Resource-Efficient Cross-Camera Video Analytics

Samvit Jain
Mentors: Yuanchao Shu, Ganesh Ananthanarayanan, Junchen Jiang

Introduction

Two trends

- Falling camera costs
- Advances in computer vision

Enterprise deployments

- Increase in network size and density
- Compute in edge cluster or on-board
- Feeds analyzed separately
  - Cost grows linearly

Spotlight search

Track person P through camera network

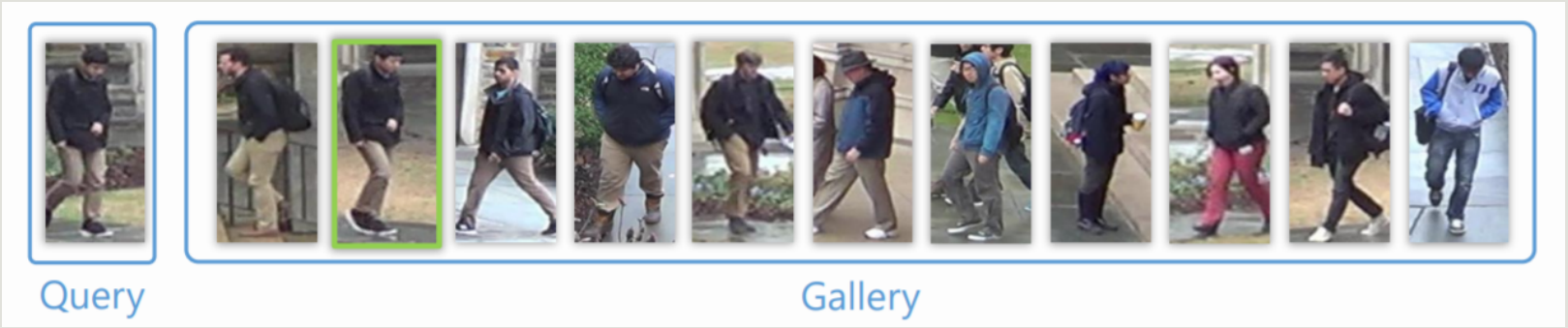
- Real-time monitoring
- Investigative search

Goals

- High accuracy – return all frames in which person is present; keep false positives low
- Low resource usage – only search cameras ‘likely’ to contain person (spotlight search)

Person re-identification

Given a query image, rank images in “gallery” by decreasing order of similarity



Ranking

$$[sim(\,,\,) > sim(\,$$

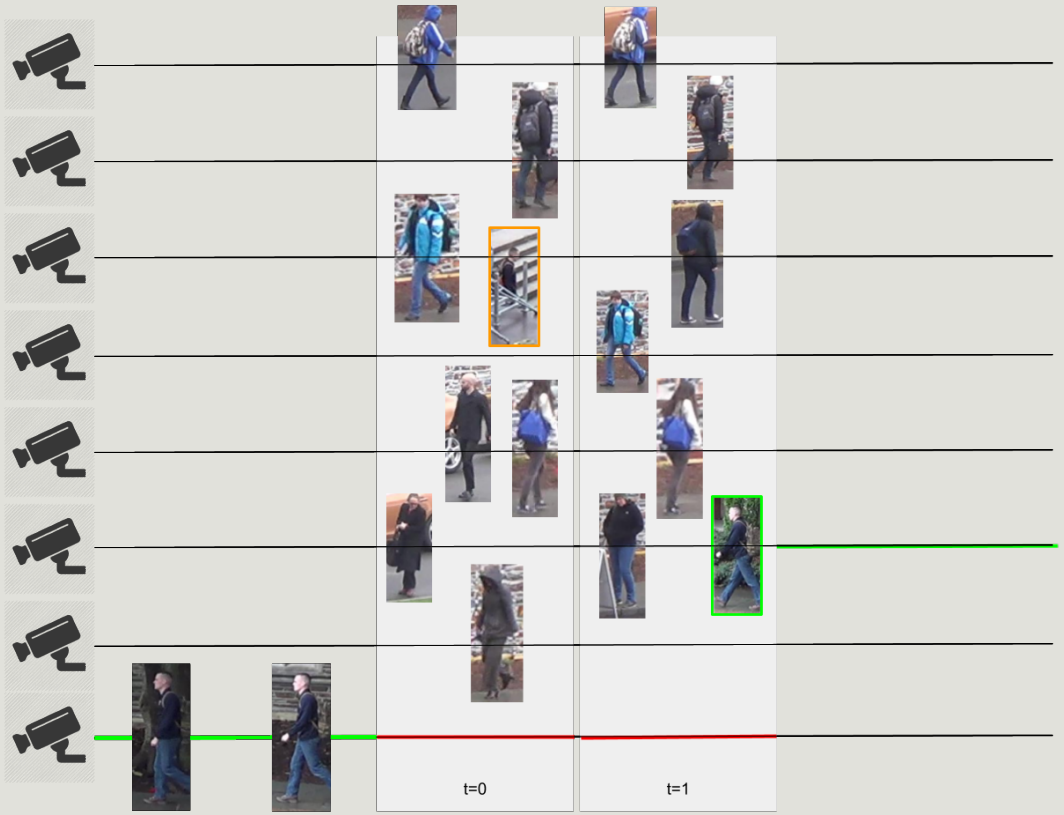
Same person

Varying appearance

Different people

Similar appearance

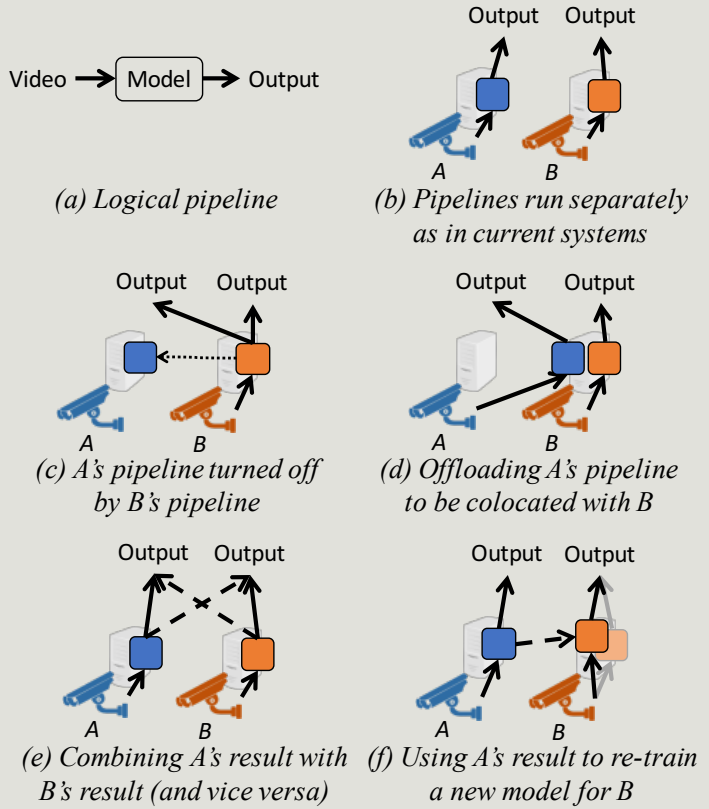
From re-id to tracking



Opportunities

Cost-efficiency
- Inference pruning (c)
- Resource pooling (d)
- Collaborative adaptation

