Spatula: Efficient cross-camera video analytics on large camera networks

Samvit Jain[†], Xun Zhang[‡], Yuhao Zhou[‡], Ganesh Ananthanarayanan[¶], Junchen Jiang[‡], Yuanchao Shu[¶], Paramvir Bahl[¶], Joseph Gonzalez[†]
[†]UC Berkeley, [‡]University of Chicago, [¶]Microsoft Research

***Abstract*—Cameras are deployed at scale with the purpose of searching and tracking objects of interest (*e.g.,* a suspected person) through the camera network on live videos. Such *cross-camera analytics* is data and compute intensive, whose costs grow with the number of cameras and time. We present Spatula, a *cost-efficient* system that enables scaling cross-camera analytics on edge compute boxes to large camera networks by leveraging the spatial and temporal cross-camera correlations. While such correlations have been used in computer vision community, Spatula uses them to drastically reduce the communication and computation costs by pruning search space of a query identity (*e.g.,* ignoring frames not correlated with the query identity's current position). Spatula provides the first system substrate on which cross-camera analytics applications can be built to efficiently harness the cross-camera correlations that are abundant in large camera deployments. Spatula reduces compute load by $8.3\times$ on an 8-camera dataset, and by $23\times - 86\times$ on two datasets with hundreds of cameras (simulated from real vehicle/pedestrian traces). We have also implemented Spatula on a testbed of 5 AWS DeepLens cameras.**

I. Introduction

The Internet of Things (IoT) has led to an explosion of data sources and applications that rely on real-time inferences over these data. In parallel, the inference models have improved their accuracy, even surpassing humans for vision tasks, but at increased resource costs. This work addresses the systems challenges of scaling up IoT applications in the specific context of *live video analytics on a fleet of cameras*.

Live video analytics over a fleet of camera feeds embodies the above trends—*massive data sources* and *network-/compute-intensive analytics*. Enterprises deploy large camera networks and analyze camera feeds on their edge compute clusters for public safety and business intelligence [13], [3], [7], [36], [49], [67]. Chicago and London police analyze 30,000 and 12,000 camera streams in real time [6], [5]. These large camera networks enable many applications that rely on *cross-camera* analytics, *i.e.,* detecting, associating and tracking queried "identities" in the live videos as they move *across cameras over time*.

Scaling cross-camera analytics to large camera networks poses a substantial system challenge. Cross-camera analytics is computationally more challenging and network-intensive than "stateless" single-camera vision queries (*e.g.,* object detection in one camera feed) as it entails discovering *associations both across frames* and *across multiple cameras*. This shoots up the network and compute costs on the edge clusters for analyzing the ever-increasing camera networks.

Prior work falls short of addressing this challenge. Recent systems optimize the cost/accuracy tradeoffs of single-video analytics via frame sampling and/or cascaded filters for discarding frames [66], [64], [39], [42], [27], [30], but their cost optimization of the analytics on one video stream is *independent* of other streams. Thus, the compute/network cost of cross-camera analytics grows with the number of cameras over which a query identity is tracked, and with the duration of the identity's presence in the camera network.

Spatio-temporal correlations: Our main insight is that the cost of cross-camera analytics can be drastically reduced by exploiting the *physical correlations* of objects among the camera streams. Spatial correlations indicate *geographical association* between cameras – the probability that objects seen in a source camera will move next to a particular destination camera's field of view. Temporal correlations indicate association between cameras *over time* – the probability that objects seen in a source camera will move next to a destination camera's view *at a particular time*.

We develop Spatula, a cross-camera analytics system that leverages *spatio-temporal correlations* to stream and run cross-camera inference only on the set of cameras and frames that are most likely to contain the *query identity*, thereby decreasing both the network and compute costs (see Figure 1). While some prior works use spatio-temporal relationships between cameras to reduce the cost of the *profiling* of the video analytics (i.e., to obtain the resource-accuracy relationship for each video stream's analytics), the analytics on the live video itself is *not* optimized based on cross-camera relationships and each video stream's analytics is independent of other videos [39]. Spatula, in contrast, uses spatio-temporal correlations to limit the amount of data sent and analyzed, thus making the costs proportional to the number of cameras that the queried object appears in at any point in time, and *not* the total number of deployed cameras. A key property of large camera deployments is that objects of interest, at any time, tend to appear only in a *small fraction of the cameras*.

Challenges: Spatula highlights three challenges of applying physical properties (spatio-temporal correlations) in the IoT world to AI applications (cross-camera video analytics). First, automatically obtaining spatio-temporal correlations is expensive on unlabeled video data. Second, to maximize

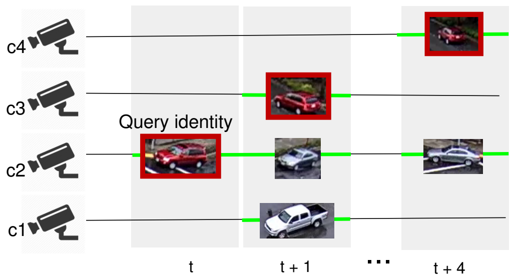

Figure 1. ***Spatio-temporal correlations* for video inference. The cameras (on the y-axis) are plotted according to their mutual distances, *e.g.,* c2 and c3 are spatially closer than c2 and c4. In searching for a query identity on c2 starting at t (marked in dark red), Spatula eliminates some cameras entirely (spatial filtering), and searches first on c1, c2 and c3 (but not c4), finds it in c3, and then searches only on c2 and c4 (but not c1 and c3). Also, it avoids searching in any of the cameras at times $t+2$ and $t+3$ (temporal filtering).**

the benefit of spatio-temporal correlations to various cross-camera inference tasks, we need clean abstractions with the necessary system supports. Finally, any spatio-temporal profile is bound to have errors that will lead to missing objects that must be rectified efficiently.

To tackle these challenges, Spatula operates in three distinct phases. 1) In an offline profiling phase, it constructs a cross-camera *spatio-temporal correlation model* from unlabeled video data, which encodes the locality observed in historical traffic patterns. This is an expensive one-time operation that requires detecting entities with an offline tracker and then aggregating them into a profile of cross-camera correlations. 2) At inference time, Spatula uses this spatio-temporal model to filter out cameras that are not correlated to the query identity's current position (camera), and is thus unlikely to contain its next instance. 3) Occasionally, this filtering will cause Spatula to miss query detections. In these cases, Spatula performs a *fast-replay search* on recently filtered frames (that it stores), uncovers the missed query instances, and gracefully recovers into its live search.

Evaluation Highlights: We evaluate Spatula using the well-studied DukeMTMC videos [56] from the Duke campus. On this 8-camera dataset, Spatula saves compute cost by $8.3\times$ and network cost by $5.5\times$ over a correlation-agnostic baseline (which is $\sim 90\%$ of the ideal savings). We also use GPS trajectories and 130/600 simulated cameras in Porto [12] and Beijing [71], and report savings of $23\times -86\times$. Additionally, Spatula largely *improves* precision, (*e.g.,* 39% on the DukeMTMC dataset), due to the significantly reduced number of false positives. These savings come at a drop in recall of only 1.6%. Finally, we have implemented Spatula on a small testbed of 5 AWS DeepLens smart cameras [16].

Contributions: Our work makes three main contributions.

1) We quantify the potential for harnessing spatio-temporal correlations in cross-camera video analytics.

2) We build a cross-camera video analytics system that learns and applies spatio-temporal profiles on live videos.

3) We develop robust error-handling mechanisms to avoid missed detections by storing and searching on recent videos.

II. Motivation and Background

We explain some example cross-camera video analytics applications (§II-A), the modules in their analytics pipelines (§II-B), and the compute models for video analytics (§II-C).

A. Cross-camera analytics applications

Large camera networks are installed in cities (such as London, Beijing, and Chicago), transport facilities (traffic intersections, airports), and enterprise campuses (corporate offices, retail shops) [15], [1], [6], [67]. A common class of applications in these camera deployments rely on *re-identifying and following objects (e.g., people or vehicles) as they move across the views of the different cameras*. The focus is on following select "objects of interest" that are typically provided by external entities (such as law enforcement). A key characteristic of cross-camera applications is that objects of interest occur only in a *small fraction of the cameras* at any given time.

1) Public safety. Cross-camera video analytics helps localize suspects after a security breach. For example, after a reported incident of a person pulling out a gun inside an office building, we will want to track that person (whose image can be obtained from the camera footage) across the cameras in the building while security personnel are dispatched.

Alternatively, after a major public attack (*e.g.,* in a train), law enforcement may track the accomplices of the identified perpetrator, which may be obtained from police databases that store people frequently associated with the perpetrator [67]. Following these accomplices across the thousands of cameras in the city allows for effective police apprehension.

2) Vehicle tracking in traffic cameras. In the U.S. and Europe, AMBER alerts are raised on suspected child abductions [2]. The license plate and vehicle details are obtained from investigations, and alerts are broadcast to citizens in the area [2]. Tracking of the suspect's vehicle across the thousands of cameras on highways and city streets can keep tabs on the suspect and victim, even as police intervene [49].

Likewise, when traffic police notice a vehicle speeding or making a dangerous maneuver, they will note its details and will be interested in tracking the vehicle as it moves across the city using cross-camera analytics to assess its behavior.

3) Retail store cameras. Using computer vision to improve shopping experience is a big thrust among retailers. "Special" shoppers (*e.g.,* loyal customers, or customers on wheelchairs) are identified *as they enter the store* and cross-camera analytics can be used to track them across the hundreds of cameras in the store to make sure they are provided timely attention (e.g., dispatching a store representative) when necessary.

