

ReXCam: Resource-Efficient Cross-Camera Video Analytics at Scale

Samvit Jain, Xun Zhang, Yuhao Zhou,
Ganesh Ananthanarayanan, Junchen Jiang, Yuanchao Shu, Joseph E. Gonzalez

University of California Berkeley, University of Chicago, Microsoft Research

Abstract

Enterprises are increasingly deploying large camera networks for video analytics. Many target applications entail a common problem template: searching for and tracking an object or activity of interest (e.g. a speeding vehicle, a break-in) through a large camera network in live video. Such cross-camera analytics is compute and data intensive, with cost growing with the number of cameras and time. To address this cost challenge, we present ReXCam, a new system for *efficient cross-camera video analytics*. ReXCam exploits spatial and temporal locality in the dynamics of real camera networks to guide its inference-time search for a query identity. In an offline profiling phase, ReXCam builds a *cross-camera correlation model* that encodes the locality observed in historical traffic patterns. At inference time, ReXCam applies this model to filter frames that are not spatially and temporally correlated with the query identity’s current position. In the cases of occasional missed detections, ReXCam performs a *fast-replay search* on recently filtered video frames, enabling gracefully recovery. Together, these techniques allow ReXCam to reduce compute workload by $8.3\times$ on an 8-camera dataset, and by $23\times - 38\times$ on a simulated 130-camera dataset. ReXCam has been implemented and deployed on a testbed of 5 AWS DeepLens cameras.

1 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 models making these inferences have improved in accuracy, even surpassing humans for certain vision tasks, but at increased resource cost. This work addresses the systems challenges of scaling up IoT applications to enable *live video analytics on a fleet of cameras*.

Live video analytics over a fleet of camera feeds embodies two key trends—*massive data sources* and *compute-intensive inference* (e.g., neural nets). On the one hand, enterprises deploy large camera networks for public safety and business intelligence [11]. For instance, Chicago and London police access footage from 30,000 and 12,000 cameras to respond to crimes in real time [4, 5]. On the other hand, many applications rely crucially on cross-camera video analytics, *i.e.*, detecting, associating and tracking queried “identities” in the live streams as these identities move *across the camera feeds over time* (e.g., high-value shoppers in a store [8, 34] or suspects in a city [46, 66]). However, cross-camera analyt-

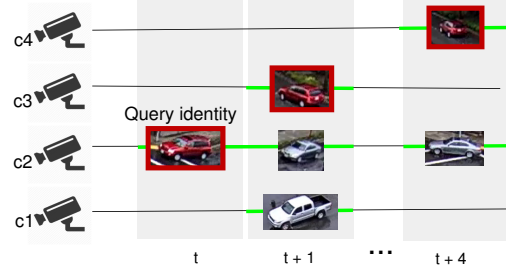


Figure 1: *Spatio-temporal correlations* for video inference. The cameras (on the y-axis) are plotted according to their mutual distances, *e.g.*, c1 and c2 are spatially closer than c1 and c3. In searching for a query identity starting at frame t (marked in dark red), ReXCam eliminates some cameras entirely (spatial filtering), as well as frames $t+2$ and $t+3$ (temporal filtering). In this example, ReXCam searches first on c1, c2, and c3 (but not c4), finds the target vehicle in c3, and then searches only on c2 and c4 (but not c1 and c3). The cameras and the times at which they are searched are marked in green. The unmarked portions represent compute savings.

ics applications are computationally more challenging than “stateless” single-camera vision tasks (such as object detection in one camera feed) as they entail discovering *associations across frames* and *across cameras*. Their compute cost thus grows *with* the number of cameras.

Prior work falls short of addressing this challenge. Work in computer vision improves accuracy of cross-camera analytics (e.g., [55, 58, 70]), but it has largely ignored the prohibitive compute costs. Recent systems have accelerated analytics on live videos via frame sampling and/or cascaded filters for discarding frames [25, 28, 37, 40, 63, 65]. However, they share a key drawback that they optimize the execution of analytics on single video feeds, *independent* of the other streams. Thus, the compute cost of cross-camera analytics still grows with more deployed cameras and longer activity time.

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. We develop ReXCam, a cross-camera analytics system that leverages inherent *spatio-temporal correlations* to aggressively prune the set of camera streams to be processed, thus decreasing compute costs. In the ideal case, ReXCam reduces cost 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 cross-camera applications is that objects of interest appear only in a *small number of cameras* at any time, even in large camera deployments.

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*. These *spatio-temporal* correlations enable ReXCam to guide its cross-camera inference search toward cameras and frames most likely to contain the *query identity* (see Figure 1). ReXCam’s use of spatio-temporal correlations to cut the cost of cross-camera analytics is fundamentally different than the cross-camera correlations used by recent work (e.g., [37]) that optimizes the resource-accuracy *profiling* but not the live video analytics itself, which still executes on each stream independently.

Challenges: ReXCam, at its core, applies the physical properties in the IoT world (spatio-temporal correlations across cameras) to high-level AI applications (cross-camera video analytics). This has led to three main challenges. First, automatically obtaining spatio-temporal correlations is expensive on unlabeled video data. Second, applying spatio-temporal correlations to existing single-camera inference modules (e.g., object trackers) is non-trivial and requires clean abstractions with the necessary system supports. Finally, any spatio-temporal profile is bound to have errors that will lead to missing objects, which need to be detected and rectified efficiently.

To tackle these challenges, ReXCam 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 converting them into an aggregate profile of cross-camera correlations. 2) At inference time, ReXCam 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 ReXCam to miss query detections. In these cases, ReXCam 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 ReXCam using the well-studied DukeMTMC video data [55] from the Duke campus. On this 8-camera dataset, ReXCam saves compute cost by $8.3\times$ over a correlation-agnostic baseline ($\sim 90\%$ of the ideal savings). These savings come at a drop in recall of only 1.6%. We also use a simulated dataset of 130 cameras in Porto (using GPS trajectories) [10], and report savings of $23\times - 38\times$. Interestingly, ReXCam *improves* precision by 39%, perhaps because the spatio-temporal pruning acts as a “low pass filter”.

Finally, we have implemented and deployed ReXCam on a small testbed of 5 AWS DeepLens smart cameras [13].

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.

2 Motivation and Background

We explain some example cross-camera video analytics applications (§2.1), the modules in their analytics pipelines (§2.2), and then the compute models for video analytics (§2.3).