Higher accuracy
- Collaborative inference (e)
- Model refinement (f)



Prior work

Person search and tracking
- Person re-identification
- Multi-target, multi-camera tracking (MTMCT)

Resource-efficient video analytics
- VideoStorm
- NoScope
- Focus

Focus: Querying Large Video Datasets with Low Latency and Low Cost

Kevin Hsieh[†§] Ganesh Ananthanarayanan[§] Peter Bodik[§] Paramvir Bahl[§] Matthai Philipose[§]
Phillip B. Gibbons[†] Onur Mutlu[*†]
[†]Carnegie Mellon University [§]Microsoft [*]ETH Zürich

Abstract

Large volumes of videos are continuously recorded from cameras deployed for traffic control and surveillance with the goal of answering "after the fact" queries: *identify video frames with objects of certain classes (cars, bags)* from many days of recorded video. While advancements in convolutional neural networks (CNNs) have enabled answering such queries with high accuracy, they are too expensive and slow. We build Focus, a system for low-latency and low-cost querying on large video datasets. Focus uses cheap ingestion techniques to index the videos by the objects occurring in them. At ingest-time, it uses compression and video-specific specialization of CNNs. Focus handles the lower accuracy of the cheap CNNs by judiciously leveraging expensive CNNs at query-time. To reduce query time latency, we cluster similar objects and hence avoid redundant processing. Using experiments on video streams from traffic, surveillance and news channels, we see that Focus uses 58× fewer GPU cycles than running expensive ingest processors and is 37× faster than processing all the video at query time.

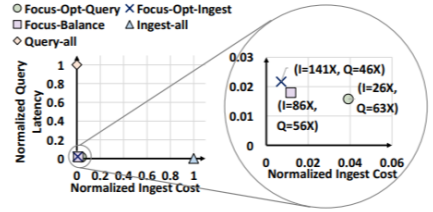
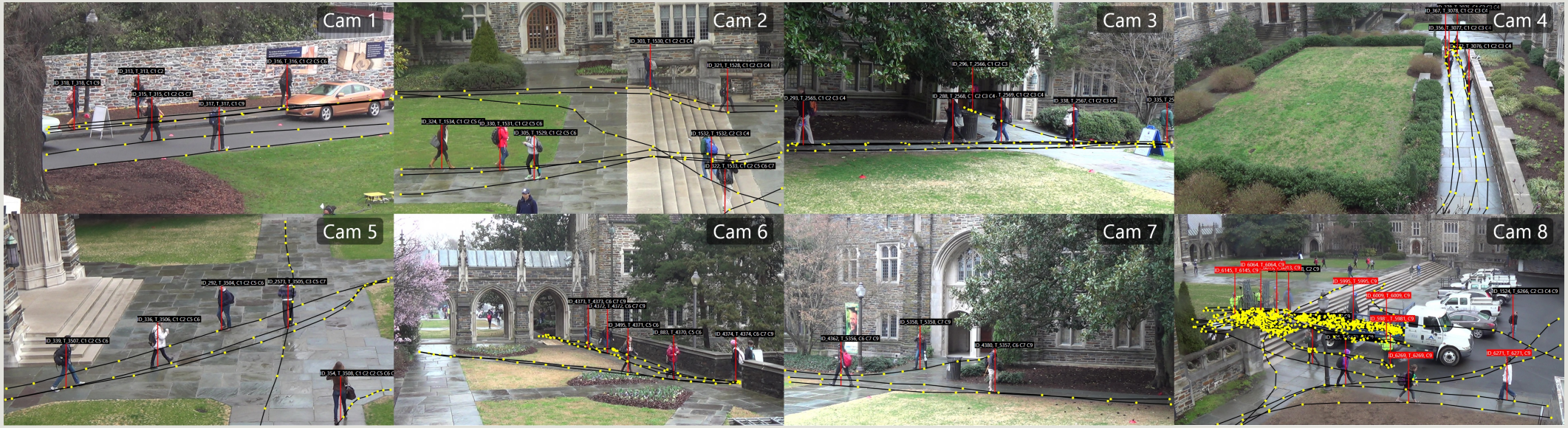


Figure 1: Effectiveness of Focus at reducing both ingest cost and query latency, for an example traffic video. We compare against two baselines: "Ingest-all" that runs ResNet152 on all video frames during ingest, and "Query-all" that runs ResNet152 on all the video frames at query time. By zooming in, we see that Focus (the Focus-Balance point) is simultaneously 86× cheaper than Ingest-all in its GPU consumption *and* 56× faster than Query-all in query latency, all the while achieving at least 95% precision and recall. (Also shown are two alternatives offering slightly different trade-offs.)

1. Introduction

Cameras are ubiquitous, with millions of them deployed by government and private entities at traffic intersections, ResNet152), using them for video analytics queries is both expensive and slow. Using the ResNet152 classifier at *query-time* to identify video frames with cars on a month-long traffic video requires 280 GPU hours and costs $250 in the Azure cloud. The latency for running queries is also high. To achieve a query latency of one

