Caesar: Cross-camera Complex Activity Recognition

Xiaochen Liu
University of Southern California
liu851@usc.edu

Pradipta Ghosh
University of Southern California
pradiptg@usc.edu

Oytun Ulutan
University of California, Santa Barbara
ulutan@ece.ucsb.edu

B.S. Manjunath
University of California, Santa Barbara
manj@ece.ucsb.edu

Kevin Chan
ARL
kevin.s.chan.civ@mail.mil

Ramesh Govindan
University of Southern California
ramesh@usc.edu

ABSTRACT

Detecting activities from video taken with a single camera is an active research area for ML-based machine vision. In this paper, we examine the next research frontier: near real-time detection of *complex activities* spanning *multiple* (possibly wireless) cameras, a capability applicable to surveillance tasks. We argue that a system for such complex activity detection must employ a *hybrid* design: one in which rule-based activity detection must complement neural network based detection. Moreover, to be practical, such a system must scale well to multiple cameras and have low end-to-end latency. Caesar, our edge computing based system for complex activity detection, provides an extensible vocabulary of activities to allow users to specify complex actions in terms of spatial and temporal relationships between actors, objects, and activities. Caesar converts these specifications to graphs, efficiently monitors camera feeds, partitions processing between cameras and the edge cluster, retrieves minimal information from cameras, carefully schedules neural network invocation, and efficiently matches specification graphs to the underlying data in order to detect complex activities. Our evaluations show that Caesar can reduce wireless bandwidth, on-board camera memory, and detection latency by an order of magnitude while achieving good precision and recall for all complex activities on a public multi-camera dataset.

CCS CONCEPTS

• **Information systems** → **Information systems applications**; • **Networks**; • **Computing methodologies** → **Computer vision**;

KEYWORDS

Action Detection, Computer Vision, Mobile Sensing, Camera Networks, Edge Computing

ACM Reference Format:
Xiaochen Liu, Pradipta Ghosh, Oytun Ulutan, B.S. Manjunath, Kevin Chan, and Ramesh Govindan. 2019. Caesar: Cross-camera Complex Activity Recognition. In *SenSys '19: Conference on Embedded Networked Sensor Systems, November 10–13, 2019, New York, NY, USA.* ACM, New York, NY, USA, 13 pages. https://doi.org/10.1145/3356250.3360041

The research was sponsored by the Army Research Laboratory with the Cooperative Agreement Number W911NF-09-2-0053 (the ARL Network Science CTA). The views and conclusions contained in this document are those of the authors and should not be interpreted as representing the official policies, either expressed or implied, of the Army Research Laboratory or the U.S. Government. The U.S. Government is authorized to reproduce and distribute reprints for Government purposes notwithstanding any copyright notation here on. This work was also supported in part by the CONIX Research Center, one of six centers in JUMP, a Semiconductor Research Corporation (SRC) program sponsored by DARPA.

1 INTRODUCTION

Being able to automatically detect activities occurring in the view of a single camera is an important challenge in machine vision. The availability of action data sets [1, 12] has enabled the use of deep learning for this problem. Deep neural networks (DNNs) can detect what we call *atomic actions* occurring within a single camera. Examples of atomic actions include "talking on the phone", "talking to someone else", "walking" *etc.*

Prior to the advent of neural networks, activity detection relied on inferring spatial and temporal relationships between objects. For example, consider the activity "getting into a car", which involves a person walking towards the car, then disappearing from the camera view. *Rules* that specify spatial and temporal relationships can express this sequence of actions, and a detection system can evaluate these rules to detect such activities.

In this paper, we consider the next frontier in activity detection research, exploring the *near real-time* detection of *complex activities* potentially occurring across *multiple cameras*. A complex activity comprises two or more atomic actions, some of which may play out in one camera and some in another: *e.g.,* a person gets into a car in one camera, then gets out of the car in another camera and hands off a bag to a person.

We take a pragmatic, systems view of the problem, and ask: given a collection of (possibly wireless) surveillance cameras, what architecture and algorithms should an end-to-end system incorporate to provide *accurate* and *scalable* complex activity detection?