2.1 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) [1, 5, 12, 66]. 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 [66]. 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 [46].

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 analyt-



Figure 2: **Illustration of identity re-identification.**

ics 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.

2.2 Video analytics pipelines

Video analytics pipelines for cross-camera applications (in §2.1) 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* [39, 50, 56]. 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 (Figure 2). 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 of them.

Cost. Tracking in large camera networks is computationally expensive. Tracking even a single object of interest through a camera network, after an initial detection, can potentially require analyzing every subsequent frame in every camera (without good heuristics for geographic localization).¹

Accuracy. Re-id is a non-trivial problem in computer vision [59, 68], 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 must rely on weak signals (like clothing), thus making it difficult among 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 (§5.4).

¹Optimizations using frame sampling in each camera stream [28, 40] are orthogonal to our idea of using spatio-temporal correlations across cameras, and we will quantify this aspect in our experiments in §8.2.

2.3 Setup and compute model

Consistent with existing deployments [23, 29, 47], our focus is on “edge” computation of video analytics. In our setup, all the cameras are in a high-speed *local* network with sufficient bandwidth to an edge compute box (e.g., Azure Data Box Edge [3]) that is managed by the enterprise (that has deployed the cameras). For example, cameras in an office building are analyzed in an edge box located in the same building. Traffic cameras in a city are analyzed in the local traffic command center [45]. Videos are streamed to this edge box and the pipeline modules (§2.2) including object detection and re-id are run on this edge. Reducing the compute load enables more video feeds to be processed on the edge box or alternately reduces the resources to be provisioned.

Our ideas also readily apply to a network of AI cameras (as we implement and deploy in §7), each of which consist of compute on-board, accelerators (e.g., GPUs), and storage [13, 53]. Our techniques will enable each camera to be provisioned with much lower resources, thus lowering their cost.

3 Quantifying spatio-temporal correlations

We analyze the potential of using spatio-temporal correlations for cross-camera video analytics using the DukeMTMC dataset [55]. 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 (§2.2) are executed.

3.1 Empirical analysis on cross-camera correlations

We now present an empirical study to quantify the *cross-camera correlations* in the DukeMTMC dataset [55], one of the most popular benchmarks in computer vision person re-id and tracking [60, 67]. 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 3), 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 [55].

3.1.1 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



Figure 3: DukeMTMC camera network [55]. Marked regions show the visual field of view of each camera.

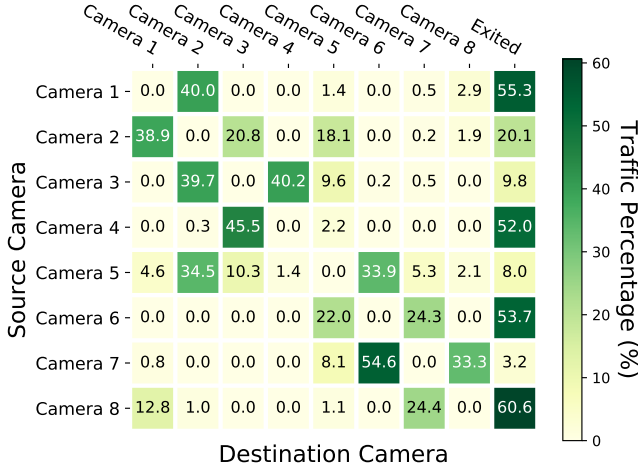


Figure 4: Spatial correlations in the DukeMTMC dataset [55]. Cells display % of outbound traffic (individuals) from each camera that appears at other cameras. Each row corresponds to a particular source camera while each column to a destination camera; each row’s values add up to 100%. The final column represents traffic that *exits* the camera network.

excluded from the traffic count of $A \rightarrow B$ and 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 4 shows the full pair-wise spatial correlations.

Exploiting this insight can significantly reduce our compute 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 §2.2), while *still* capturing 95% of all detections as per our experiments.

An interesting aspect is that geographical proximity is *not* necessarily a good spatial filter. Consider camera-5 (Figure 4), 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

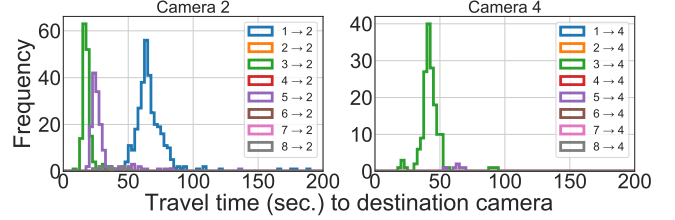


Figure 5: Temporal correlations in the DukeMTMC dataset [55] (for two example destination cameras 2 and 4). Plots display distribution of inter-camera travel times. Each plot corresponds to traffic to the particular destination camera. Each colored line represents a particular source camera.

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 §8.2). 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.

3.1.2 Temporal correlation.

Cross-camera traffic also demonstrates a high degree of *temporal* correlation. As Figure 5 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.

3.2 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 §3.1, our analysis shows a potential gain of $9.4\times$ 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 §8, ReXCam achieves $8.3\times$ 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 ReXCam, with little drop in recall.

4 ReXCam Overview

Building upon the strong spatial/temporal correlations across cameras seen in §3, we develop ReXCam, a **resource-efficient cross-camera** analytics system that leverages the correlations across cameras to reduce computing cost. As depicted in Figure 6, ReXCam provides two core functions for cross-camera video analytics applications.

The spatio-temporal model (§5.1) 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 (§5.2 and §5.3) 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 ReXCam provides the abstraction of replay search to allow faster-than-real time search over some history videos (that were ignored) for error correction.

In §5.2 we demonstrate how cross-camera identity tracking (tracking an identity across cameras over time from a known starting point) is performed using spatio-temporal pruning. We also show the generality of the functionalities of ReXCam 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 §5.4 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.

5 Spatio-temporal correlations in ReXCam

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

5.1 Defining the spatio-temporal model

ReXCam builds upon the cross-camera correlations in §3.