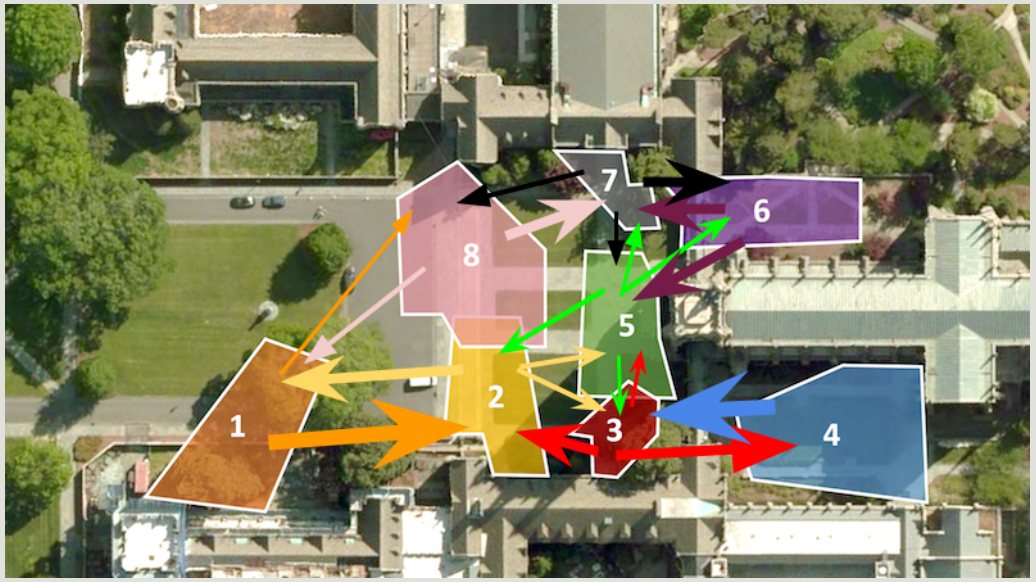
Duke MTMC dataset

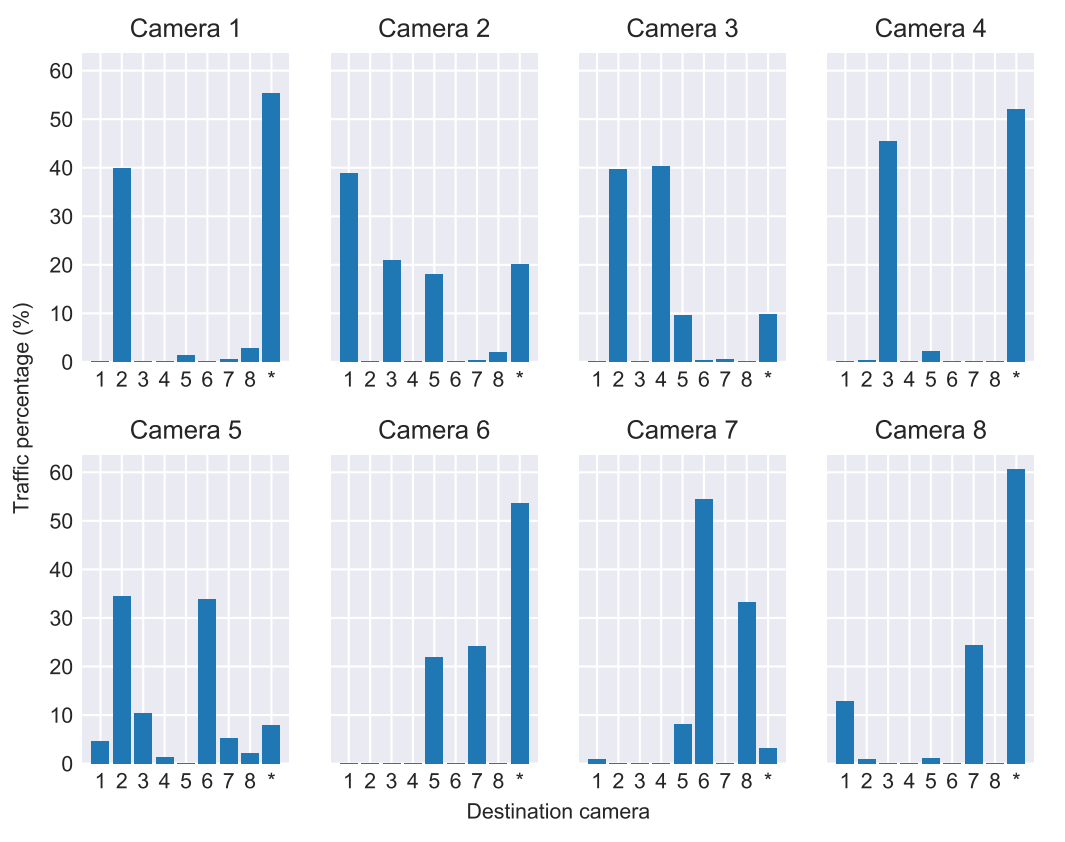


Duke MTMC dataset

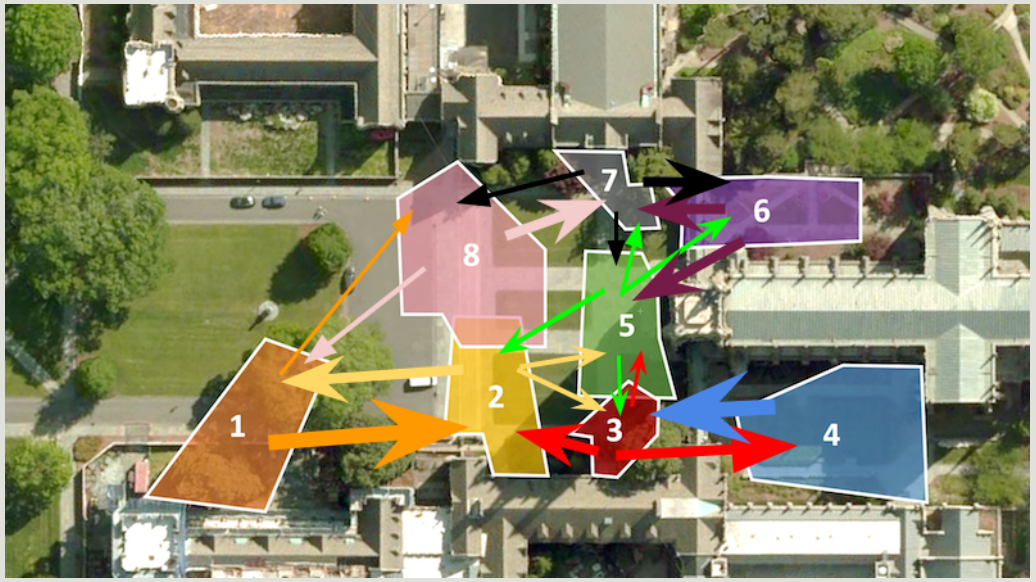
Source: https://github.com/ergysr/DeepCC

