: A large-scale multi-camera person re-identification dataset," in *Proceedings of CVPR Workshops*, 2017, pp. 10–19.

[20] Y.-J. Cho, S.-A. Kim, J.-H. Park, K. Lee, and K.-J. Yoon, "Joint person re-identification and camera network topology inference in multiple cameras," *Computer Vision and Image Understanding*, vol. 180, pp. 34–46, mar 2019.

[21] S. Oh, A. Hoogs, A. Perera, N. Cuntoor, C.-C. Chen, J. T. Lee, S. Mukherjee, J. K. Aggarwal, H. Lee, L. Davis, E. Swears, X. Wang, Q. Ji, K. Reddy, M. Shah, C. Vondrick, H. Pirsiavash, D. Ramanan, J. Yuen, A. Torralba, B. Song, A. Fong, A. Roy-Chowdhury, and M. Desai, "A large-scale benchmark dataset for event recognition in surveillance video," in *Proceedings of CVPR*. IEEE, jun 2011, pp. 3153–3160.

[22] G. Song, B. Leng, Y. Liu, C. Hetang, and S. Cai, "Region-Based Quality Estimation Network for Large-Scale Person Re-Identification," in *Proceedings of the AAAI Conference on Artificial Intelligence*, vol. 32, no. 1, apr 2018.

[23] L. Zheng, L. Shen, L. Tian, S. Wang, J. Wang, and Q. Tian, "Scalable Person Re-identification: A Benchmark," in *Proceedings of ICCV*. IEEE, dec 2015, pp. 1116–1124.

[24] M. Zheng, S. Karanam, and R. J. Radke, "RPIfield: A New Dataset for Temporally Evaluating Person Re-identification," in *Proceedings of CVPR Workshops*. IEEE, jun 2018, pp. 1974–19 742.

[25] A. Dosovitskiy, G. Ros, F. Codevilla, A. Lopez, and V. Koltun, "CARLA: An open urban driving simulator," in *Proceedings of the 1st Annual Conference on Robot Learning*, 2017, pp. 1–16.

[26] A. Alahi, K. Goel, V. Ramanathan, A. Robicquet, L. Fei-Fei, and S. Savarese, "Social LSTM: Human trajectory prediction in crowded spaces," in *Proceedings of CVPR*, 2016, pp. 961–971.

[27] N. Deo and M. M. Trivedi, "Convolutional social pooling for vehicle trajectory prediction," in *CVPR Workshops*, 2018, pp. 1549–1557.

[28] K. Messaoud, I. Yahiaoui, A. Verroust-Blondet, and F. Nashashibi, "Attention Based Vehicle Trajectory Prediction," *IEEE T-IV*, vol. 6, no. 1, pp. 175–185, 2021.

[29] H. Xue, D. Q. Huynh, and M. Reynolds, "SS-LSTM: A Hierarchical LSTM Model for Pedestrian Trajectory Prediction," in *Proceedings of WACV*, 2018, pp. 1186–1194.

[30] X. Song, K. Chen, J. Sun, B. Hou, Y. Cui, B. Zhang, G. Xiong, and Z. Wang, "Pedestrian Trajectory Prediction Based on Deep Convolutional LSTM Network," *IEEE T-ITS*, vol. 22, no. 6, pp. 3285–3302, 2021.

[31] K. Zhou, Y. Yang, A. Cavallaro, and T. Xiang, "Omni-scale feature learning for person re-identification," in *Proceedings of ICCV*, 2019, pp. 3702–3712.

[32] B. T. Morris and M. M. Trivedi, "Learning and classification of trajectories in dynamic scenes: A general framework for live video analysis," in *Proceedings of AVVS*, 2008, pp. 154–161.

[33] B. Morris and M. Trivedi, "Learning, Modeling, and Classification of Vehicle Track Patterns from Live Video," *IEEE T-ITS*, vol. 9, no. 3, pp. 425–437, sep 2008.

[34] H. Xue, D. Q. Huynh, and M. Reynolds, "A Location-Velocity-Temporal Attention LSTM Model for Pedestrian Trajectory Prediction," *IEEE Access*, vol. 8, pp. 44 576–44 589, 2020.

[35] C. Yu, X. Ma, J. Ren, H. Zhao, and Y. Shuai, "Spatio-Temporal Graph Transformer Networks for Pedestrian Trajectory Prediction," in *Proceedings of ECCV*, 2020, pp. 125–141.

[36] P. Zhang, W. Ouyang, P. Zhang, J. Xue, and N. Zheng, "SR-LSTM: State refinement for lstm towards pedestrian trajectory prediction," in *Proceedings of CVPR*, 2019, pp. 12 077–12 086.

[37] "Jetson nano," 2022. [Online]. Available: https://developer.nvidia.com/embedded/jetson-nano

[38] Y. Zheng, Y. Chen, Q. Li, X. Xie, and W. Y. Ma, "Understanding transportation modes based on GPS data for web applications," *ACM Transactions on the Web*, vol. 4, no. 1, pp. 1–36, 2010.

[39] K. Zhou and T. Xiang, "Torchreid: A library for deep learning person re-identification in pytorch," *arXiv:1910.10093*, 2019.

[40] S. Y. Jang, B. Kostadinov, and D. Lee, "Microservice-based Edge Device Architecture for Video Analytics," in *Proceedings of SEC*, 2021, pp. 165–177.

[41] "Kubernetes," 2014. [Online]. Available: https://kubernetes.io/