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 cam-

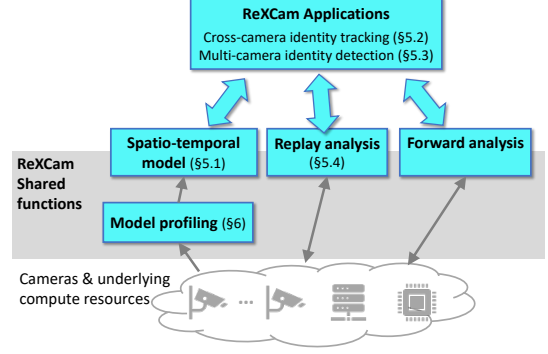


Figure 6: Architecture of ReXCam.

era’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 §3.1.1); camera c_s may *not* be highly correlated with camera c_i , even if the converse is true. In cross-camera identity search, ReXCam 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 (§3.1.2). In cross-camera identity search, ReXCam 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 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)

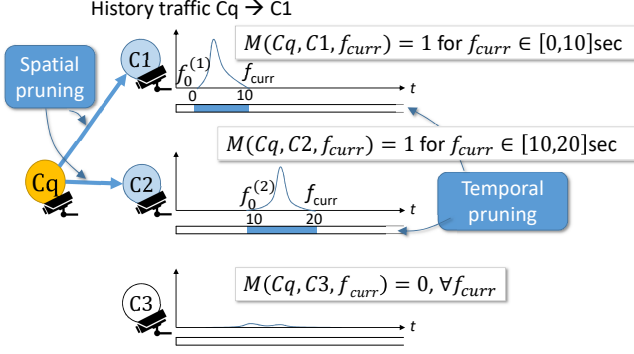


Figure 7: **Spatio-temporal correlations between camera C_q** (where the object was first spotted) and three other cameras. $C1$ and $C2$ have spatial and temporal correlations with C_q (in different time intervals). $C1$ is correlated with C_q in the times $[0, 10]$ but not otherwise; and $C2$ is correlated with C_q only in times $[10, 20]$. $C3$ is not correlated with C_q .

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:

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

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 false.) Figure 7 shows an illustration for using M with f_0 values for each destination camera. (We construct the model M in §6.)

Search hits and misses: Leveraging the spatio-temporal model M allows us to explore the subset of the inference 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 §5.3, 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} .

Algorithm 1 Tracking with the spatio-temporal model

```

1: input: video feeds  $\{V_c\}$  for camera  $c$ ,
2:    $\text{sp\_corr}(c_s, c_d) \rightarrow \{\text{true}, \text{false}\}$ 
3:    $\text{tp\_corr}(c_s, c_d, f) \rightarrow \{\text{true}, \text{false}\}$ 
4: for query  $(q, f_q, c_q) \in Q$  do
5:    $q_{\text{feat}} = \text{features}(q)$   $\triangleright$  extract image features
6:    $f_{\text{curr}} = f_q + 1$   $\triangleright$  init current frame index
7:    $M_q = []$   $\triangleright$  init query match array
8:    $\text{phase} = 1$   $\triangleright$  start phase one
9:   while  $(f_{\text{curr}} - f_q) \leq \text{exit\_t}$  do
10:     $V_{\text{corr}} = \text{filter}(\text{sp\_corr}, \text{tp\_corr}, c_q, f_{\text{curr}}, V)$ 
11:     $\text{frames} = \text{get\_frames}(V_{\text{corr}}, f_{\text{curr}})$ 
12:     $\text{gallery} = \text{extract\_entities}(\text{frames})$ 
13:     $\text{ranked} = \text{rank\_reid}(q_{\text{feat}}, \text{gallery})$ 
14:    if  $\text{ranked}[0][\text{dist}] < \text{match\_thresh}$  then
15:       $M_q = \text{append}(M_q, \text{ranked}[0][\text{img}])$ 
16:       $q_{\text{feat}} = \text{update\_rep}(q_{\text{feat}}, \text{ranked}[0][\text{feat}])$ 
17:       $f_q = f_{\text{curr}}$ 
18:     $\text{phase} = 1$   $\triangleright$  reset to phase one
19:    break
20:     $f_{\text{curr}} = \text{increment}(f_{\text{curr}})$ 
21:    if  $\text{phase} = 1$  and  $T(c_s, c_d, [f_0, f_{\text{curr}}]) > 1 - t_{\text{thresh}}$  then
22:       $f_{\text{curr}} = f_q + 1$   $\triangleright$  reset frame index
23:       $\text{sp\_corr} = \text{relax}(\text{sp\_corr})$ 
24:       $\text{tp\_corr} = \text{relax}(\text{tp\_corr})$ 
25:       $\text{phase} = 2$   $\triangleright$  start phase two
26: output: matched detections  $\{M_q\}$ 

```

5.2 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 §2.2, 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 (Figure 2).

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 7, the set V_{corr} would be only C1 in the times $[0, 10]$, only C2 in the times $[10, 20]$, and null set at all other times.

5.3 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 a 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 only needs to be 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. ReXCam 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 [63, 65] 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; §2.3) 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 (§8.3).

5.4 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 camera deployments. In this section, we apply our cross-camera spatio-temporal model (§5.1) 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 (§5.2) to provide it the starting point of the query q (which we have been referring to as camera c_q).

Identity detection refers to finding a given identity q (*e.g.*, an image of a lost baby or suspect) 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.

Formally, we define $P_{c,w}$ to be the probability of any un-scanned 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”. We also define P_c^* is the probability of the identity entering the whole camera network at camera c at any point in time. We estimate this value by looking at the history trace and dividing the number of objects who appear camera c first by total number of objects. Then $P_{c,0} = P_c^*$ and $P_{c,w}$ with $w > 0$ can be derived iteratively by the following equation:

$$P_{c,w} = P_c^* + \sum_{w_j \leq w, c_i} I_{c_i, w_j} \cdot P_{c_i, w_j} \cdot S(c_i, c) \cdot T(c_i, c, w)$$

where I_{c_i, w_j} is a binary flag indicating if camera c_i was searched at time window w_j ($I_{c_i, w_j} = 0$) or not ($I_{c_i, w_j} = 1$). The equation can be intuitively interpreted as following: the probability of query object q to appear in camera c and time window

w is the sum of the probability of it entering the whole network at c (i.e., P_c^*) and the probability of q moving from another camera c_i to at time w_j , i.e., $I_{c_i, w_j} \cdot P_{c_i, w_j} \cdot S(c_i, c) \cdot T(c_i, c, w)$.