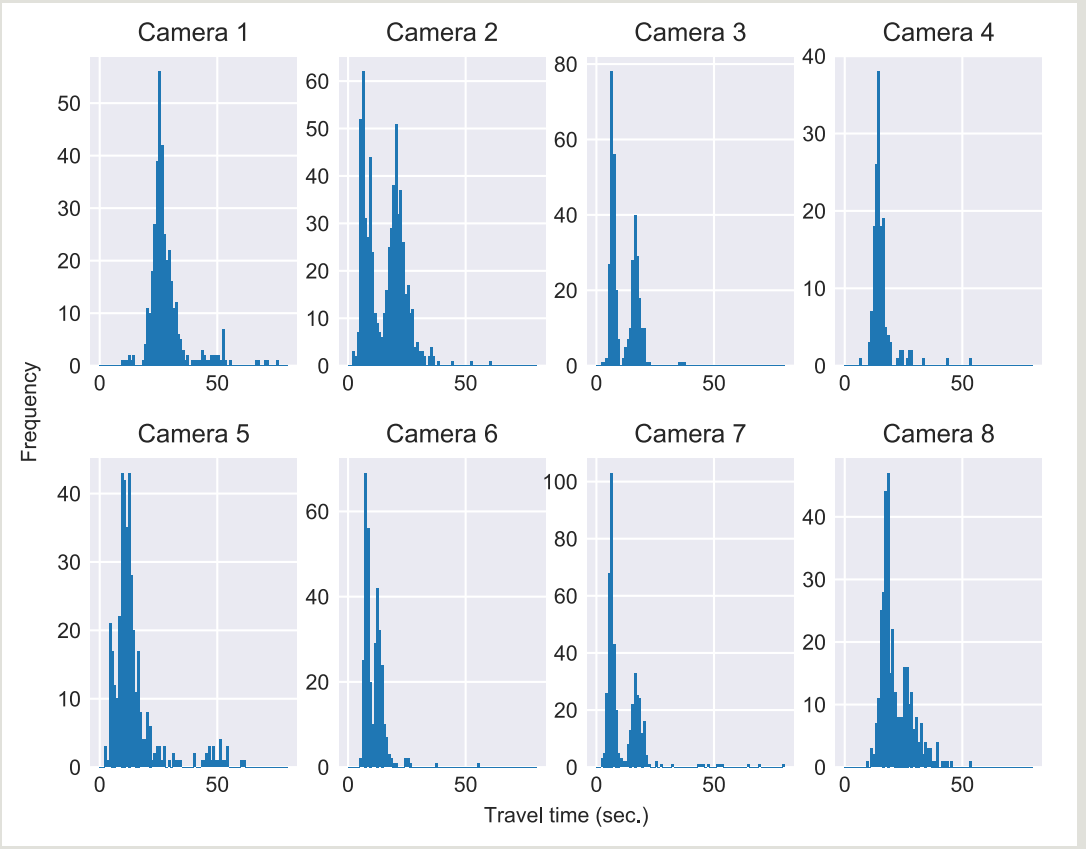
Spatial traffic patterns





Temporal traffic patterns

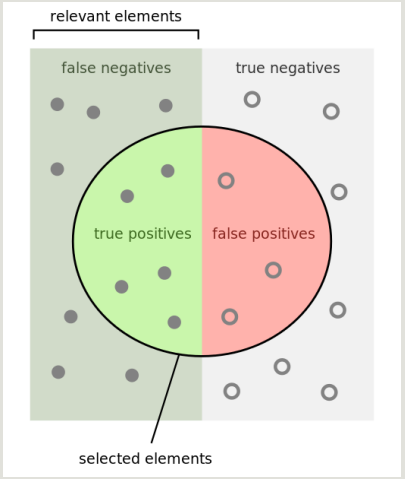




Metrics

Accuracy

- Recall – correct identifications of **q** (TP) / all instances of **q** in dataset (TP + FN)
- Precision – correct identifications of **q** (TP) / all identifications (TP + FP)



Resource usage

- Speedup – detections processed by baseline / detections processed by model

Delay

- Total lag – total delay (in sec) accumulated by tracker relative to video feed
- Average lag – average gap (in sec) between tracker and video feed over tracking duration

Algorithm

For query q:

- curr_q = features(q)
- while !should_exit:
  - frames = get_frames(video_feeds, time_window)
  - frames = filter(frames, spatial_filter, temporal_filter)
  - ranked = rank(curr_q, frames)
  - if ranked[0][score] < match_threshold:

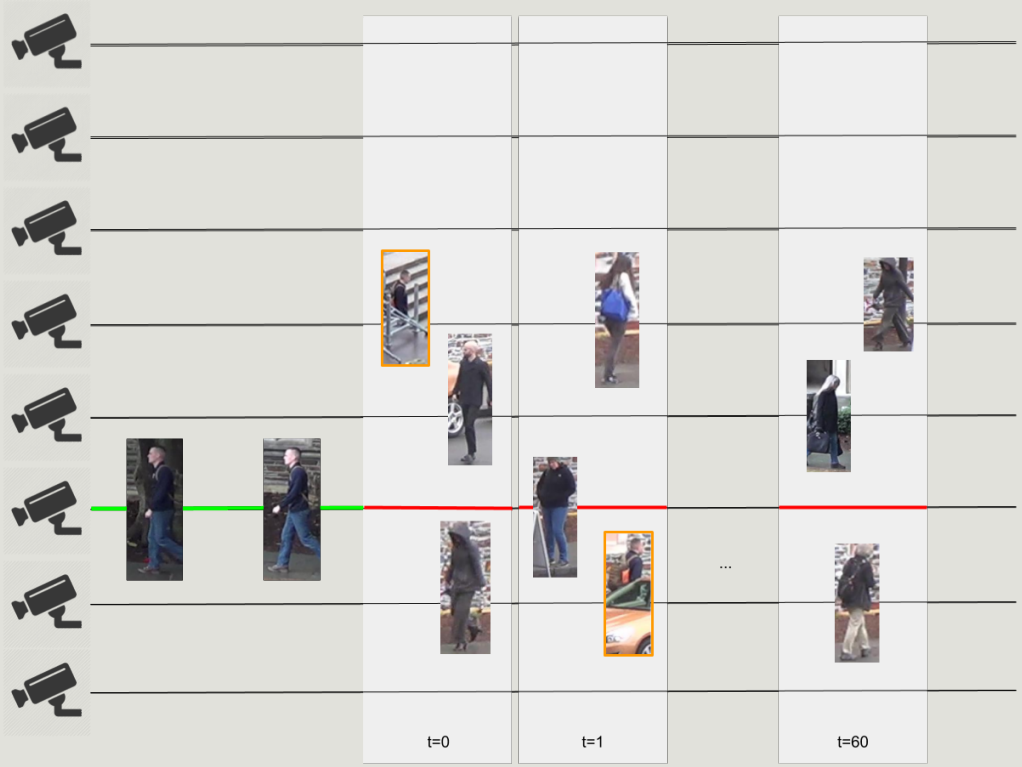
    // New query instance declared!
    - curr_q = update(curr_q, ranked[0][features] // update features
    - time_window = reset(time_window) // reset tracking
  - else:
    - time_window = increment(time_window) // go to next window
  - should_exit = check_exit_condition(time_window)

Algorithm

For query q:

- curr_q = features(q)
- while !should_exit:
  - frames = get_frames(video_feeds, time_window)
  - frames = filter(frames, spatial_filter, temporal_filter)
  - ranked = rank(curr_q, frames)
  - if ranked[0][score] < match_threshold:

    // New query instance declared!
    - curr_q = update(curr_q, ranked[0][features] // update features
    - time_window = reset(time_window) // reset tracking
  - else:
    - time_window = increment(time_window) // go to next window
  - should_exit = check_exit_condition(time_window)



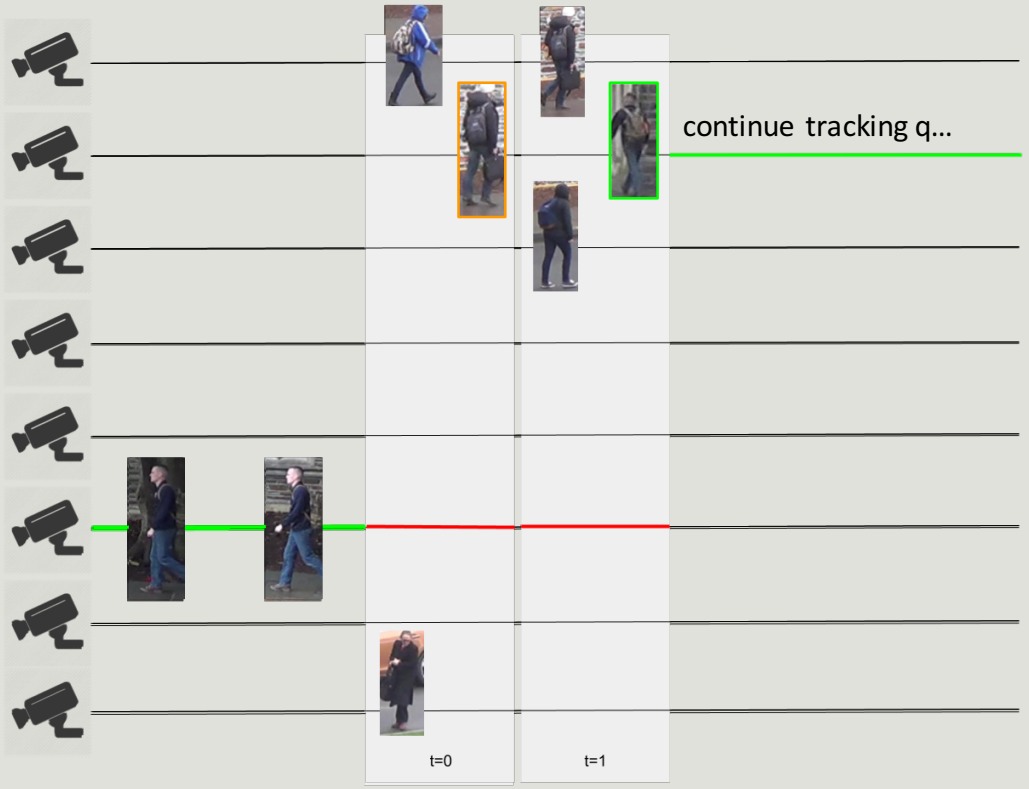
declare exit

Algorithm - Phase 2

For query q:

- curr_q = features(q)
- while !should_exit:
  - frames = get_frames(video_feeds, time_window)
  - frames = filter(frames, !spatial_filter, temporal_filter)
  - ranked = rank(curr_q, frames)
  - if ranked[0][score] < match_threshold:

    // New query instance declared!
    - curr_q = update(curr_q, ranked[0][features]) // update features
    - time_window = reset(time_window) // reset tracking
  - else:
    - time_window = increment(time_window) // go to next window
  - should_exit = check_exit_condition(time_window)



Implementation

Spotlight search

- PyTorch re-identification framework
- Built tracking layer

Hyper-parameters

match threshold, feature weights, algorithm meta-structure

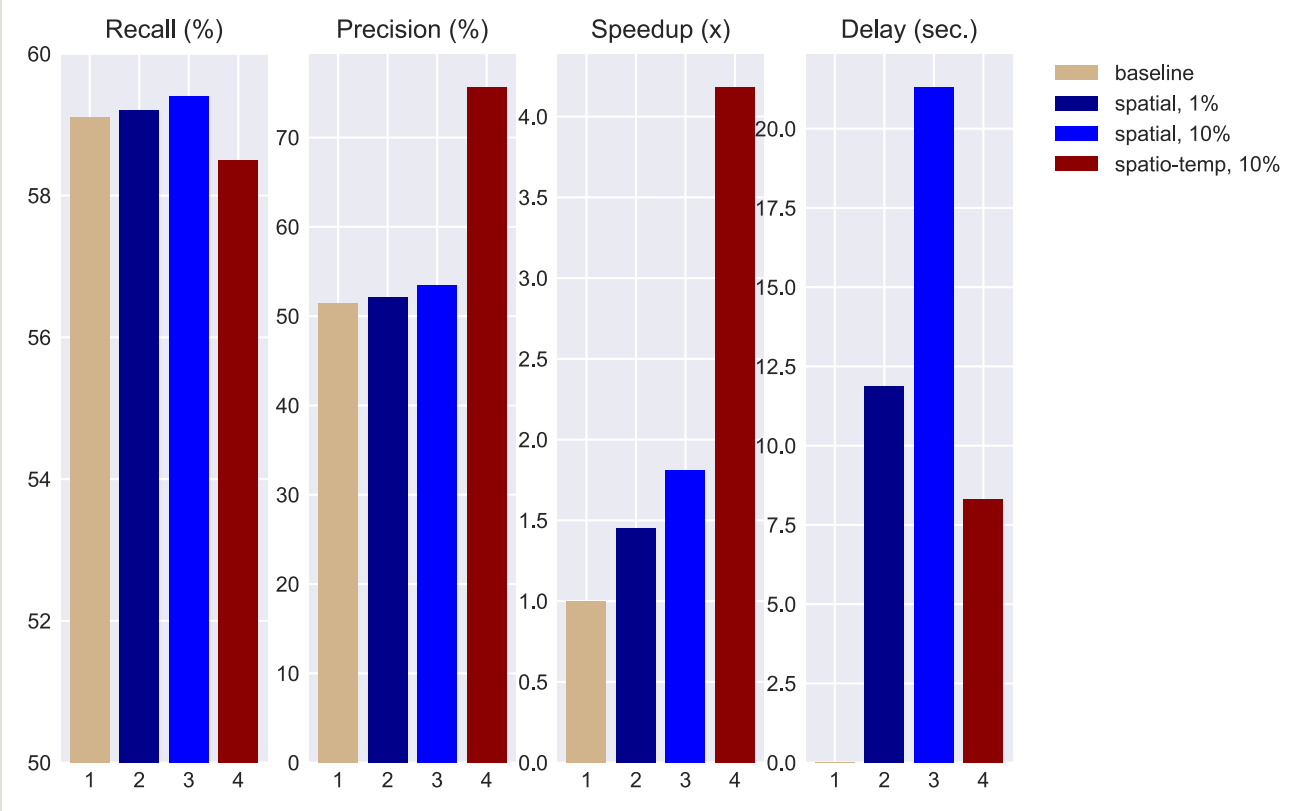
Spatio-temporal pruning

- Extracted statistics from dataset labels
- For each camera **i**, compiled:
  - List of correlated cameras — cameras receiving ≥ 10% of traffic
  - Arrival **start times** for each destination camera **j** — time **t** such that 100% of traffic appears *after* **t**
  - Arrival **end times** for each destination camera **j** — time **t** such that 99% of traffic appears *before* **t**