B. Video analytics pipelines

Video analytics pipelines for cross-camera applications (in §II-A) typically consist of a series of *modules* on the decoded frames of the video stream: (1) an *object detection* module, which extracts and classifies objects of interest in each video frame (*e.g.,* people, gun), and (2) a *re-identification* module, which given a query image (*e.g.,* of a person), returns positions of co-identical instances of the query in subsequent frames (if present). Cross-camera analytics pipelines detect objects in each camera, and track the objects across cameras. Core to this pipeline is the vision primitive of *identity re-identification* [53], [57], [41]. Given an image of a query identity q, a re-identification (re-id) algorithm ranks every image in a gallery G based on its *feature distance* to q; the lower the distance the higher the similarity. Typically, features are the intermediate representation of a neural network trained to associate instances of co-identical entities.

Object detection and re-id are the most challenging steps of cross-camera video analytics – in terms of cost and accuracy – and our work focuses on improving both.

Compute cost: Tracking in large camera networks is computationally expensive. Tracking even a single object of interest through a camera network, after an initial detection, potentially requires analyzing every subsequent frame in every camera (without heuristics for geographic pruning).[1]

Network cost: Transmitting video streams to the edge cluster (or even cloud) for inference is network-intensive, especially with the superlinear growth in high-resolution videos [8]. Wi-Fi networks often have insufficient bandwidth to transmit many video streams simultaneously. In outdoor deployments where cameras have cellular connectivity, transmitting large data volumes increases the data costs [14].

Accuracy: Re-id is a non-trivial problem in computer vision [69], [60], being particularly difficult in crowded scenes and in large camera networks due to significant differences in lighting and viewpoint across cameras. Often, re-id models rely on weak signals (like clothing), thus making it difficult with a large gallery of objects in a frame.

Our use of spatio-temporal correlations to *prune* the video frames to analyze – i.e., run object detection and re-id – significantly cuts down the inference space, thus improving *both cost as well as accuracy*. While our focus is on cross-camera applications, we also show how spatio-temporal correlations improve the cost of even single-camera applications (§V-D).

C. Compute and network model

Consistent with existing deployments [25], [31], [50], our focus is on "edge" computation of video analytics with an edge compute (e.g., Azure Stack Edge [4]) that is managed by the enterprise (that has deployed the cameras). For example, cameras in an office building are analyzed in an edge cluster located in the same building. Traffic cameras in a city are analyzed in the local traffic command center [47]. Videos are streamed to this edge cluster and the pipeline modules (§II-B) including object detection and re-id are run on this edge. Spatio-temporal pruning leads to lesser data being transmitted to the edge (thus, reducing cellular data costs or Wi-Fi interference). Such pruning also reduces the compute load, enabling more video feeds to be processed on the edge box or alternately reducing the resources to be provisioned.

Our ideas also readily apply to a network of AI cameras (as we implement and deploy in §VIII), each of which consist of compute on-board, accelerators (*e.g.,* GPUs), and storage [16], [54]. Our techniques will enable each camera to be provisioned with lower resources, thus lowering their cost.

[1]Optimizations using frame sampling in each camera stream [30], [42] are orthogonal to our idea of using spatio-temporal correlations across cameras, and we will quantify this aspect in our experiments in §IX-B.

III. Quantifying spatio-temporal correlations

We analyze the potential of using spatio-temporal correlations for cross-camera video analytics using the DukeMTMC dataset [56]. We study *cross-camera identity tracking* that involves tracking an object of interest, in real time, through a camera network. In particular, given an instance of a query identity q (*e.g.,* a person) flagged in camera c_q at frame f, we return all subsequent frames, across all cameras, in which q appears as it moves around. We measure the reduction in compute, *i.e.,* the number of frames on which object detection and re-id operations (§II-B) are executed.

A. Empirical analysis on cross-camera correlations

We now present an empirical study to quantify the *cross-camera correlations* in the DukeMTMC dataset [56], one of the most popular benchmarks in computer vision person re-id and tracking [68], [61]. This quantification motivates our design of a video analytics system that leverages such correlations to improve the performance of cross-camera analytics. The DukeMTMC dataset contains footage from eight cameras placed on the Duke University campus (see Figure 2), in an area with significant pedestrian traffic. The field of views of the cameras do not mostly intersect, but the cameras are placed close enough that people frequently appear in multiple cameras, as is typical in camera deployments. The dataset contains over 2,700 unique identities across 85 minutes of footage, recorded at 60 frames per second [56].

1) Spatial correlation.

Cross-camera movement of individuals (or "traffic") demonstrates a high degree of spatial correlation. Here, "traffic" between cameras A and B is defined as the set of unique individuals detected in camera A that are *next* detected in camera B. (Note that a person that moves from A to B via camera C are excluded from the traffic count of $A \rightarrow B$ and

Figure 2. **DukeMTMC camera network [56]. Marked regions show the visual field of view of each camera.**

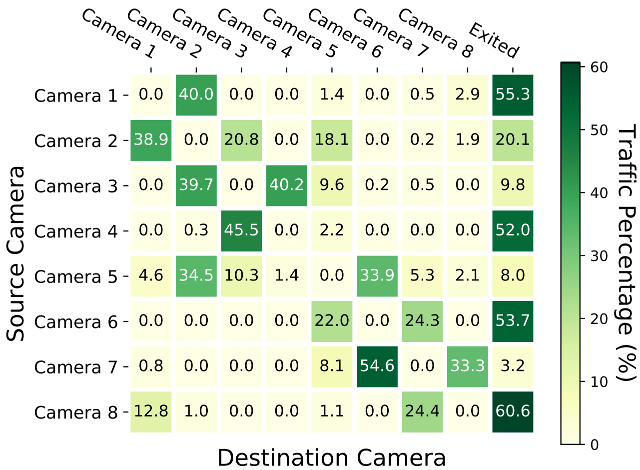

Figure 3. **Spatial correlations in the DukeMTMC dataset [56]. Cells display % of outbound traffic (individuals) from each camera that appears at other cameras. Each row corresponds to a particular source camera while each column to a destination camera; each row's values add up to 100%. The final column represents traffic that *exits* the camera network.**

instead included in the $A \rightarrow C$ traffic count.) We find that individuals seen at a camera c_q move next to only a small number of c_q's peer cameras. On the 8-camera DukeMTMC dataset, only 1.9 of 7 potential peer cameras, on average, receive *even 5%* of the total outbound traffic (or individuals) from a given camera. Figure 3 shows the full pair-wise spatial correlations.

Exploiting this insight can significantly reduce our workload, at little cost to accuracy, when searching for a query identity q (*e.g.,* a person), that was first detected in camera c_q. In comparison to a scheme that searches all $n-1$ peers, a smarter scheme that searches only those camera feeds that receive *at least 5%* of the traffic from c_q, reduces our compute by 3.7$\times$ (we search only 1.9 cameras instead of 7, or 3.7$\times$ fewer frames to run object detection and re-id; see §II-B), while *still* capturing 95% of all detections in our experiments.

An interesting aspect is that geographical proximity is *not* necessarily a good spatial filter. Consider camera-5 (Figure 3), out of which a significant fraction of individuals (traffic) go to cameras 2 and 6 *but not to* 7 or 8 even though they are also spatially proximate. Likewise, little traffic moves out of camera 8 to cameras 2 and 5 even though these are physically proximate. Thus, learning these patterns in a data-driven fashion is a more robust approach (as we will quantify in §IX-B). Data-driven learning also allows us to capture asymmetry in the traffic patterns between cameras, for e.g., over 50% of traffic from camera-7 move to camera-6 but less than 25% of traffic moves in the reverse direction from camera-6 to 7.

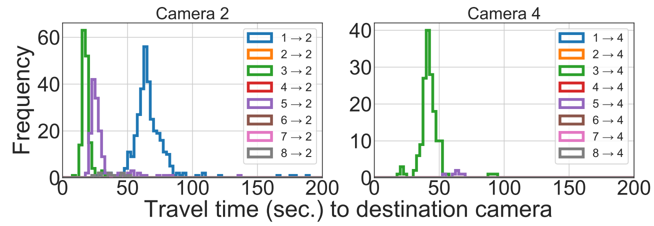

Figure 4. **Temporal correlations in the DukeMTMC dataset [56] (for two example destination cameras 2 and 4). Plots display distribution of inter-camera travel times. Each plot corresponds to traffic to the particular destination camera. Each colored line represents a particular source camera.**

2) Temporal correlation.

Cross-camera traffic also demonstrates a high degree of *temporal* correlation. As Figure 4 shows, travel times of individuals between a particular source camera and a destination camera in the DukeMTMC dataset are highly correlated. This is explained by the fact that these are static cameras and thus their pairwise distances are also static. Thus, for a given pair of cameras, the travel times for people to leave the feed of one camera and appear in the other camera are likely to be clustered around a mean value. In the DukeMTMC dataset, the average travel time between all camera pairs is 44.2s, and the standard deviation is only 10.3s (or only 23% of the mean).

Exploiting temporal correlations, even on its own, has the potential to provide compute savings. Given the task of locating a given query identity q, first identified in camera c_q, in one of the $n-1$ possible destination camera streams, we can simply search *each of the* $n-1$ streams (ignoring spatial correlations) but only for the time window when the query identities are most likely to show up. We probabilistically set the time window to be when *at least 98%* of the objects appear. Such an approach has the potential to reduce our compute load by 7.5$\times$ compared to a naive approach that does not use such a (time) windowed search. This shows the considerable potential in leveraging the tight distribution of travel times of individuals between the views of the cameras.