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. Otherwise, we set $I_{c_i, w_j} = 0$ and update other $P_{c, w}$. This is run until we find the queried identity. §8.5 evaluates our gains with identity detection.

6 Profiling spatio-temporal correlations

A final piece of ReXCam system is the profiling and maintaining of the spatio-temporal correlations. ReXCam takes an approach that builds on standard techniques from computer vision. Before ReXCam 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 interval $[t_1, t_2]$. These quantities translate directly to our spatio-temporal model M in Eq. 1 (see §5.1).

However, directly using MTMC trackers to profile spatio-temporal correlations in the history video is computationally expensive, neutralizing 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 the 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 model. In particular, the profiling cost can be reduced by labeling fewer frames with the MTMC tracker (e.g., by selecting a lower frame sampling rate or choosing a smaller 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 empirically show this in §8.4.

Finally, ReXCam needs to cope with potential changes in the spatio-temporal correlations (e.g., a road work may block a busy segment, which can reduce the correlation between two cameras). These ‘changes are relatively infrequent, but when they do happen, ReXCam can automatically detect them and initiate re-profiling. In particular, ReXCam tracks the number of objects that are missed in the normal pruned search but detected in the subsequent replay search (in an “uncorrelated” time interval or camera), and triggers a re-profiling of the spatio-temporal correlations between the *corresponding cameras* when there is a spike in pruning errors. Note that the error in the spatio-temporal profile *during the* re-profiling

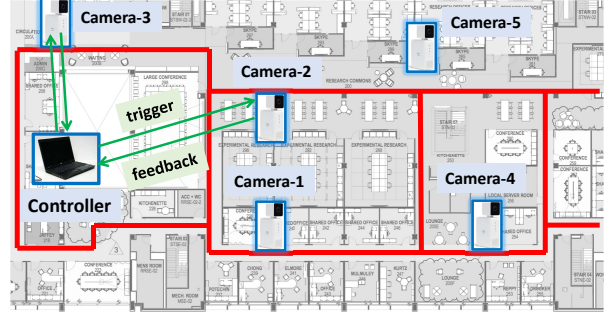


Figure 8: ReXCam testbed deployment at AnonCampus with five AWS DeepLens smart cameras. The red lines show the walkways in the building, and we learn the spatio-temporal correlation of people traversing the walkways. The controller and all the cameras exchange “trigger” and “feedback” messages.

will not affect ReXCam’s inference, but only increase latency because the replay search handles the errors.

7 System Implementation & Deployment

We implement ReXCam with 1.5K line of Python code over AWS DeepLens smart cameras [13]. 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 Anon-Campus (Figure 8). In our testbed, video analytics modules (object detection, re-id) run on DeepLens’s on-chip GPU and CPU. The testbed of smart cameras contrasts the alternate model for video analytics using nearby edge boxes (§2.3).

We use a laptop (connected to the same Wi-Fi network as the cameras) to run the ReXCam *controller*. The ReXCam controller is responsible for profiling (§6) 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 8):

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 (§5.3).
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 and is sent as soon as the incident occurs.

Fault tolerance: The cameras broadcast a heartbeat every few seconds to the controller to handle instances of cameras failing. The ReXCam controller can be replicated for resilience. The only persistent state held by the ReXCam controller is the model of spatio-temporal correlations, which is backed up, and is updated only at coarse timescales. The spatio-temporal



Figure 9: Example snapshots from AnonCampus (left) and DukeMTMC [55] (right) cameras.

pruning algorithm (Algorithm 1) is also stateless, and triggered by feedback messages from the cameras.

8 Evaluation

Our evaluation of ReXCam shows the following highlights.

- 1) ReXCam’s compute savings on the 8-camera DukeMTMC dataset is $8.3\times$ (which is $\sim 90\%$ of the potential; §3). ReXCam also improves precision from 51% to 90%. On the larger simulated dataset of 130 cameras from Porto, our savings grow with the number of cameras. (§8.2, §8.3)
- 2) Deployment on the 5-camera testbed with AWS DeepLens cameras leads to $3.4\times$ savings in compute. (§8.2)
- 3) ReXCam’s optimizes to keep the profiling costs small without impacting the precision and recall. (§8.4)

We evaluate ReXCam for single-camera analytics in §8.5.

8.1 Methodology

A. Datasets — We evaluate ReXCam on three datasets.

- 1) *AnonCampus dataset* (§7) consists of 35 minutes of 1080p video recorded at 24 frames per second, captured by five DeepLens cameras deployed in a school building (see Figure 8). The dataset is manually labeled with person identities.
- 2) *DukeMTMC dataset* is a video surveillance dataset with footage from eight cameras installed on the Duke University campus (see Figure 3). The data consists of 85 minutes of 1080p video from each camera recorded at 60 frames per second. In all, the footage contains over 2,700 unique identities and over 4 million person detections (all labeled).

Figure 9 shows snapshots from eight different cameras (four each) from the AnonCampus and DukeMTMC datasets. 3) *Porto dataset* is generated from 1,710,671 trajectories obtained from 442 taxis running in the city of Porto, Portugal between Jan. 2013 and June 2014 [10]. Each trajectory contains timestamps and GPS coordinates sampled every 15 seconds. To emulate cross-camera tracking, 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 $l = 100\text{m}$. We assume the accuracy of object detection and re-id equal to the values reported in DukeMTMC-reID [7] for objects in the camera’s view. The main objective is to measure ReXCam’s gains in a large city-wide setting of cameras.

B. Models — For our re-id model, we use an open-source, ResNet-50-based implementation of person re-id [6], trained in PyTorch on a subset of the Duke dataset called

DukeMTMC-reID [7]. We then implement our tracking (Algorithms 1), which applies this model iteratively at inference time to discover all instances of a query identity in the Duke dataset. Since DeepLens uses the cldnn and Intel GPUs, we leverage person-reidentification-retail-0076 from the OpenVINO model zoo [32] for re-id in the AnonCampus dataset.

To build our spatio-temporal model on unlabeled video data (simulating real deployment conditions), we apply an offline multi-target multi-camera (MTMC) tracker [9] (§6) to label every person detection in a subset of the dataset (*i.e.*, profile set with 16352 frames). We implement a *profiler* to extract spatial and temporal correlations from these labels.