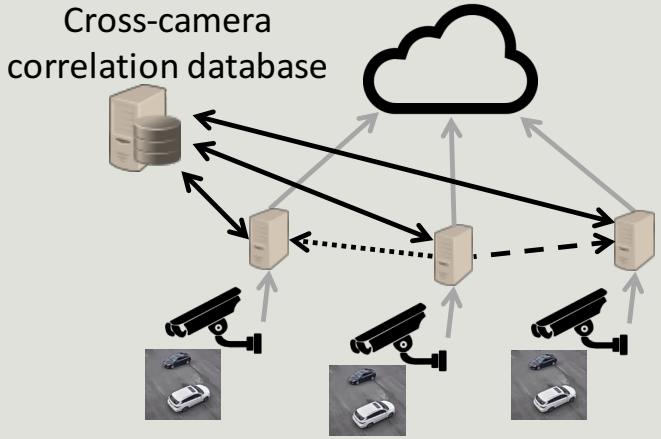
Results

Four schemes

- Baseline – searches all 8 cameras, from t=0 to t=exit_time
- Spatial filter, 1% – first searches cameras expecting ≥1% of traffic
- Spatial filter, 10% – first searches cameras expecting ≥10% of traffic
- Spatio-temporal filter – only searches cameras during time window containing [0%, 99%] of hist. traffic



Proposed architecture

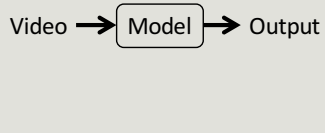


Per-camera pipeline

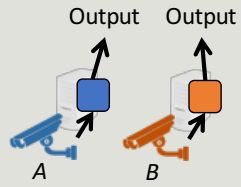
Cross-camera correlation info

Triggering reconfig.

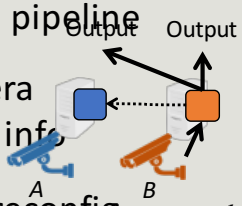
Compositing pipelines



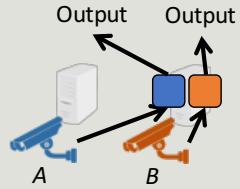
(a) Logical pipeline



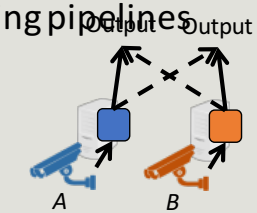
(b) Pipelines run separately as in current systems



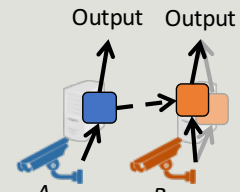
(c) A's pipeline turned off by B's pipeline



(d) Offloading A's pipeline to be colocated with B



(e) Combining A's result with B's result (and vice versa)



(f) Using A's result to re-train a new model for B

Next steps…

Spatio-temporal model

Questions

- Can we build a spatio-temporal model, without using labelled data?
  - Yes, using multi-camera, multi-target tracking (“track all the people”)
- How often must this model be updated?
  - Whenever the test data distribution shifts substantially
- How expensive is it to build/update this model?
  - As expensive as it is to label the training data (i.e. identify all distinct trajectories)

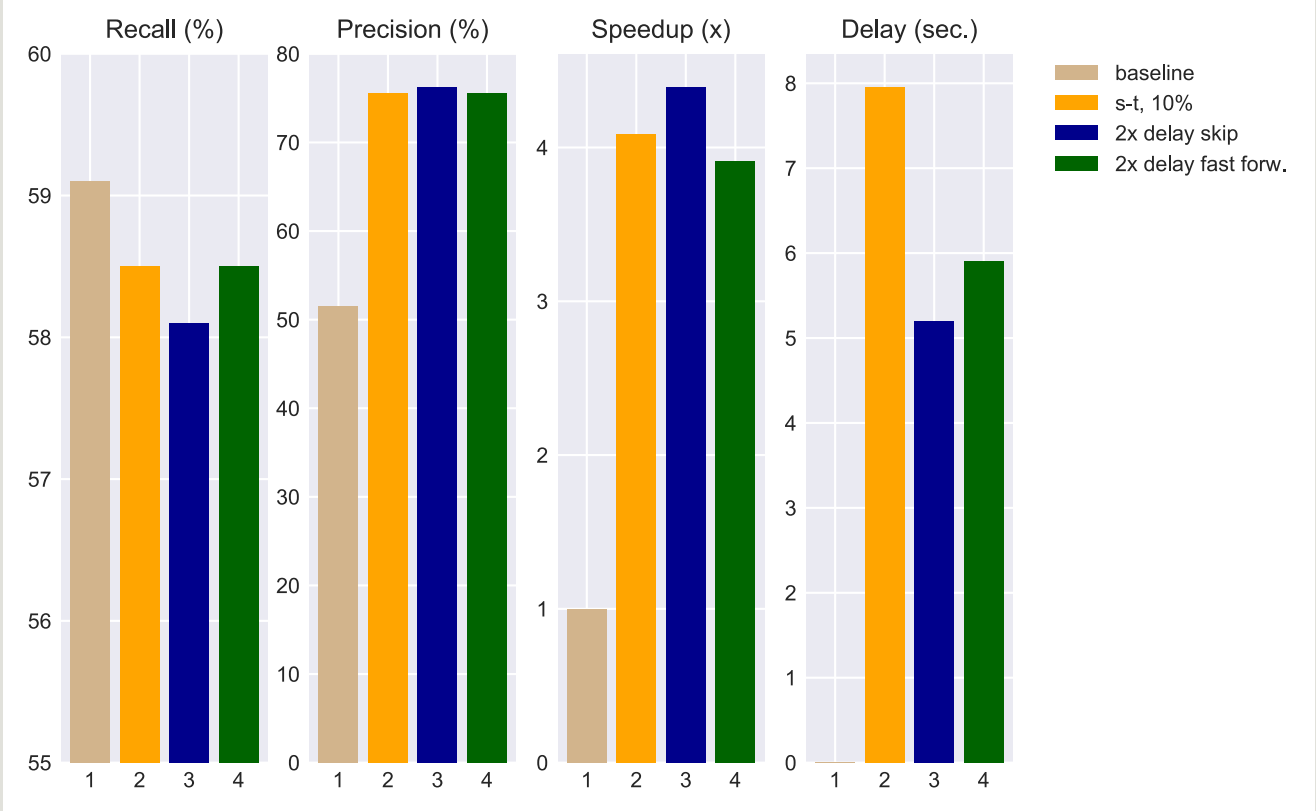
Delay mitigation

Two approaches

- Skip frames – lower frame sampling rate on historical search
- Fast forward – boost processing throughput on historical search