B. Potential gains: spatial & temporal correlations

We now put together the gains due to spatial and temporal filtering combined over a baseline that searches all $n-1$ cameras (for a maximum duration). We assume ideal knowledge about the spatial correlations between the cameras as well as the temporal characteristics of travel times of individuals between the views of the cameras. Using the same thresholds as in §III-A, our analysis shows a potential gain of 9.4× savings in the compute cost. This encouraging potential for savings, even for a 8-camera dataset, motivates us to both learn and exploit the spatio-temporal correlations for cross-camera video analytics. As we will show in §IX, Spatula achieves 8.3× reduction in compute cost, which is $\sim 90\%$ of the potential. In addition, the filtering of frames to search also improves the *precision* of the results from 51% for the baseline approach to 90% with Spatula, with little drop in recall.

IV. Spatula Overview

Building upon the strong spatial/temporal correlations across cameras seen in §III, we develop Spatula, a **r**esource-**e**fficient **cross-cam**era analytics system that leverages the correlations across cameras to reduce computing cost. As depicted in Figure 5, Spatula provides two core functions for cross-camera video analytics applications.

The spatio-temporal model (§V-A) describes the spatial and temporal correlation between cameras, and can be queried by applications. At a high level, one can query the model with two cameras, c_s and c_d, and a time window, and it will return how likely an object leaving c_s will appear in c_d (*i.e.,* the spatial correlation) and if it appears in c_d how likely it will appear within the time window (*i.e.,* temporal correlation).

The forward and replay analysis (§V-B and §V-C) perform real-time inference on live videos (*i.e.,* forward) as well as inference on history video (*i.e.,* replay). Both capabilities operate jointly, and replay search is inherently needed for spatio-temporal pruning: ignoring a camera due to weak spatial/temporal correlation will inevitably introduce false negatives that a baseline of searching all cameras would have avoided, so Spatula provides the abstraction of replay search to allow faster-than-real time search over some history videos (that were ignored) for error correction.

In §V-B we demonstrate how cross-camera identity tracking (tracking an identity across cameras over time from a known starting point) using spatio-temporal pruning. We also show the generality of Spatula by applying spatio-temporal pruning for cross-camera identity *detection* (finding a queried identity, e.g., a lost child, in a large camera deployment) in §V-D that is both an important *single-camera* application as well as ties to the cross-camera identity tracking by providing it the starting point for its tracking.

V. Spatio-temporal correlations in Spatula

We now describe Spatula's solution for leveraging spatio-temporal correlations in cross-camera video analytics.

A. Defining the spatio-temporal model

Spatula builds upon the cross-camera correlations in §III.

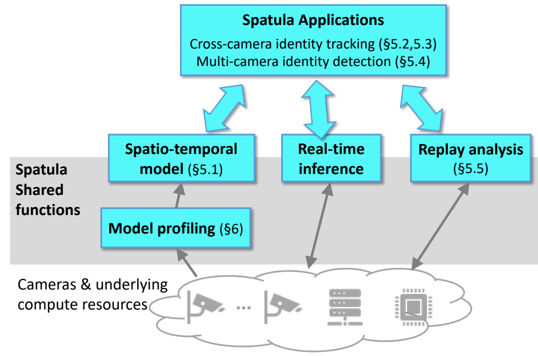

Figure 5. **Architecture of Spatula.**

1) Spatial correlations capture associations between camera pairs arising from the movement of traffic (individuals) between the views of the camera streams. The degree of spatial correlation S between two cameras c_s, c_d is quantified by the ratio of: (a) the number of individuals leaving the source camera's stream for the destination camera, $n(c_s, c_d)$, to (b) the total number of entities leaving the source camera:

$$S(c_s, c_d) = \frac{n(c_s, c_d)}{\sum_i n(c_s, c_i)}$$

When a large fraction of individuals that leave c_s's view are seen next in a camera c_i, we say that c_i is *highly correlated* to camera c_s. Note that S may be asymmetric (as seen in our analysis in §III-A1); camera c_s may *not* be highly correlated with camera c_i, even if the converse is true. In cross-camera identity search, Spatula exploits spatial correlations by prioritizing cameras that are highly correlated to the last camera where the queried identity q was spot (called *query camera*).

2) Temporal correlations capture associations between camera pairs *over time*. If a large fraction of the traffic leaving camera c_s for camera c_d arrives within durations t_1 and t_2, then camera c_d is said to be *highly correlated in the time window* $[t_1, t_2]$ to camera c_s. The degree of temporal correlation T between two cameras c_s, c_d during a window $[t_1, t_2]$ is the ratio of: (a) individuals reaching c_d from c_s within a duration window $[t_1, t_2]$ to (b) total individuals reaching c_d from c_s:

$$T(c_s, c_d, [t_1, t_2]) = \frac{n(c_s, c_d, [t_1, t_2])}{n(c_s, c_d)}$$

Indeed, cameras in real-world deployments have substantial temporal correlation (§III-A2). In cross-camera identity search, Spatula exploits temporal correlations by prioritizing the *time window* $[t_1, t_2]$ in which a destination camera is most correlated with the query camera.

Spatio-temporal model Given a source camera c_s, the current frame index f_{curr} (which serves as a timestamp), and a destination camera c_d, our proposed spatio-temporal model M outputs `true` if c_d is both spatially and temporally correlated with c_s at f_{curr}, and `false` otherwise. In our

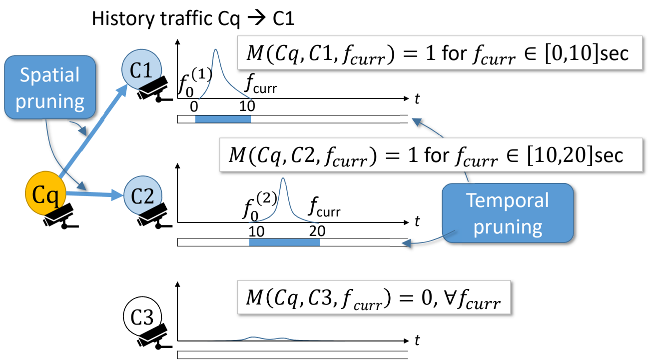

Figure 6. **Spatio-temporal correlations between camera Cq (where the object was first spotted) and three other cameras. C1 and C2 have spatial and temporal correlations with Cq (in different time intervals). C1 is correlated with Cq in the times [0, 10] but not otherwise; and C2 is correlated with Cq only in times [10, 20]. C3 is not correlated with Cq.**

description, the frame index f_{curr} serves the role of the timestamp.

The thresholds for being spatially correlated with c_s, and temporally correlated with c_s at time f_{curr} are model parameters. As an example, we may first wish to search cameras receiving at least $s_{\text{thresh}} = 5\%$ of traffic from c_s, during the time window containing the first $1 - t_{\text{thresh}} = 98\%$ of traffic from c_s. These parameter settings exclude both *outlier cameras* (cameras receiving less than 5% of the traffic from c_s) and *outlier frames* (frames containing the last 2% of the traffic from c_s). Defining s_{thresh} and t_{thresh} as a percent of traffic (or individuals) directly translates to precision and recall of the entities being tracked. M is formally defined as:

$$M(c_s, c_d, f_{\text{curr}}) = \begin{cases} 1, & S(c_s, c_d) \geq s_{\text{thresh}} \\ & \quad \textbf{and} \\ & T(c_s, c_d, [f_0, f_{\text{curr}}]) \leq 1 - t_{\text{thresh}} \\ 0, & \text{otherwise} \end{cases} \tag{1}$$

Here f_0 is the frame index at which the first historical arrival at c_d from c_s was recorded. The reason of having f_0 is because it takes time to travel from c_s to c_d, and cost savings can be maximized by not searching on frames while objects are moving between cameras. As a result, our temporal filter checks if the volume of historical traffic that arrived at c_d between $[f_0, f_{\text{curr}}]$ is less than $1 - t_{\text{thresh}}$ of the total traffic. This ensures that f_{curr} falls in the "dense" part of the travel time distribution, where we are likely to find q. (Note that we must check that $f_{\text{curr}} \geq f_0$. When $f_{\text{curr}} < f_0$, M is $\texttt{false}$.) Figure 6 shows an illustration for using M with f_0 values for each destination camera. (We construct the model M in §VI.)

Search hits and misses: Leveraging the spatio-temporal model M allows us to explore the subset of the inference

Algorithm 1 Tracking with the spatio-temporal model

input: video feeds $\{V_c\}$ for camera c,
  sp_corr$(c_s, c_d) \rightarrow \{\texttt{true}, \texttt{false}\}$
  tp_corr$(c_s, c_d, f) \rightarrow \{\texttt{true}, \texttt{false}\}$
for query $(q, f_q, c_q) \in Q$ **do**
  $q_{\text{feat}} = \text{features}(q)$ ▷ extract image features
  $f_{\text{curr}} = f_q + 1$ ▷ init current frame index
  $M_q = []$ ▷ init query match array
  phase $= 1$ ▷ start phase one
  while $(f_{\text{curr}} - f_q) \leq$ exit_t **do**
    $V_{\text{corr}} =$ **filter**(sp_corr, tp_corr, c_q, f_{curr}, V)
    frames = get_frames(V_{corr}, f_{curr})
    gallery = extract_entities(frames)
    ranked = rank_reid(q_{feat}, gallery)
    if ranked[0][dist] < match_thresh **then**
      M_q = append(M_q, ranked[0][img])
      q_{feat} = update_rep(q_{feat}, ranked[0][feat])
      $f_q = f_{\text{curr}}$
      phase $= 1$ ▷ reset to phase one
      break
    f_{curr} = increment(f_{curr})
    if phase $= 1$ **and** $T(c_s, c_d, [f_0, f_{\text{curr}}]) > 1 - t_{\text{thresh}}$ **then**
      $f_{\text{curr}} = f_q + 1$ ▷ reset frame index
      sp_corr = relax(sp_corr)
      tp_corr = relax(tp_corr)
      phase $= 2$ ▷ start phase two
output: matched detections $\{M_q\}$

space (camera streams and time windows) that is most likely to contain q. A "hit" reduces cost, as we avoid searching the entire space. On the (rare) misses, we go back and find q in the *past* video frames over all the camera streams we had filtered out using M. In §V-C, we will explain how we handle misses and mitigate the *delay* it introduces. Maximizing the cost savings from hits and minimizing the miss-induced delays is a tradeoff controlled by the parameters s_{thresh} and t_{thresh}.