C. Workload — We run a set of 100 tracking queries, $\{q_i\}$, drawn from the `test` query partition of the DukeMTMC-reID dataset [7] (20 from the AnonCampus dataset, and 100 from the Porto dataset). Each tracking query consists of multiple *iterations*. Each iteration involves searching for the next *instance*, q_i^j , of the query identity in the dataset, starting with the initial instance q_i^0 . A tracking query terminates when no more instances can be found. Experiments on the DukeMTMC dataset were conducted on AWS EC2 p2.xlarge instances (contains one Nvidia Tesla K80 GPU).

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) *Recall (%)* — Ratio of query instances retrieved to all query instances in dataset, q_i^j . (iii) *Precision (%)* — Ratio of query instances retrieved to all retrieved instances, r_i^j . (iv) *Delay (sec.)* — Lag between position of tracker and current video frame, in seconds, at the end of a tracking query. This will be 0 for a query if no replay search was performed. Compute cost, recall, and precision are reported in aggregation. Delay is reported as an average value per query.

E. Compared Schemes — To evaluate our spatio-temporal filtering, we compare against two schemes:

- 1) **Baseline (all)** - Searches for query identity q in all the cameras at every frame step. Uses state-of-the-art re-id model [6]. *no* spatio-temporal filtering is utilized.
- 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. Uses state-of-the-art re-id model [6]. For DukeMTMC dataset, we manually set pairs of neighboring cameras using Figure 3 while for Porto dataset, we set geographical proximity threshold to $4l$ (where $l = 100\text{m}$).
- 3) **ReXCam** - Searches for query identity q only on cameras that are currently *spatio-temporally correlated* with c_q (as per Algorithm 1). The same person re-id model is used as in the baseline [6]. 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). For instance, **S5-T2** filters cameras

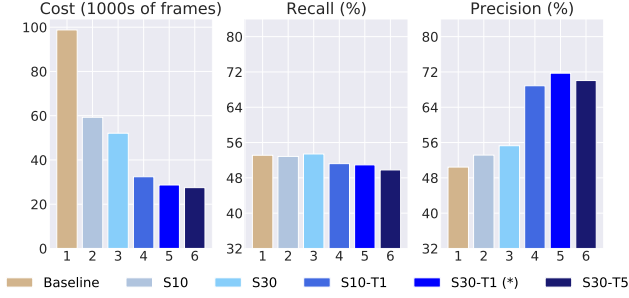


Figure 10: Results for all-camera baseline (tan) vs. five versions of ReXCam (blues) on the AnonCampus dataset. We argue S30-T1 (*) offers the best trade-off on all metrics.

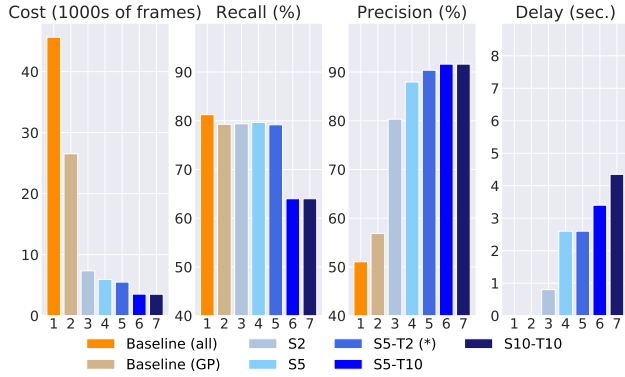


Figure 11: Results for all-camera baseline (orange), geoproximity baseline (tan) vs. five versions of ReXCam (blues) on the DukeMTMC dataset. We argue S5-T2 (*) offers the best trade-off on all metrics.

that receive $<5\%$ of the traffic from query camera c_q . In addition, its filter frames outside the time window containing the first 98% of traffic from c_q .

8.2 Spatio-temporal filtering gains

Figure 10, Figure 11 and Figure 12 compare the performance of the baseline and various ReXCam versions on three datasets, respectively. We find that ReXCam *significantly outperforms both baselines*, by (1) reducing compute cost and (2) improving precision, while maintaining comparable recall. It is noteworthy that the best thresholds for ReXCam is dependent on the dataset. ReXCam versions **S30-T1**, **S5-T2**, **S1-T1** offer the best trade-off between compute cost, recall, precision, and delay in the three datasets, and in general have to be tuned. We term these schemes **ReXCam-O**(ptimal).

1) Compute cost – Baseline (all) is by far the most compute-intensive, processing 98,760 frames for 20 queries and 45,638/85,890 frames for 100 queries on the DukeMTMC/Porto dataset, respectively. Baseline (GP) saves the cost quite a bit but its performance fluctuates on different settings due to the discrepancy between spatial correlation and geographical proximity (as also pointed out in §3.1.1). Each successive version of ReXCam achieves lower compute cost than its predecessor. For instance, in Figure 11, the most

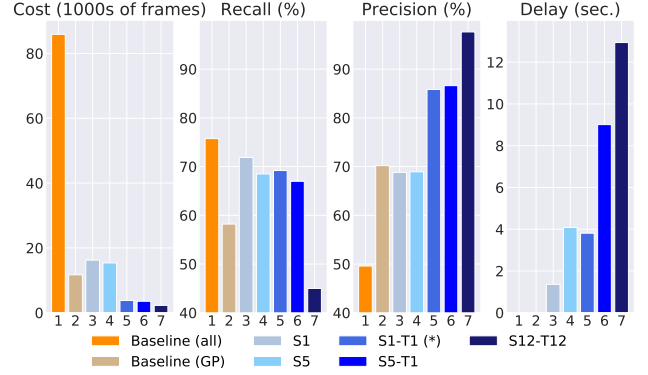


Figure 12: Results for all-camera baseline (orange), geoproximity baseline (tan) vs. four versions of ReXCam (blues) on the Porto dataset (130 cameras).

aggressive version of ReXCam, S10-T10, processes only 3,513 frames, and achieves $13\times$ lower compute cost on 8 cameras than the all-camera baseline. Similarly, a maximal value of $3.6\times$ compute savings can be achieved in Figure 10.

In comparison, **ReXCam-O** processes 28,680/5,500/3,776 frames, which translates to $3.4\times/8.3\times/23\times$ lower cost than the all-camera baseline in the five-camera (AnonCampus), eight-camera (DukeMTMC), and 130-camera (Porto) dataset.