Available tradeoffs

- Lower accuracy – lower frame sampling rate → missed detections
- Lower speedup factor – higher processing rate → lower savings



Scaling

Q: Does speedup factor scale with the number of cameras in the dataset?

- True if: number of correlated cameras does NOT increase with camera deployment size

Thank you!

Scaling Video Analytics Systems to Large Camera Deployments

Paper # 173

ABSTRACT

New computer vision techniques, which enable accurate extraction of insights from videos, and the deployment of cameras *en masse* have made many previously inconceivable applications possible. Scaling video analytics to massive camera deployments, however, presents a tremendous challenge, as cost and resource consumption grow proportionally to the number of camera feeds. This paper is driven by a simple question: can we scale video analytics in such a way that *cost grows sublinearly* (or even remains constant) as we deploy more cameras, *while the accuracy of the analytics remains stable* (or even improves)? We believe the answer is yes. Our key insight is that as cameras are densely installed in a physical space, their video feeds become increasingly *correlated* with each other, correlations that can be harnessed to improve the cost efficiency and accuracy of multi-camera video analytics. By explaining use-cases and technical challenges, we shed light on the potential of leveraging *cross-camera correlations* to scale out video analytics to many cameras, and call on the community to unlock this potential.

1 Introduction

The whole is greater than the sum of its parts.
Aristotle

Driven by plummeting camera prices and advances in computer vision (DNNs), organizations are deploying cameras at scale for applications ranging from surveillance (*e.g.,* airport or campus security) to flow control (*e.g.,* at traffic lights, building entrances) and retail planning (*e.g.,* inventory management) [29, 21, 41]. Processing the video feeds from these large deployments, using new analytics techniques, however, is becoming prohibitively resource intensive.

A key reason for the rising costs is the fact that in today's video analytics systems, video streams are analyzed *independently*. As a result, the compute required to process the videos grows linearly with the number of cameras. We believe there is an opportunity to both stem this trend of linearly increasing costs, and improve the accuracy of the video analytics, by viewing the cameras *collectively*. Our initial evaluation on a real-world dataset shows that using cameras collectively can yield resource savings of at least 51%, while also improving inference accuracy.

This position paper is based on a simple observation—the dense deployment of cameras over a physical area means that their video feeds are increasingly *correlated*, spatially and temporally. For instance, they may see the same objects, though from different angles or at different points in time.

Our key insight is that these cross-camera correlations can be harnessed, so as to use *substantially lower resources* and/or achieve *higher inference accuracy* than a system that runs complex inference on all video feeds independently. For example, when searching for a specific person, if that person is identified in one camera feed, we can then exclude the possibility of the person appearing in a far-away camera within a short time period, eliminating extraneous queries and reducing false positive detection (Figure 1(a)). Similarly, one can improve accuracy by combining the perspectives of multiple cameras that monitor the same objects at different angles (Figure 1(b)). Moreover, one can opportunistically utilize the compute resources of multiple cameras to collectively alleviate hot spots in the inherently dynamic video analytics workloads. More such opportunities are outlined in §3.

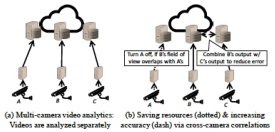


Figure 1: ***Contrasting (a) the traditional per-camera video analytics with (b) the proposed approach that leverages cross-camera correlations.***

With the recent increase in interest for systems infrastructure for video analytics [38, 8, 18, 32, 16], we believe the important next step for the community is designing video analytics stacks for cross-camera applications over a collection of cameras. Current video analytics systems generally analyze video streams independently *even while* useful cross-camera correlations exist [38, 18, 16]. The computer vision literature has optimized for specific applications over a group of cameras (*e.g.,* tracking [29, 41, 35]), but has failed to address the growing cost of inference itself.

We propose to re-architect the video analytics stack so that it can scale to many cameras by fully leveraging these cross-camera correlations. We identify several architectural aspects that are critical to improving resource efficiency and accuracy but are missing in current video analytics systems. Firstly, we illustrate the need for a new module that dynamically generates and maintains up-to-date *spatio-temporal correlations* across cameras. Secondly, we discuss online pipeline reconfiguration and composition, where video pipelines incorporate information from other correlated cameras (*e.g.,* eliminating redundant inference to save cost, or serving as an model ensemble to improve accuracy). Finally, we note opportunities to use cameras collaboratively. We defer the details and discussion of practical challenges to §4.

Moving forward, we will assume that the cameras have the compute (CPU, GPU) and memory (RAM, SSD) resources to run video analytics modules, and the network connectivity and bandwidth to communicate with each other [6, 25, 3].

3.2 Better cost efficiency

Leveraging cross-camera correlations improves the *cost efficiency* of multi-camera video analytics—one can use significantly fewer resources than a system that runs expensive analytics separately on each feed, without reducing accuracy.

C1: Eliminating redundant inference

In cross-camera applications like spotlight search, there are often far fewer objects of interest than cameras. Hence, ideally, query resource consumption over multiple cameras should not grow proportionally to the number of cameras. We envision two potential ways of doing this by leveraging content-level correlations across cameras (§3.1).

- When two spatially correlated cameras have (partly) overlapping views (*e.g.,* cameras in the same room or covering the same hallway), the overlapped region need only be analyzed once.
- When an object leaves a camera, only a small set of relevant cameras (*e.g.,* cameras likely to see the object in the next few seconds), identified via their spatio-temporal correlations, need search for the object.

In spotlight search, for example, once a suspicious activity is detected and the individual to be tracked is identified, we can selectively trigger multi-class object detection or person re-identification models only on the cameras that the individual is likely to traverse. In other words, we can use spatio-temporal correlations to *forecast* the trajectory of objects.