B. Cross-camera identity tracking

Algorithm 1 explains our *cross-camera identity tracking*. In cross-camera identity tracking, the input consists of a query image q, last seen in frame f_q on camera c_q. (If the input does not contain the frame f_q, we can first run the next application, multi-camera identity detection, to locate it.) The goal is to flag all subsequent frames, on all cameras, where q appears. Note that q can appear again on the same camera ($c = c_q$), different cameras ($c \neq c_q$), or else exit the network altogether. For each query q, we begin by extracting image features q_{feat} and initializing an empty array of discovered matches M_q. For each frame, as explained in §II-B, we: (1) extract individuals (objects) from each frame using an object detection model, (2) rank the objects based on their feature similarity distance to q using a re-id model (§II-B).

If the top-ranked detection is within a threshold (*match_thresh* in Algorithm 1), i.e., a co-identical instance is found by the re-id model, we add the detection to our array of matches M_q, update our query *representation* q_{feat} to incorporate the features of the new instance of q, update the query frame index f_q to f_{curr}, and proceed with tracking q; lines 14-18. We continue searching until the gap between the last detected instance of q and our current frame index exceeds a pre-defined *exit threshold* (defined as exit_t in Algorithm 1). At this point, we conclude that q must have exited the camera network, and cease tracking q.

We apply the spatio-temporal model to cross-camera tracking as follows (marked in blue in Algorithm 1). The model M has two filters (lines 2 and 3): (1) **spatial_corr**(c_s, c_d), which given a source camera c_s and a destination camera c_d returns `true` if c_d is correlated with c_s, and (2) **temporal_corr**(c_s, c_d, f), which given a source camera c_s, a destination camera c_d, and a frame index f, returns `true` if c_d is correlated with c_s at f. At query time, these two functions are passed to the **filter** function (line 10), which given a list of video feeds V, returns the subset of cameras (V_{corr}) that are *both* spatially and temporally correlated to c_q at f_{curr}.

Applying **filter** reduces the inference search space, at each frame step f_{curr}, from all entity detections at f_{curr} on every camera to all entity detections at f_{curr} on *correlated* cameras. This allows us to abstain from running object detection and feature extraction models on non-correlated cameras, and reduces the size of the re-id gallery in the ranking step. If **filter** in Algorithm 1 were applied to the example in Figure 6, the set V_{corr} would be only C1 in in the times [0, 10], only C2 in the times [10, 20], and null set at all other times.

C. Handling pruning errors via replay search

Spatio-temporal pruning may cause a drop in recall: *missing actual occurrences* of the query identity q, which would be discovered by a baseline that exhaustively searches all the frames of all the cameras. When tracking on the spatially filtered cameras does *not* discover q after exit_t time (line 22 in Algorithm 1), we will initiate a "second pass" through the video frames that we skipped; we call this *replay search*.

Replay subset: We initiate replay search on a broader subset of cameras and timespans. In particular, we go back to the last camera that the queried identity was seen, c_q (*i.e.,* restart the tracking procedure from $f_{\text{curr}} = f_q + 1$, line 23, as f_q was the last frame the queried object was seen), and find all the correlated cameras and time windows that c_q is correlated with using the spatio-temporal profile *but* now with thresholds s_{thresh} and t_{thresh} decreased by a factor of 10. If we do discover an instance of q, we proceed with tracking from that detection, initiating a new phase one in Algorithm 1. If we still do not, we search the entire camera network until the exit threshold.

Note that despite relaxing s_{thresh} and t_{thresh}, the cameras over which we perform replay search will still be only a small fraction of the overall camera network and for only a small duration in the past. This is because the vast majority of cameras (in a large deployment) will have never seen traffic (individuals) from c_q. Implicit to replay search is also the ability to store videos in the past. However, this is only for the last few minutes (few 100 MBs even for HD videos).

Replay delay: Searching on videos from the past indicates that we are lagging behind tracking the identity. Thus, it is desirable to speed up the search process. Spatula processes the *historical videos* at *faster-than-real-time*.

a) Skip frame mode – Process the historical videos at lower frame rate (via frame sampling) and lower resolution (via frame downsizing) to increase processing rate but potentially lower accuracy. We use offline profiling [64], [66] to decide the frame rates and resolution to limit the drop in accuracy.

b) Parallelism mode – Process the historical videos by parallelizing them across other cameras or edge machines (depending on the setup; §II-C) that are idle. As explained above, the broader replay search is likely still only a small subset of all the videos, so spare resources will be available.

We implement both solutions and investigate their trade-offs on accuracy and delay in our evaluation (§IX-D).

D. Multi-camera identity detection

While our focus thus far has been on cross-camera video analytics, spatio-temporal models can also be applied to reduce the cost of *single-camera analytics*, e.g., find a lost baby or lost car in a mall's or city's cameras. This involves running object detectors *independently on each camera stream*, and is expensive for large deployments. In this section, we apply our cross-camera spatio-temporal model (§V-A) to such single-camera "identity detection". Not only is it an application of wide relevance on its own, it also ties closely with cross-camera tracking (§V-B) to provide it the starting point of the query q (which we have been calling as camera c_q).

Identity detection refers to finding a given identity q (*e.g.,* an image of a lost baby) in many camera streams. The intuition why the spatio-temporal model helps is that if q is not found in camera C1 and the spatio-temporal model indicates that most objects appearing in camera C2 have recently appeared in C1, then camera C2 is unlikely to contain q. In other words, the model allows to prune the cameras and time windows in which q is *unlikely* to be found based on when and where q was *not* found earlier. At any point of time, we maintain a probability for each camera to contain an object that has not been "scanned" (*i.e.,* not found in the camera feeds we have searched so far). The cameras with high values of this probability will be prioritized in the search.

We define $P_{c,w}$ to be the probability of any unscanned object (*i.e.,* an object that did not appear in any camera when it was searched) appearing in camera c in time window w. Thus, the greater the $P_{c,w}$ is, the more likely searching camera c in window w would yield a "hit". While we omit the details for calculating $P_{c,w}$, at any point in time, we search the camera c and time window w whose $P_{c,w}$ is greater than a threshold θ. If the identity is found, the search ends.

VI. Profiling spatio-temporal correlations

A final piece of Spatula system is the profiling and maintaining of the spatio-temporal correlations. Spatula takes an approach that builds on standard techniques from computer vision. Before Spatula is deployed, we first use a *multi-target, multi-camera* (MTMC) *tracker* to label entities in a dataset of historical video, collected from the same camera deployment on which the live tracking is executed. Logically, such a tracker will return for each detected entity instance i a tuple, (c_i, f_i, e_i), containing the camera identifier c_i, frame index f_i, and entity identifier e_i for the detection, respectively.

Using these, we compute $n(c_s, c_d, [t_1, t_2])$, the number of entities leaving any source camera c_s for any destination camera c_d within a time $[t_1, t_2]$. These quantities translate directly to our spatio-temporal model M in Eq. 1 (see §V-A).

Using MTMC trackers to profile spatio-temporal correlations in the history video is computationally expensive, and can neutralize the savings from the search pruning. This is because unlike single-target tracking, a MTMC tracker will track *all entities* in the dataset. To limit profiling overheads, we explore the trade-off between the robustness of offline profiling and the accuracy of subsequent single-target cross-camera tracking using the generated profile. In particular, the profiling cost can be reduced by labeling fewer frames with the MTMC tracker (*e.g.,* by choosing a subset of the data to label). At first glance, this will likely reduce the search accuracy as the spatio-temporal correlations is based on a sampled subset of entities. In practice, however, we found that despite labeling fewer frames for the profiling, our precision and recall drops are only mild, and thus our solution of labeling fewer frames significantly reduces the profiling cost *without* impacting accuracy. We evaluate this in §IX-E.

VII. Limitations & Future Work

We acknowledge two main limitations in Spatula's design.

1) The spatio-temporal model in §V-A has likelihood thresholds, that makes it vulnerable to missing "outliers", i.e., the rare spatial movement patterns or temporal anomalies in the movement speeds. While replay search (§V-C) detects such instances and is largely able to correct them, it is not perfect. Ensuring that no outliers are missed is part of future work.

2) Spatio-temporal correlations may change with time (*e.g.,* a road work may block a busy segment, which can reduce the correlation between two cameras). Spatula can trigger re-profiling (§VI) by keeping track of the number of objects that are missed in the normal pruned search but detected in the subsequent replay search (*i.e.,* in an "uncorrelated" time interval or camera), thus reacting to spikes in pruning errors.