2) Recall (%) – Compared with both baselines, recall of the ReXCam versions *declines* slightly when spatial/temporal filtering is introduced. In Figure 11, for example, baseline (all) achieves recall of 81.3%. Both spatial-only schemes achieve 79.3% recall. **ReXCam-O** achieves 79.7%, a 1.6% drop from the baseline. Similar patterns are observed in Figure 10 and Figure 12. The reason why recall becomes lower in the AnonCampus deployment is because of the increased instances of occlusions in indoor environments (see Figure 9). Note that in Figure 12, recall drops significantly from baseline (all) to baseline (GP), as a number of relevant cameras are mistakenly excluded by geographical proximity-based pruning.

3) Precision (%) – Baseline (all) achieves precision of 50.4%, 51.1% and 49.6% on three datasets, respectively. All versions of ReXCam improve on this, but **ReXCam-O** in particular achieves 71.7%/90.4%/85.8% precision, which is a *gain of 21.3%/39.3%/36.2% over the baseline*. Compared with baseline (GP), precision gain from **ReXCam-O** remains as high as 33.5%/15.6% on the DukeMTMC and Porto dataset. Higher precision is a key benefit of spatio-temporal filtering for cross-camera video analytics. By searching fewer irrelevant cameras, and fewer irrelevant frames, ReXCam 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 (§5.3)), 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 both DukeMTMC and Porto results, we find that delay increases with more spatial or temporal pruning. This is expected as there are more instances of misses. **ReXCam-O**, in

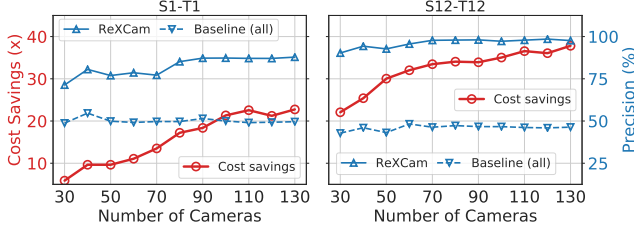


Figure 13: Cost savings vs. number of cameras (Porto dataset).

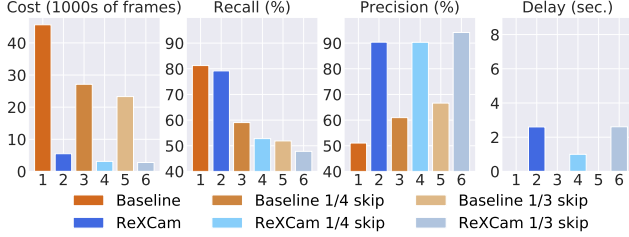


Figure 14: Results for all-camera baseline vs. ReXCam S5-T2 on the DukeMTMC dataset with frame skipping.

particular, incurs moderate delay – less delay than S5-T1 and S5-T10 but more delay than spatial-only filtering.

Given this analysis, **ReXCam-O** offers a favorable trade-off between the four metrics – achieving nearly the lowest compute cost ($3.4\times/8.3\times/23\times$ lower), nearly the highest precision ($21.3\%/39.3\%/36.2\%$ higher), competitive recall ($2.2\%/1.6\%/6.5\%$ lower), and moderate lag ($\approx 3.2s$), when compared to the locality-agnostic, all-camera baseline. Next, we analyze the impact of two key factors on ReXCam.

Large-scale camera data: The key objective of using the trajectories from the Porto dataset was to experiment on ReXCam’s gains at scale (§8.1); unfortunately there are no video datasets available for hundreds of cameras. Figure 13 shows cost savings and precision of ReXCam/Baseline (all) with increasing number of cameras. Cost savings steadily grows with increasing number of cameras, achieving up to $38\times$ lower cost than baseline (all) in ReXCam S12-T12 for 130 cameras. We believe this is an encouraging result for ReXCam’s value for large camera deployments. All through, ReXCam maintains a 34.5% gain on precision with little impact on recall.

Frame skipping: Frame sampling is a key technique in prior work [28, 37, 65] to make *single-camera* analytics cheaper. Such techniques are orthogonal to ReXCam’s spatio-temporal pruning for cross-camera analytics, and we quantify our point. Figure 14 measures the impact of frame skipping—uniformly skip one in 3 frames, and one in 4 frames—on both baseline (all) and ReXCam. As shown in the figure, ReXCam maintains a much lower compute cost in both skipping cases. Specifically, the cost savings are $8.6\times$ 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 ReXCam.

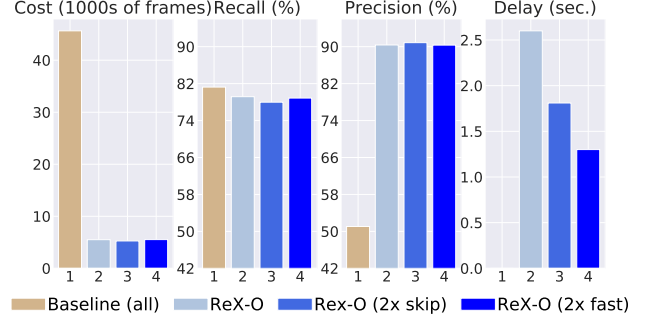


Figure 15: Replay search. Schemes compared: baseline, ReXCam-O (normal replay search), ReXCam-O ($2\times$ skip), ReXCam-O ($2\times$ fast-forward). Scheme $2\times$ skip outperforms $2\times$ fast-forward on both compute cost and delay.

8.3 Replay search

In this section, we evaluate the effectiveness in reducing lag in replay search using the two proposed schemes from §5.3: *Skip frame mode* - Employ a $\frac{1}{2}$ frame sampling rate to increase throughput on historical frames, at the price of lower accuracy (via missed detections). ($2\times$ skip)

Parallelism mode - Employ a $2\times$ frame processing rate to increase throughput, at the price of increased compute cost (via increased resource usage). ($2\times$ ff)

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

As Figure 15 shows, both $2\times$ skip and $2\times$ ff achieve delay reductions, decreasing final cumulative lag to 1.8s and 1.3s, respectively. The reason why $2\times$ 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 $2\times$ skip and $2\times$ ff come with different tradeoffs. $2\times$ 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). $2\times$ 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.