We analyze the popular "DukeMTMC" video dataset [28], which contains footage from eight cameras on the Duke University campus. Figure 2 shows a map of the different cameras, along with the percentage of traffic leaving a particular camera i that next appears in another camera j. Figures are calculated based on manually annotated human identity labels. As an example observation, within a time window of 90 minutes, 89% of all traffic leaving Camera 1 appears in Camera 2. At Camera 3, an equal percentage of traffic, about 45%, leaves for Cameras 2 and 4. Gains achieved by leveraging these spatial traffic patterns are discussed in §3.4.

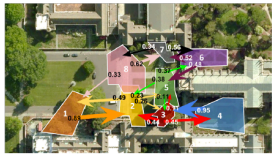


Figure 2: ***Camera topology and traffic flow in the DukeMTMC dataset.***

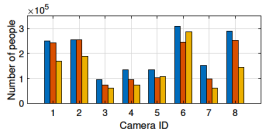


Figure 3: ***Number of people detections on different cameras in the DukeMTMC dataset.***

C2: Pooling resources across cameras

Since objects/activities of interest are usually sparse, most cameras do not need to run analytics models all the time. Instead, models can be intelligently triggered on demand. This creates a substantial heterogeneity in workloads across different cameras. For instance, one camera may monitor a central hallway and detect many candidate persons, while another camera detects no people in the same time window.

Such workload heterogeneity provides an opportunity for dynamic offloading, in which more heavily utilized cameras transfer part of their analytics load to less-utilized cameras. For instance, a camera that runs complex per-frame inference can offload queries on some frames to other "idle" cameras whose video feeds are static. By pooling available resources and balancing the workloads among multiple cameras, one can greatly reduce the amount of resources provisioned on each camera (*e.g.,* size/number of GPUs), from an allocation that would support peak workloads. Such a scheme could also reduce (or altogether eliminate) the need to relegate compute loads to the cloud, a capability constrained by available bandwidth, and in some applications, privacy concerns. *Content* correlations, §3.1 directly facilitate this offloading as they foretell query trajectories, and by extension, workloads on different cameras.

Figure 3 shows the total number of people detected at 60 fps over three consecutive 16.5 minute intervals. Evidently, there is a huge imbalance in traffic both over time and across cameras. Notably, provisioning compute resources based on ideal load balancing here (*i.e.,* splitting the entire workload equally across each camera and time interval), as opposed to peak workload without temporal and cross-camera load balancing, would offer *potential resource savings of* 89.5%.

C3: Collaborative adaptation

Recent work has shown that varying the configurations of video analytics pipelines (*e.g.,* frame rate, resolution) can substantially reduce the resource consumption (by up to two orders of magnitude [38]) with little drop in accuracy; *e.g.,* a low frame rate is adequate to track vehicles during peak or a hybrid system. Different correlations can be represented in various ways. For example, content correlations can be modeled as the *conditional probability* of detecting a specific object in camera B at time t, given its appearance at time $t - \Delta t$ in camera A, and stored as a discrete, 3-D matrix in a database. This database of cross-camera correlations must be dynamically updated, because the correlations between cameras can vary over time for various reasons: video patterns can evolve, cameras can enter or leave the system, and camera positions and viewpoints may change with time. We discuss the intricacy of discovering these correlations, and the implementation of this new module, in §4.2.

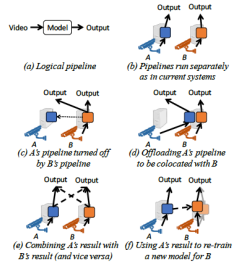


Figure 4: ***Illustrative examples of peer-triggered reconfiguration (c, d) and pipeline composition (e, f) using an example logical pipeline (a) running on two cameras (b).***

#2: Peer-triggered reconfiguration

Today, the execution of a video analytics pipeline (what resources to use and which video to analyze) is largely preconfigured. To take advantage of cross-camera correlations, however, an analytics pipeline must be aware of the inference results of other relevant video streams, and support *peer-triggered reconfiguration* at runtime. Depending on the content of other related video streams, an analytics task can be assigned to the computing resources of *any* relevant camera to process *any* video stream at *any* time. This effectively separates the logical analytics pipeline from its execution. To eliminate redundant inference (C1 of §3.2), for instance, one video stream pipeline may need to dynamically trigger (or switch off) another video pipeline (Figure 4(c). Similarly, to pool resources across cameras (C2 of §3.2), a video stream may need to dynamically offload computation to another camera, depending on correlation-based workload projections (Figure 4(d). To trigger these reconfigurations, inference results need to be shared in real-time *between* pipelines.

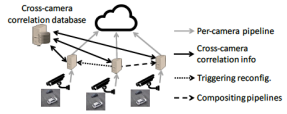


Figure 5: ***End-to-end, cross-camera analytics architecture***

While prior work explores task offloading across cameras and between the edge and the cloud [38, 3, 19], the trigger is usually workload changes on a single camera. In contrast, we argue that such reconfigurations must also consider events on the video streams of other, related cameras.

#3: Video pipeline composition

Analyzing each video stream in isolation also precludes learning from the content of other camera feeds. As we noted in §3.3, by combining the inference results of multiple correlated cameras, *i.e.,* composing multiple video pipelines, one can significantly improve inference accuracy. Figure 4 shows two examples. Firstly, by sharing inference results across pipelines in real-time (Figure 4(e), one can correct the inference error of another less well-positioned camera (A 1 in §3.3). Secondly, the inference model for one pipeline can be refined/retrained (Figure 4(f) based on the inference results of another better positioned camera (A2 in §3.3). Unlike the aforementioned reconfiguration of video pipelines, *merging* pipelines in this way actually impacts pipeline output.

#4: Fast analytics on stored video

Recall from §2 that spotlight search can involve tracking an object *backward* for short periods of time to its first appearance in the camera network. This requires a new feature, absent from most video stream analytics systems: fast analysis of stored (past) video data, *in parallel* with live video streams. Stored video must be processed with very low latency (*e.g.,* several seconds), as subsequent tracking decisions depend on the results of the search. In particular, this introduces a new requirement: processing many seconds or minutes of stored video at *faster-than-real-time* rates.

Putting it all together: Figure 5 depicts a new video analytics system that incorporates these proposed changes, along with two new required interfaces. Firstly, the correlation database must expose an interface to the analytics pipelines that reveals the spatio-temporal correlation between any two cameras. Secondly, pipelines must support an interface for real-time communication, to (1) trigger reconfiguration (C1 in §3.2) and (2) share inference results (A1 and A2 in §3.3). This channel can be extended to support the sharing of resource availability (C2) and optimal configurations (C3).

4.2 Technical challenges

In this section, we highlight the technical challenges that must be resolved to fully leverage cross-camera correlations.