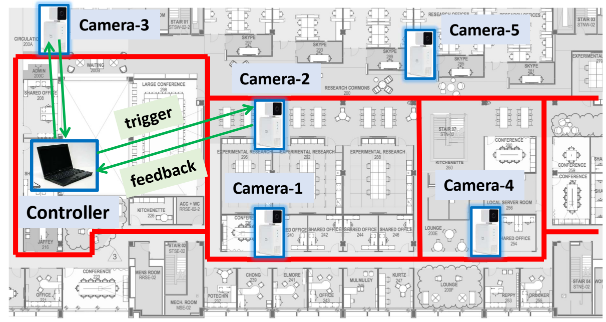

Figure 7. **Spatula testbed at AnonCampus with five AWS DeepLens smart cameras. The red lines show walkways in the building, and we learn the spatio-temporal correlation of people traversing the walkways. The *controller* and all the cameras exchange "trigger" and "feedback" messages.**

VIII. System Implementation & Deployment

We implement Spatula with 1.5K line of Python code over AWS DeepLens cameras [16]. Each DeepLens camera runs Ubuntu OS-16.04 LTS, and is equipped with an Intel Gen9 GPU and Intel Atom Processor CPU, 8GB RAM, and 16GB built-in storage. Our testbed includes five such cameras connected to each other via Wi-Fi and deployed on AnonCampus (Figure 7). In our testbed, video analytics modules (object detection, re-id) run on DeepLens's on-chip GPU and CPU.

We use a laptop (connected to the same Wi-Fi network as the cameras) to run the Spatula *controller*. The Spatula controller is responsible for profiling (§VI) and maintaining the spatio-temporal model of correlations among cameras. The connectivity between the controller and the cameras is only to exchange "control messages" and not video data. We implement two main control inferences (Figure 7):

1. A *trigger* message from the controller to a camera triggers the camera to start (or stop) searching for a specified query identity in its video within a specified time interval. The trigger message can also be used to initiate search in history videos for replay search (§V-C).
2. A *feedback* message from a camera to the controller notifies the controller on an interesting incident (*e.g.,* the specified identity has just been detected, or left the camera's view) in real-time. A feedback follows an activation message.

IX. Evaluation

Our evaluation of Spatula shows the following highlights.

1) Spatula's compute savings on the 8-camera DukeMTMC dataset is $8.3\times$ ($\sim$ 90% of the potential; §III), and its network saving is $5.5\times$. Spatula improves

Dataset	Comp. sav.	Netw. sav.	Prec.	Recall
AnonCampus	3.4x	3.0x	21.3% ↑	2.2% ↓
DukeMTMC	8.3x	5.5x	39.3% ↑	1.6% ↓
Porto	22.7x	n/a	36.2% ↑	6.5% ↓
Beijing	85.5x	n/a	45.5% ↑	7.3% ↓

Table I
SPATULA EVALUATION HIGHLIGHTS.

precision from 51% to 90%. On the larger simulated dataset of 130 and 600 cameras from Porto and Beijing, our savings are $23\times - 86\times$. (§IX-B, §IX-D)

2) Deployment on the 5-camera testbed with AWS DeepLens cameras leads to $3.4\times$ savings in compute and $3.0\times$ savings in network. (§IX-B)

3) Spatula's optimizes to keep the profiling costs small without impacting the precision and recall. (§IX-E)

A. Methodology

A. Datasets — We evaluate Spatula on four datasets.
1) AnonCampus dataset (§VIII) is captured by 5 DeepLens cameras deployed in a school building (see Figure 7).
2) DukeMTMC dataset is a video surveillance dataset from eight cameras in the Duke University (Figure 2). The data labels 2,700 unique identities and 4 million person detections.
3) Porto dataset is a simulated dataset generated from 1,710,671 GPS trajectories (time-stamped) obtained from 442 taxis running in the city of Porto, Portugal [12]. To emulate cameras, we *manually pin* 130 cameras at intersections of the city (we get the cameras' coordinates from Google Maps), and set each camera's field-of-view to be a square area centered at the camera with length $l = 100$m. Since our objective is to measure Spatula's gains in a large city-wide setting of cameras, we assume the accuracy of object detection and re-id equal to the values reported in DukeMTMC-reID [10].
4) Beijing dataset is also a simulated large dataset from 17,621 GPS trajectories [71]. Similar to the Porto dataset, we *manually pin* 600 cameras at intersections of the city and set $l = 100$m. We also use the Beijing dataset to evaluate Spatula under mixed mobility modes (*i.e.,* walking, cycling, driving).

B. Models — For our re-id model, we use a ResNet-50-based implementation of person re-id [9], trained in PyTorch. We then implement our tracking (Algorithms 1), which applies this model iteratively at inference time. For the AnonCampus dataset, we leverage person-reidentification-retail-0076 from the OpenVINO model zoo [34].

For our profiler (§VI) to build the spatio-temporal model, we apply the MTMC tracker [11] to label a subset of the dataset (*i.e.,* `profile` set with 16352 frames).

C. Workload — We run a set of 100 tracking queries, $\{q_i\}$, drawn from the `test` query partition of the DukeMTMC-reID dataset [10] (20 from the AnonCampus

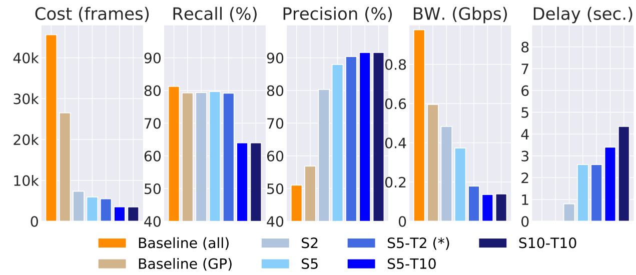

Figure 8. **Results for all-camera baseline (orange), geo-proximity baseline (tan) vs. five versions of Spatula (blues) on the DukeMTMC dataset. We argue S5-T2 (*) offers the best trade-off on all metrics.**

dataset, and 100 from the Porto dataset). Each tracking query consists of multiple *iterations*. Each iteration searches for the next *instance*, q_i^j of the query identity, starting with the initial instance q_i^0, and stopping when no more instances are found.

D. Metrics — We report the following four metrics which are computed over the entire query set. (i) *Compute cost* – Number of video frames processed, aggregated over all queries $\{q_i\}$. (ii) *Network cost* – Average network bandwidth usage of transmitting encoded videos required by search algorithms. (iii) *Recall (%)* – Ratio of query instances retrieved to all query instances in dataset, q_i^j. (iv) *Precision (%)* – Ratio of query instances retrieved to all retrieved instances, r_i^j. (v) *Delay (sec.)* – Lag between position of tracker and current video frame, in seconds, at the end of a tracking query.

E. Baselines — We compare Spatula against two schemes:
1) Baseline (all) - Searches for query identity q in all the cameras at every frame step. (*No* spatio-temporal filtering.)
2) Baseline (GP) - Searches for query identity q only in the cameras that are in geographical proximity to the query camera at every frame step. For DukeMTMC dataset, we manually set pairs of neighboring cameras using Figure 2 while for Porto and Beijing datasets, we set geographical proximity threshold to $10l$ (where $l = 100$m).
3) Spatula - Searches for query identity q only on cameras that are currently *spatio-temporally correlated* with c_q (as per Algorithm 1). The baselines and Spatula use the same person re-id model [9]. We consider various versions of Equation 1, corresponding to different spatio-temporal filters. Each version is coded as Ss-Tt, where s indicates the spatial filtering threshold and t indicates the temporal filtering threshold. Higher values of s and t indicate more aggressive filtering (no t value indicates no temporal filtering and helps measure the gains of spatial filtering alone). Frames are encoded and sent to the network after spatial and temporal filtering.

B. Spatio-temporal filtering gains

Figure 9, Figure 8, and Figure 10 compare the performance

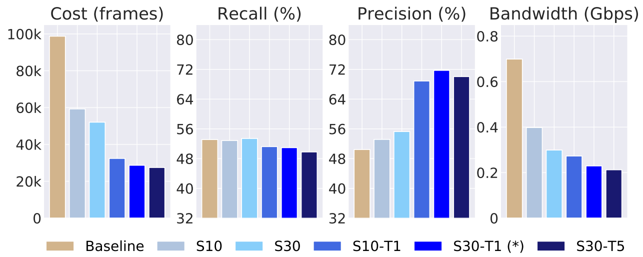

Figure 9. **Results for all-camera baseline (tan) vs. five versions of Spatula (blues) on the AnonCampus dataset. We argue S30-T1 (*) offers the best trade-off on all metrics.**

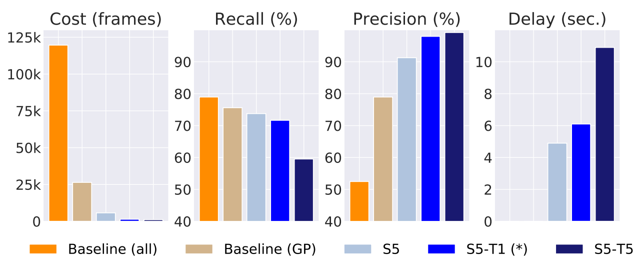

Figure 10. **Results for all-camera baseline (orange), geo-proximity baseline (tan) vs. three versions of Spatula (blues) on the Beijing dataset. We argue S5-T1 (*) offers the best trade-off on all metrics.**

of the baseline and various Spatula versions on different datasets[2]. We find that Spatula *significantly outperforms both baselines*, by (1) reducing compute and network cost and (2) improving precision, while maintaining comparable recall. It is noteworthy that the best thresholds for Spatula is dependent on the dataset. Spatula versions **S30-T1**, **S5-T2**, **S1-T1**, **S5-T1** offer the best trade-off between compute cost, recall, precision, and delay in the four datasets, and in general have to be tuned. We term these schemes **Spatula-O**(ptimal).

1) Compute and network cost – Each successive version of Spatula achieves lower compute cost than its predecessor, and all of them outperform the baselines. For instance, in Figure 8, the most aggressive version of Spatula, S10-T10 *achieves 13× lower compute cost* on 8 cameras than the all-camera baseline. Similarly, a maximal value of *3.6× compute savings* can be achieved in Figure 9. Baseline (GP) saves over Baseline (all) but its performance fluctuates on different settings due to the discrepancy between spatial correlation and geographical proximity (as also pointed out in §III-A1).