8.4 Profiling cost vs. tracking accuracy

Profiling cost increases with the number of frames that must be processed by the MTMC tracker (§6). We investigate the trade-off between profiling cost and subsequent tracking accuracy. Specifically, we test whether 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 (§8.1) for profiling, which translates to correspondingly lower profiling costs.

As Figure 16 shows, recall of ReXCam 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

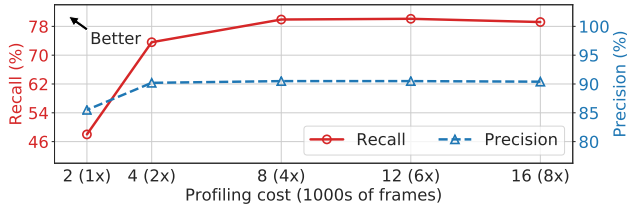


Figure 16: **Offline profiling cost vs. online recall. Profile intervals compared (in minutes of data used per camera): 49.4 min. (full), 37.1 min., 24.7 min. (half), 12.4 min., 6.2 min.**

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. 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 ($\sim 90\%$) in Figure 16 when more than 4K (*i.e.*, $2\times$ sampling) frames are used for training.

If we combine the profiling cost with the cost of the live video analytics, we see that ReXCam 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 [65, 66] that track many hundreds of thousands of queries. Hence ReXCam’s profiling costs are small and will not dent the gains, leaving it to remain sizable.

8.5 Identity detection

Lastly, we evaluate ReXCam’s spatio-temporal pruning on identity detection, the *single-camera* application described in §5.4. As Figure 17 shows, ReXCam achieves as high as $7.6\times$ 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 trends in 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 $6.6\times$. This experiment shows the generality of applying ReXCam for both cross-camera as well as single-camera applications.

9 Related Work

Video Analytics Systems. A sizable body of work on video analytics has emerged recently [28, 40, 46, 65]. Chameleon exploits correlations in camera content (*e.g.*, velocity of objects) to amortize *profiling costs*, but not the cost of the video analytics itself [37]. These works leave three problems unexplored, each of which ReXCam 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 track-

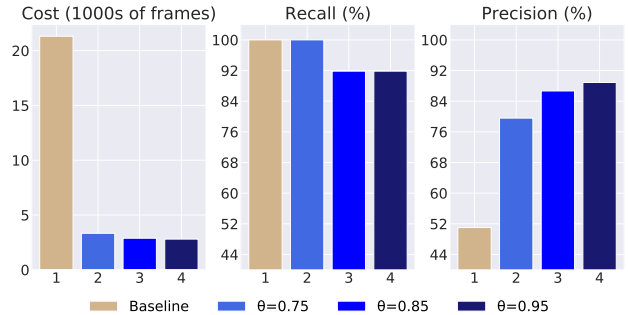


Figure 17: **Identity detection. All-camera baseline (tan) vs. three versions of ReXCam (blues) on the DukeMTMC dataset.**

ing, 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, *i.e.*, movement across cameras, however, yields substantial accuracy gains.

Efficient Machine Learning. Improving ML models using model compression [26, 42], compact architectures [31, 44], knowledge distillation [14, 24, 27], and model specialization [28, 40] is orthogonal to ReXCam, which would gain from any efficiency improvement of the models (*e.g.*, for re-id).

Unlike systems that tradeoff model resources and accuracy [19–21, 25, 30, 36, 43, 49, 66], ReXCam entails a new approach: instead of running cheaper models, we run inference on *less data* by 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 [55, 59, 60, 69], (2) new neural network architectures [54, 59, 60, 69], or (3) new training schemes [57, 60, 69, 70]. However, past computer vision work do not address the *inference cost* of re-id and MTMC tracking [16, 17, 35, 41, 48], nor does it study *online* tracking (iterated re-id), a key application of interest in camera systems.

Visual Data Management. Image and video databases explore the use of classical computer vision techniques to index video efficiently [15, 18, 22, 51, 52]. *Cross-camera inference* with CNNs on *live video* entails substantially different challenges than the target domain of these works.

Mobility Modeling. Mobility modeling and prediction has long been a topic of interest in mobile computing. Studies have shown promising results in generating human/vehicle mobility models from call detail records [33, 64], wireless signals [62], social media [38], and transactions in transportation systems [61, 64]. While none of these works apply mobility models to video analytics, ReXCam could benefit from their techniques on building accurate spatial-temporal models.

10 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 ReXCam, 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 video analytics. ReXCam 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: ReXCam reduces compute workload by $8.3\times$ 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, which we plan to expand for further experiments.

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] Azure data box edge. <https://docs.microsoft.com/en-us/azure/databox-online/data-box-edge-overview>.
- [4] British transport police: Cctv. http://www.btp.police.uk/advice_and_information/safety_on_and_near_the_railway/cctv.aspx. Accessed: 2018-10-27.
- [5] 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.
- [6] Deep person reid. <https://github.com/KaiyangZhou/deep-person-reid/>. Accessed: 2018-10-28.
- [7] Dukemtmc-reid. https://github.com/layumi/DukeMTMC-reID_evaluation. Accessed: 2018-10-28.
- [8] Genetec announces airport sense. <https://www.genetec.com/about-us/news/press-center/press-releases/>. Accessed: 2018-10-30.
- [9] Multi-target, multi-camera tracking. <https://github.com/ergysr/DeepCC>. Accessed: 2018-10-28.
- [10] 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>.
- [11] 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.
- [12] 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-surveillance>. Accessed: 2018-10-27.
- [13] Amazon. AWS DeepLens. <https://aws.amazon.com/deeplens/>, 2017.
- [14] 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.
- [15] Walid Aref, et al. Video query processing in the VDBMS testbed for video database research. In *ACM MMDB*, 2003.
- [16] Yinghao Cai and Gérard Medioni. Exploring Context Information for Inter-Camera Multiple Target Tracking. In *IEEE WACV*, 2014.
- [17] Simone Calderara, Rita Cucchiara, and Andrea Prati. Bayesian-Competitive Consistent Labeling for People Surveillance. *TPAMI*, 30, 2007.
- [18] M. La Cascia and E. Arduzzone. Jacob: Just a content-based query system for video databases. In *IEEE ICASSP*, 1996.
- [19] Tiffany Yu-han Chen. Glimpse: Continuous, Real-Time Object Recognition on Mobile Devices Categories and Subject Descriptors. In *ACM SenSys*, 2015.
- [20] 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.
- [21] Biyi Fang, Xiao Zeng, and Mi Zhang. NestDNN: Resource-Aware Multi-Tenant On-Device Deep Learning for Continuous Mobile Vision. In *ACM MobiCom*, 2018.
- [22] Myron Flickner, et al. Query by Image and Video Content: The QBIC System. *Computer*, 28, 1995.
- [23] 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.
- [24] 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.
- [25] 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.
- [26] Song Han, Huizi Mao, and William J. Dally. Deep Compression: Compressing Deep Neural Networks with Pruning, Trained Quantization and Huffman Coding. In *ICLR*, 2016.
- [27] Geoffrey Hinton, Oriol Vinyals, and Jeff Dean. Distilling the Knowledge in a Neural Network. In *NIPS*, 2014.
- [28] Kevin Hsieh, 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.
- [29] Chien-Chun Hung, Ganesh Ananthanarayanan, Peter Bodik, Leana Golubchik, Minlan Yu, Victor Bahl, and Matthai Philipose. VideoEdge: Processing Camera Streams using Hierarchical Clusters. In *ACM SEC*, 2018.
- [30] Loc N Huynh, Youngki Lee, and Rajesh K Balan. DeepMon: Mobile GPU-based Deep Learning Framework for Continuous Vision Applications. In *ACM MobiSys*, 2017.
- [31] 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.
- [32] Intel. OpenVINO. <https://github.com/opencl/OpenCL>, 2019.
- [33] 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.
- [34] Samvit Jain, Ganesh Ananthanarayanan, Junchen Jiang, Yuanchao Shu, and Joseph E. Gonzalez. Scaling Video Analytics Systems to Large Camera Deployments. In *ACM HotMobile*, 2019.
- [35] 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.
- [36] 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.
- [37] 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.
- [38] Raja Jurdak, Kun Zhao, Jiajun Liu, Maurice Abou-Jaoude, Mark Cameron, and David Newth. Understanding Human Mobility from Twitter. *PLOS ONE*, 10(7):1–16, 07 2015.
- [39] K. Jüngling, C. Bodensteiner, and M. Arens. Person re-identification in multi-camera networks. In *IEEE CVPR*, 2011.
- [40] Daniel Kang, John Emmons, Firas Abuzaid, Peter Bailis, and Matei Zaharia. NoScope: Optimizing Neural Network Queries over Video at Scale. In *VLDB*, 2017.
- [41] Cheng-Hao Kuo, Chang Huang, and Ram Nevatia. Inter-camera Association of Multi-target Tracks by On-Line Learned Appearance Affinity Models. In *ECCV*, 2010.
- [42] Hao Li, Asim Kadav, Igor Durdanovic, Hanan Samet, and Hans Peter Graf. Pruning Filters for Efficient ConvNets. In *ICLR*, 2017.
- [43] Robert LiKamWa and Lin Zhong. Starfish: Efficient Concurrency Support for Computer Vision Applications. In *ACM MobiSys*, 2015.
- [44] Min Lin, Qiang Chen, and Shuicheng Yan. Network in network. In *ICLR*, 2014.
- [45] Franz Loewenherz, Victor Bahl, and Yinhai Wang. Video analytics towards vision zero. In *ITE Journal*, 2017.
- [46] Yao Lu, Aakanksha Chowdhery, and Srikanth Kandula. Optasia: A Relational Platform for Efficient Large-Scale Video Analytics. In *ACM SoCC*, 2016.
- [47] Mahadev Satyanarayanan, Victor Bahl, Ramon Cáceres, Nigel Davies. The Case for VM-based Cloudlets in Mobile Computing. In *IEEE Pervasive Computing*, 2009.

- [48] Dimitrios Makris, Tim Ellis, and James Black. Bridging the gaps between cameras. In *CVPR*, 2004.
- [49] 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.
- [50] 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.
- [51] V. E. Ogle and M. Stonebraker. Chabot: Retrieval from a relational database of images. *Computer*, 28, 1995.
- [52] JungHwan Oh and Kien A. Hua. Efficient and cost-effective techniques for browsing and indexing large video databases. In *ACM SIGMOD*, 2000.
- [53] Qualcomm. Vision Intelligence Platform. <https://www.qualcomm.com/news/releases/2018/04/11/qualcomm-unveils-vision-intelligence-platform-purpose-built-iot-devices>, 2018.
- [54] 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.
- [55] 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.
- [56] Ergys Ristani and Carlo Tomasi. Features for Multi-Target Multi-Camera Tracking and Re-Identification. In *IEEE CVPR*, 2018.
- [57] Ergys Ristani and Carlo Tomasi. Features for Multi-Target Multi-Camera Tracking and Re-Identification. In *IEEE CVPR*, 2018.
- [58] Xiaogang Wang. Intelligent multi-camera video surveillance: A review. *Pattern recognition letters*, 34(1):3–19, 2013.
- [59] Longhui Wei, Shiliang Zhang, Wen Gao, and Qi Tian. Person Transfer GAN to Bridge Domain Gap for Person Re-Identification. In *IEEE CVPR*, 2018.
- [60] Tong Xiao, Shuang Li, Bochao Wang, Liang Lin, and Xiaogang Wang. Joint Detection and Identification Feature Learning for Person Search. In *IEEE CVPR*, 2017.
- [61] 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.
- [62] Jungkeun Yoon, Brian D. Noble, Mingyan Liu, and Minkyong Kim. Building Realistic Mobility Models from Coarse-grained Traces. In *ACM MobiSys*, 2006.
- [63] Ben Zhang, Xin Jin, Sylvia Ratnasamy, John Wawrzynnek, 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.
- [64] 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.
- [65] 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.
- [66] 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.
- [67] 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.
- [68] Liang Zheng, Yi Yang, and Alexander G Hauptmann. Person Re-identification: Past, Present and Future. *arXiv:1610.02984*, 2015.
- [69] Liang Zheng, Hengheng Zhang, Shaoyan Sun, Manmohan Chandraker, Yi Yang, and Qi Tian. Person Re-identification in the Wild. In *IEEE CVPR*, 2017.
- [70] Zheng Zhu, Wei Wu, Wei Zou, and Junjie Yan. End-to-End Flow Correlation Tracking With Spatial-Temporal Attention. In *IEEE CVPR*, 2018.