In comparison, **Spatula-O**'s costs are {3.4×, 8.3×, 22.7×, 85.5×} lower than the all-camera baseline in the five-camera (AnonCampus), eight-camera (DukeMTMC), 130-camera (Porto), and 600-camera (Beijing) dataset, respectively.

Similar trend in network saving can be seen in Figure 8 and Figure 9. On the DukeMTMC dataset, for instance,

[2]Due to space constraints, we don't include the figure on the Porto dataset. Instead, we report the numbers in the text.

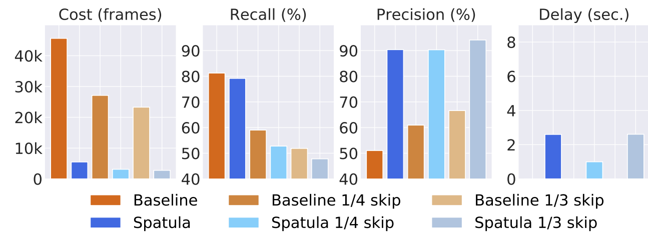

Figure 11. **Results for all-camera baseline vs. Spatula S5-T2 on the DukeMTMC dataset with frame skipping.**

Spatula-O achieves a 5.5× lower bandwidth usage due to the reduced number of videos for processing. Since encoding discontinuous video segments incurs more key frames, bandwidth saving is smaller than compute saving (8.3×).

2) Recall (%) – Compared with both baselines, recall of the Spatula versions *declines* slightly when spatial/temporal filtering is introduced. In Figure 8, for example, baseline (all) achieves recall of 81.3%. Both spatial-only schemes achieve 79.3% recall. **Spatula-O** achieves 79.7%, a 1.6% drop from the baseline. Similar patterns are observed on other datasets.

3) Precision (%) – Baseline (all) achieves precision of {50.4%, 51.1%, 49.6%, 52.5%} on four datasets, respectively. All versions of Spatula improve on this, but **Spatula-O** in particular achieves {71.7%, 90.4%, 85.8%, 98.0%} precision, which is a *gain of* {21.3%, 39.3%, 36.2%, 45.5%} *over the baseline*. Compared with baseline (GP), precision improvement of **Spatula-O** remains as high as {*33.5%, 15.6%, 19.0%*} on the DukeMTMC, Porto, and Beijing dataset. Higher precision is a key benefit of spatio-temporal filtering for cross-camera video analytics. By searching fewer irrelevant cameras, Spatula is less likely to declare matches that do not actually match the query.

4) Delay (sec.) – Here we report total cumulative lag (lag in the absence of replay search (§V-C)), averaged over all queries. We do not report the delay from the AnonCampus deployment since among all 20 queries, only one needed replay search. For DukeMTMC, Porto, and Beijing results, we find that delay increases with more spatial or temporal pruning. This is expected as there are more instances of misses. **Spatula-O**, in particular, incurs less delay than S5-T10, S5-T1 and S5-T5 but more delay than spatial-only filtering.

Given this analysis, **Spatula-O** offers a favorable tradeoff between the four metrics – achieving the lowest compute cost, the highest precision, competitive recall, and moderate lag (≈3.6s), when compared to the baselines.

Network constraints: While the above results on network savings were on the volume of data transferred (which is useful when cameras use the metered LTE network), we also evaluate Spatula under scenarios of bandwidth constraints, e.g., cameras connecting to the same Wi-Fi access point. Figure 12 shows F1 score of Spatula/Baseline (all) with

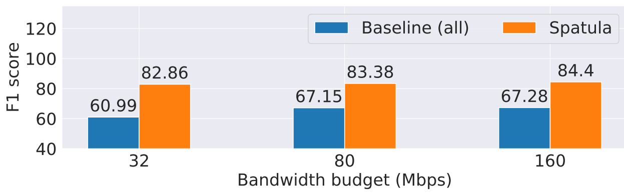

Figure 12. **Results for Spatula S5-T2 on the DukeMTMC data with network bandwidth constraints.**

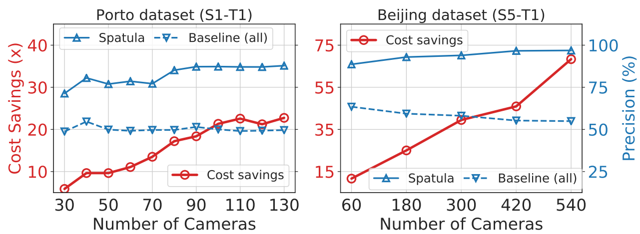

Figure 13. **Cost savings vs. number of cameras.**

different bandwidth budgets. Baseline (all) achieves F1 score of 61.0%, 67.2%, and 67.3%, on three bandwidth limitations, respectively. Spatula, on the other hand, improves on accuracy and achieves 82.9%, 83.4%, and 84.4% F1 score, 21.9%, 16.2%, 17.1% higher than the baseline. Since frames after spatio-temporal filtering can be transmitted at a higher resolution, this leads to a higher analytics accuracy. While Spatula is able to send spatio-temporally correlated frames consistently at 1080p under 160 Mbps bandwidth constraints (typical of Wi-Fi networks), Baseline (all) has to scale down the video resolution to 480p due to the bandwidth constraint.

Frame skipping: Prior work uses frame sampling to make *single-camera* analytics cheaper [66], [39], [30]. It is orthogonal to Spatula's spatio-temporal pruning. Figure 11 measures the impact of frame skipping—uniformly skip one in 3 frames, and one in 4 frames—on both baseline (all) and Spatula. Spatula maintains a much lower compute cost with cost savings of 8.6$\times$ and 8.4$\times$, which is in the same ballpark as without frame skipping of 8.3$\times$, thus showing the orthogonality of frame skipping to Spatula.

C. Large-scale camera data

The key objective of using the trajectories from the Porto and Beijing datasets (§IX-A) was to measure Spatula's gains at scale; unfortunately there are no video datasets available for hundreds of cameras. Figure 13 shows cost savings and precision of Spatula with increasing number of cameras from these two datasets. Cost savings steadily grows with increasing number of cameras, achieving up to 68$\times$ lower cost than baseline (all) in Spatula S5-T1 for 540 cameras. We believe this is an encouraging result for Spatula's value for large camera deployments. All through, Spatula maintains a $25\% - 40\%$ gain on precision with little impact on recall.

D. Replay search

We evaluate the lag reduction with replay search from §V-C:

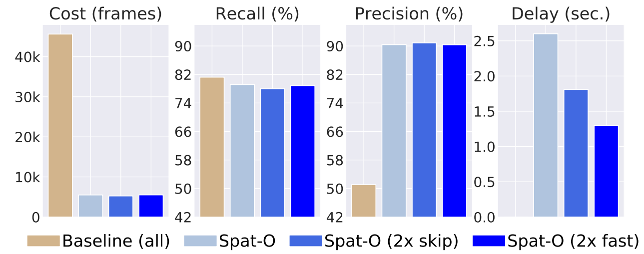

Figure 14. **Replay search. Schemes compared: baseline, Spatula-O (normal replay search), Spatula-O (2$\times$ skip), Spatula-O (2$\times$ fast-forward). Scheme 2$\times$ skip outperforms 2$\times$ fast-forward on both compute cost and delay.**

Skip frame mode - Employ a $\frac{x}{2}$ frame sampling rate to increase throughput on historical frames, at the price of lower accuracy (via missed detections). (**2x skip**)
Parallelism mode - Employ a $2x$ frame processing rate to increase throughput, at the price of increased compute cost (via increased resource usage). (**2x ff**)

Both schemes are applied to **Spatula-O**, and compared to (a) the all-camera baseline and (b) **Spatula-O** with the default *real-time* replay search, which incurs 2.6s of delay.

As Figure 14 shows, both **2x skip** and **2x ff** achieve delay reductions, decreasing final cumulative lag to 1.8s and 1.3s, respectively. The reason why **2x skip** doesn't halve the delay is due to the skipped query instances during the first round of replay search where s_{thresh} and t_{thresh} decreased by a factor of 10. Also, delay reductions from **2x skip** and **2x ff** come with different tradeoffs. **2x skip** reduces recall by 1.2% to 78.0%, but *increases precision* from 90.37% to 90.87% and *increase compute cost savings* from 8.30$\times$ to 8.68$\times$ better than the baseline (by processing fewer historical frames). **2x ff** does not impact recall and precision, but reduces compute cost savings from 8.30$\times$ to only 8.27$\times$ better than the baseline.

E. Profiling cost vs. tracking accuracy

Profiling cost increases with the number of frames that must be processed by the MTMC tracker (§VI). We investigate the trade-off between profiling cost and subsequent tracking accuracy. Specifically, we test if we can build a precise spatio-temporal model on smaller subsets of the training data obtained by uniformly sampling the frames. We apply a sampling rate of 8$\times$, 6$\times$, 4$\times$, 2$\times$, and 1$\times$ (using X in 8 frames) in the `profile` partition of the Duke dataset (§IX-A) for profiling, which translates to correspondingly lower profiling costs.

As Figure 15 shows, recall of Spatula during live tracking reaches the maximum of 80.1% with 6$\times$ sampling, *i.e.,* when half of the frames are labeled for offline profiling to obtain the spatio-temporal model. Interestingly, on either side of this, the recall falls. On the left side, the drop is caused by insufficient amount of profiling data. On the right side, the small drop is because extra data results in a spatial-temporal model being overfit to the `profile` partition.

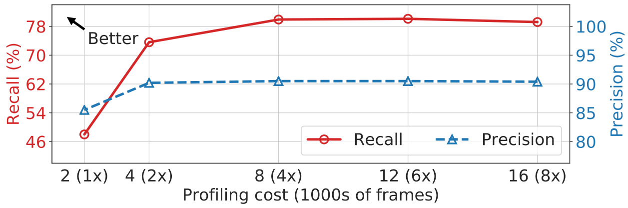

Figure 15. **Offline profiling cost vs. online recall. Profile intervals compared (in minutes of data used per camera):** 49.4 **min. (full),** 37.1 **min.,** 24.7 **min. (half),** 12.4 **min.,** 6.2 **min.**

This experiment indicates that spatial-temporal model can be built on a reasonably small set of training data (*i.e.,* 37.1 min). However, the exact amount of data to train the spatial-temporal model varies among datasets, and thus should be chosen carefully. Precision remains stable (∼90%) in Figure 15 when more than 4K (*i.e.,* 2× sampling) frames are used for training.

If we combine the profiling cost with the cost of the live video analytics, we see that Spatula would need to run *only* 34 live tracking queries to break-even with locality-agnostic tracking (calculations omitted). This represents a small fraction of the expected annual workload in large video analytics operations [67], [66] that track many hundreds of thousands of queries. Hence Spatula's profiling costs are small and will not dent the gains, leaving it to remain sizable.

F. Identity detection

Lastly, we evaluate Spatula's pruning on identity detection, the *single-camera* application described in §V-D. Spatula achieves as high as 7.6× cost reduction with $\theta = 0.95$ on the 8-camera DukeMTMC dataset (θ is the likelihood threshold for searching a camera's stream). Similar to cross-camera tracking, the gain on precision far outweighs the drop on recall. In fact, for $\theta = 0.75$, recall does not drop at all while precision improves by 28% even as cost savings stay at 6.6×. This experiment shows the generality of applying Spatula for both cross-camera as well as single-camera applications.

X. Related Work

Video Analytics Systems. A sizable body of work on video analytics has emerged recently [49], [66], [42], [30]. Chameleon exploits correlations in camera content (*e.g.,* velocity of objects) to amortize *profiling costs*, but not the live video analytics themselves [39]. These works leave three problems unexplored, each of which Spatula addresses. First, they focus on *single-frame* tasks (*e.g.,* object detection and classification), which are stateless. In contrast, surveillance applications, like the real-time tracking we focus on, involve multi-frame tracking, where future questions depend on past inference results. Second, they study *single camera* analytics. Thus, they do not explore the complexities involved in cross-camera *inference* on live video (*e.g.,* occlusions) that define applications such as person re-id. Third, in contrast to classification tasks, many security applications search for new object instances (*e.g.,* a suspicious person) where the training data is skewed toward negative examples. Our use of correlations across cameras, however, yields substantial accuracy gains.

Efficient Machine Learning. Improving ML models using model compression [28], [44], compact architectures [33], [46], knowledge distillation [29], [26], [17], and model specialization [42], [30], are orthogonal to Spatula, which would gain from any efficiency improvement of the models (*e.g.,* for re-id).

Orthogonal to systems that tradeoff resources and accuracy of models [27], [24], [22], [67], [20], [32], [52], [45], [38], Spatula entails a new approach: instead of running cheaper models, it processes on *less data* using spatio-temporal correlations.

Computer Vision. Techniques for person re-id and multi-target, multi-camera (MTMC) tracking make the following contributions: (1) new datasets [56], [70], [61], [60], (2) new neural network architectures [70], [61], [60], [55], (3) new training schemes [72], [58], [70], [61], or new understandings of multi-camera correlations [23], [59], [21], [48]. While the re-id and tracking modules in Spatula could take advantages of these advancements, the computer vision works do not address the *inference cost* of re-id and MTMC tracking [37], [18], [43], [51], [19], nor does it study *online* tracking (iterated re-id), a key application of interest in camera systems.

Mobility Modeling. Studies have shown promising results of generating human/vehicle mobility models from call detail records [35], [65], wireless signals [63], social media [40], GPS [71], and transactions in transportation systems [62], [65]. While none of the works apply mobility models to video analytics, Spatula could benefit from their techniques on building spatial-temporal models.

XI. Conclusions

Cross-camera analytics is a computationally expensive functionality that underpins a range of real-world video analytics applications, from suspect tracking to intelligent retail stores. We presented Spatula, a system that leverages a *learned model of cross-camera correlations* to drastically reduce the size of the inference time search space, thus reducing the cost of cross-camera analytics. Spatula directs its search towards the camera streams that likely contain the identity being tracked, while gracefully recovering from (rare) misses using a replay search on historical videos. Our results are promising: Spatula reduces compute workload by 8.3× on the 8-camera DukeMTMC dataset, and improve inference precision by 39%. On a simulated dataset of 130 cameras, its gains grow with the number of cameras. We have deployed a five-camera testbed on campus and are expanding our testbed.

REFERENCES

[1] Absolutely everywhere in beijing is now covered by police video surveillance. https://qz.com/518874/. Accessed: 2018-10-27.

[2] Amber alert. https://en.wikipedia.org/wiki/AMBER_Alert. Accessed: 2018-10-27.

[3] Are we ready for ai-powered security cameras? https://thenewstack.io/are-we-ready-for-ai-powered-security-cameras/.

[4] Azure stack edge. https://azure.microsoft.com/en-us/services/databox/edge/.

[5] British transport police: Cctv. http://www.btp.police.uk/advice_and_information/safety_on_and_near_the_railway/cctv.aspx. Accessed: 2018-10-27.

[6] Can 30,000 cameras help solve chicago's crime problem? https://www.nytimes.com/2018/05/26/us/chicago-police-surveillance.html. Accessed: 2018-10-27.

[7] Cloud-based video analytics as a service of 2018. https://www.asmag.com/showpost/27143.aspx.

[8] Data generated by new surveillance cameras to increase exponentially in the coming years. http://www.securityinfowatch.com/news/12160483/.

[9] Deep person reid. https://github.com/KaiyangZhou/deep-person-reid/. Accessed: 2018-10-28.

[10] Dukemtmc-reid. https://github.com/layumi/DukeMTMC-reID_evaluation. Accessed: 2018-10-28.

[11] Multi-target, multi-camera tracking. https://github.com/ergysr/DeepCC. Accessed: 2018-10-28.

[12] Taxi Service Trajectory - Prediction Challenge, ECML PKDD 2015 Data Set. https://archive.ics.uci.edu/ml/datasets/Taxi+Service+Trajectory+-+Prediction+Challenge,+ECML+PKDD+2015.

[13] Video meets the internet of things. https://www.mckinsey.com/industries/high-tech/our-insights/video-meets-the-internet-of-things. Accessed: 2018-10-28.

[14] Wi-fi vs. cellular: Which is better for iot? https://www.verypossible.com/blog/wi-fi-vs-cellular-which-is-better-for-iot.

[15] You're being watched: there's one cctv camera for every 32 people in uk. https://www.theguardian.com/uk/2011/mar/02/cctv-cameras-watching-surveillancel. Accessed: 2018-10-27.

[16] Amazon. AWS DeepLens. https://aws.amazon.com/deeplens/, 2017.

[17] Rohan Anil, Gabriel Pereyra, Alexandre Passos, Robert Ormandi, George E Dahl, and Geoffrey E Hinton. Large scale distributed neural network training through online distillation. In *ICLR*, 2018.

[18] Yinghao Cai and Gérard Medioni. Exploring Context Information for Inter-Camera Multiple Target Tracking. In *IEEE WACV*, 2014.

[19] Simone Calderara, Rita Cucchiara, and Andrea Prati. Bayesian-Competitive Consistent Labeling for People Surveillance. *TPAMI*, 30, 2007.

[20] Tiffany Yu-han Chen. Glimpse: Continuous, Real-Time Object Recognition on Mobile Devices Categories and Subject Descriptors. In *ACM SenSys*, 2015.

[21] Chen Change Loy, T. Xiang, and S. Gong. Multi-camera activity correlation analysis. In *IEEE CVPR*, 2009.

[22] Daniel Crankshaw, Xin Wang, Guilio Zhou, Michael J. Franklin, Joseph E. Gonzalez, and Ion Stoica. Clipper: A Low-Latency Online Prediction Serving System. In *USENIX NSDI*, 2017.

[23] T J Ellis, D Makris, and J K Black. Learning a multi-camera topology. In *Joint IEEE International Workshop on Visual Surveillance and Performance Evaluation of Tracking and Surveillance*, pages 165–171, 2003.

[24] Biyi Fang, Xiao Zeng, and Mi Zhang. NestDNN: Resource-Aware Multi-Tenant On-Device Deep Learning for Continuous Mobile Vision. In *ACM MobiCom*, 2018.

[25] Ganesh Ananthanarayanan, Victor Bahl, Peter Bodík, Krishna Chintalapudi, Matthai Philipose, Lenin Ravindranath Sivalingam, Sudipta Sinha. Real-time Video Analytics - the killer app for edge computing. In *IEEE Computer*, 2017.

[26] Urban Gregor, Krzysztof J. Geras, Ebrahimi Kahou Samira, Ozlem Aslan, Wang Shengjie, Abdelrahman Mohamed, Matthai Philipose, Matt Richardson, and Caruana Rich. Do Deep Convolutional Nets Really Need To Be Deep And Convolutional? In *ICLR*, 2017.

[27] S Han, H Shen, M Philipose, S Agarwal, A Wolman, and A Krishnamurthy. MCDNN: An approximation-based execution framework for deep stream processing under resource constraints. In *ACM MobiSys*, 2016.

[28] Song Han, Huizi Mao, and William J. Dally. Deep Compression: Compressing Deep Neural Networks with Pruning, Trained Quantization and Huffman Coding. In *ICLR*, 2016.

[29] Geoffrey Hinton, Oriol Vinyals, and Jeff Dean. Distilling the Knowledge in a Neural Network. In *NIPS*, 2014.

[30] Kevin Hseih, Ganesh Ananthanarayanan, Peter Bodik, Paramvir Bahl, Matthai Philipose, Phillip B. Gibbons, and Onur Mutlu. Focus: Querying Large Video Datasets with Low Latency and Low Cost. In *USENIX OSDI*, 2018.

[31] Chien-Chun Hung, Ganesh Ananthanarayanan, Peter Bodík, Leana Golubchik, Minlan Yu, Victor Bahl, and Matthai Philipose. VideoEdge: Processing Camera Streams using Hierarchical Clusters. In *ACM SEC*, 2018.

[32] Loc N Huynh, Youngki Lee, and Rajesh K Balan. DeepMon: Mobile GPU-based Deep Learning Framework for Continuous Vision Applications. In *ACM MobiSys*, 2017.

[33] Forrest N. Iandola, Matthew W. Moskewicz, Khalid Ashraf, Song Han, William J. Dally, and Kurt Keutzer. Squeezenet: Alexnet-level accuracy with 50x fewer parameters and <1mb model size. *arXiv:1602.07360*, 2016.

[34] Intel. OpenVINO. https://github.com/opencv/open_model_zoo, 2019.

[35] Sibren Isaacman, Richard Becker, Ramón Cáceres, Margaret Martonosi, James Rowland, Alexander Varshavsky, and Walter Willinger. Human Mobility Modeling at Metropolitan Scales. In *ACM MobiSys*, 2012.

[36] Samvit Jain, Ganesh Ananthanarayanan, Junchen Jiang, Yuanchao Shu, and Joseph E. Gonzalez. Scaling Video Analytics Systems to Large Camera Deployments. In *ACM HotMobile*, 2019.

[37] Omar Javed, Khurram Shafique, Zeeshan Rasheed, and Mubarak Shah. Modeling inter-camera space-time and appearance relationships for tracking across non-overlapping views. *CVIU*, 109, 2008.

[38] Angela H Jiang, Daniel LK Wong, Christopher Canel, Lilia Tang, Ishan Misra, Michael Kaminsky, Michael A Kozuch, Padmanabhan Pillai, Daniel L-K Wong, David G Andersen, and Gregory R Ganger. Mainstream: Dynamic Stem-Sharing for Multi-Tenant Video Processing. In *USENIX ATC*, 2018.

[39] Junchen Jiang, Ganesh Ananthanarayanan, Peter Bodik, Siddhartha Sen, and Ion Stoica. Chameleon: Video Analytics at Scale via Adaptive Configurations and Cross-Camera Correlations. In *ACM SIGCOMM*, 2018.

[40] Raja Jurdak, Kun Zhao, Jiajun Liu, Maurice AbouJaoude, Mark Cameron, and David Newth. Understanding Human Mobility from Twitter. *PLOS ONE*, 10(7):1–16, 07 2015.

[41] K. Jüngling, C. Bodensteiner, and M. Arens. Person reidentification in multi-camera networks. In *IEEE CVPR*, 2011.

[42] Daniel Kang, John Emmons, Firas Abuzaid, Peter Bailis, and Matei Zaharia. NoScope: Optimizing Neural Network Queries over Video at Scale. In *VLDB*, 2017.

[43] Cheng-Hao Kuo, Chang Huang, and Ram Nevatia. Intercamera Association of Multi-target Tracks by On-Line Learned Appearance Affinity Models. In *ECCV*, 2010.

[44] Hao Li, Asim Kadav, Igor Durdanovic, Hanan Samet, and Hans Peter Graf. Pruning Filters for Efficient ConvNets. In *ICLR*, 2017.

[45] Robert LiKamWa and Lin Zhong. Starfish: Efficient Concurrency Support for Computer Vision Applications. In *ACM MobiSys*, 2015.

[46] Min Lin, Qiang Chen, and Shuicheng Yan. Network in network. In *ICLR*, 2014.

[47] Franz Loewenherz, Victor Bahl, and Yinhai Wang. Video analytics towards vision zero. In *ITE Journal*, 2017.

[48] Chen Change Loy, Tao Xiang, and Shaogang Gong. Time-Delayed Correlation Analysis for Multi-Camera Activity Understanding. *International Journal of Computer Vision*, 90:106–129, 2010.

[49] Yao Lu, Aakanksha Chowdhery, and Srikanth Kandula. Optasia: A Relational Platform for Efficient Large-Scale Video Analytics. In *ACM SoCC*, 2016.

[50] Mahadev Satyanarayanan, Victor Bahl, Ramon Caceres, Nigel Davies. The Case for VM-based Cloudlets in Mobile Computing. In *IEEE Pervasive Computing*, 2009.

[51] Dimitrios Makris, Tim Ellis, and James Black. Bridging the gaps between cameras. In *CVPR*, 2004.

[52] Akhil Mathur, Nicholas D. Lane, Sourav Bhattacharya, Aidan Boran, Claudio Forlivesi, and Fahim Kawsar. DeepEye: Resource Efficient Local Execution of Multiple Deep Vision Models Using Wearable Commodity Hardware. In *ACM MobiSys*, 2017.

[53] N. Narayan, N. Sankaran, D. Arpit, K. Dantu, S. Setlur, and V. Govindaraju. Person Re-identification for Improved Multi-person Multi-camera Tracking by Continuous Entity Association. In *IEEE CVPR*, 2017.

[54] Qualcomm. Vision Intelligence Platform. https://www.qualcomm.com/news/releases/2018/04/11/qualcomm-unveils-vision-intelligence-platform-purpose-built-iot-devices, 2018.

[55] Kumar S. Ray, Vijayan K. Asari, and Soma Chakraborty. Object Detection by Spatio-Temporal Analysis and Tracking of the Detected Objects in a Video with Variable Background. In *arXiv:1705.02949*, 2017.

[56] Ergys Ristani, Francesco Solera, Roger Zou, Rita Cucchiara, and Carlo Tomasi. Performance Measures and a Data Set for Multi-Target, Multi-Camera Tracking. In *ECCV Workshops*, 2016.

[57] Ergys Ristani and Carlo Tomasi. Features for Multi-Target Multi-Camera Tracking and Re-Identification. In *IEEE CVPR*, 2018.

[58] Ergys Ristani and Carlo Tomasi. Features for Multi-Target Multi-Camera Tracking and Re-Identification. In *IEEE CVPR*, 2018.

[59] Kinh Tieu, Gerald Dalley, and W. Eric L. Grimson. Inference of Non-Overlapping Camera Network Topology by Measuring Statistical Dependence. In *IEEE ICCV*, 2005.

[60] Longhui Wei, Shiliang Zhang, Wen Gao, and Qi Tian. Person Trasfer GAN to Bridge Domain Gap for Person Re-Identification. In *IEEE CVPR*, 2018.

[61] Tong Xiao, Shuang Li, Bochao Wang, Liang Lin, and Xiaogang Wang. Joint Detection and Identification Feature Learning for Person Search. In *IEEE CVPR*, 2017.

[62] Zidong Yang, Ji Hu, Yuanchao Shu, Peng Cheng, Jiming Chen, and Thomas Moscibroda. Mobility Modeling and Prediction in Bike-Sharing Systems. In *ACM MobiSys*, 2016.

[63] Jungkeun Yoon, Brian D. Noble, Mingyan Liu, and Minkyong Kim. Building Realistic Mobility Models from Coarse-grained Traces. In *ACM MobiSys*, 2006.

[64] Ben Zhang, Xin Jin, Sylvia Ratnasamy, John Wawrzynek, and Edward A Lee. Awstream: Adaptive wide-area streaming analytics. In *Proceedings of the 2018 Conference of the ACM Special Interest Group on Data Communication*, pages 236–252. ACM, 2018.

[65] Desheng Zhang, Jun Huang, Ye Li, Fan Zhang, Chengzhong Xu, and Tian He. Exploring human mobility with multi-source data at extremely large metropolitan scales. In *ACM MobiCom*, 2014.

[66] Haoyu Zhang, Ganesh Ananthanarayanan, Peter Bodik, Matthai Philipose, Paramvir Bahl, and Michael J. Freedman. Live Video Analytics at Scale with Approximation and Delay-Tolerance. In *USENIX NSDI*, 2017.

[67] Tan Zhang, Aakanksha Chowdhery, Paramvir Bahl, Kyle Jamieson, and Suman Banerjee. The Design and Implementation of a Wireless Video Surveillance System. In *ACM MobiCom*, 2015.

[68] Zhimeng Zhang, Jianan Wu, Xuan Zhang, and Chi Zhang. Multi-Target, Multi-Camera Tracking by Hierarchical Clustering: Recent Progress on DukeMTMC Project. *arXiv:1712.09531*, 2017.

[69] Liang Zheng, Yi Yang, and Alexander G Hauptmann. Person Re-identification: Past, Present and Future. *arXiv:1610.02984*, 2015.

[70] Liang Zheng, Hengheng Zhang, Shaoyan Sun, Manmohan Chandraker, Yi Yang, and Qi Tian. Person Re-identification in the Wild. In *IEEE CVPR*, 2017.

[71] Yu Zheng, Xing Xie, and Wei-Ying Ma. Understanding Mobility Based on GPS Data. In *ACM Ubicomp)*, 2008.

[72] Zheng Zhu, Wei Wu, Wei Zou, and Junjie Yan. End-to-End Flow Correlation Tracking With Spatial-Temporal Attention. In *IEEE CVPR*, 2018.