Future cameras are likely to be wireless and incorporate onboard GPUs (§2). However, activity detection using DNNs is too resource intensive for embedded GPUs on these cameras. Moreover, because complex activities may occur across multiple cameras, another device may need to aggregate detections at individual cameras. An *edge cluster* at a cable head end or a cellular base station is ideal for our setting: GPUs on this edge cluster can process videos from multiple cameras with low detection latency because the edge cluster is topologically close to the cameras (Figure 1).

Even with this architecture, complex activity detection poses several challenges: (a) How to specify complex activities occurring across multiple cameras? (b) How to partition the processing of the videos between compute resources available on the camera and the edge cluster? (c) How to reduce the wireless bandwidth requirement

between the camera and the edge cluster? (d) How to scale processing on the edge cluster in order to multiplex multiple cameras on a single cluster while still being able to process cameras in near real-time?

Contributions. In addressing these challenges, Caesar makes three important contributions.

First, it adopts a *hybrid* approach to complex activity detection where some parts of the complex activity use DNNs, while others are rule-based. This architectural choice is unavoidable: in the foreseeable future, purely DNN-based complex activity detection is unlikely, since training data for such complex activities is hard to come by. Moreover, a hybrid approach permits evolution of complex activity descriptions: as training data becomes available over time, it may be possible to train DNNs to detect more atomic actions.

Second, to support this evolution, Caesar defines a language to describe complex activities (§3). In this language, a complex activity consists of a sequence of *clauses* linked together by temporal relationships. A clause can either express a spatial relationship, or an atomic action. Caesar users can express multiple complex activities of interest, and Caesar can process camera feeds in near real-time to identify these complex activities.

Third, Caesar incorporates a *graph matching* algorithm that efficiently matches camera feeds to complex activity descriptions (§3). This algorithm leverages these descriptions to optimize wireless network bandwidth and edge cluster scaling. To optimize wireless network bandwidth, it performs object detection on the camera, then, at the edge cluster, lazily retrieves images associated with the detected objects only when needed (*e.g.,* to identify whether an object has appeared in another camera). To scale the edge cluster computation, it lazily invokes the action detection DNNs (the computational bottleneck) only when necessary.

Using a publicly available multi-camera data set, and an implementation of Caesar on an edge cluster, we show (§4) that, compared to a strawman approach which does not incorporate our optimizations, Caesar has 1-2 orders of magnitude lower detection latency and requires an order of magnitude less on-board camera memory (to support lazy retrieval of images). Caesar's graph matching algorithm works perfectly, and its accuracy is only limited by the DNNs we use for action detection and re-identification (determining whether two human images belong to the same person).

While prior work (§5) has explored the single-camera action detection [46, 54, 61], tracking of people across multiple overlapping cameras [33, 47, 59] and non-overlapping cameras [22, 42, 53], to our knowledge, no prior work has explored a near real-time hybrid system for multi-camera complex activity detection.

2 BACKGROUND AND MOTIVATION

Goal and requirements. Caesar detects *complex activities* across *multiple non-overlapping cameras*. It must support *accurate*, *efficient*, *near real-time* detection while permitting *hybrid* activity specifications. In this section, we discuss the goal and these requirements in greater detail.

Atomic and complex activities. An *atomic* activity is one that can be succinctly described by a single word label or short phrase, such as "walking", "talking", "using a phone". In this paper, we assume that atomic activities can be entirely captured on a single camera.

Figure 1: ***The high-level concept of a complex activity detection system: the user defines the rule then the system monitors incoming videos and outputs the matched frames.***

A *complex* activity (i) involves multiple atomic activities (ii) related in time (*e.g.,* one occurs before or after another), space (*e.g.,* two atomic activities occur near each other), or in the set of participants (*e.g.,* the same person takes part in two atomic activities), and (iii) can span multiple cameras whose views do not overlap. An example of a complex activity is: "A person walking while talking on the phone in one camera, and the same person talking to another person at a different camera a short while later". This statement expresses temporal relationships between activities occurring in two cameras ("a short while later") and spatial relationships between participants ("talking to another person").

Applications of complex activity detection. Increasingly, cities are installing surveillance cameras on light poles or mobile platforms like police cars and drones. However, manually monitoring all cameras is labor intensive given the large number of cameras [13], so today's surveillance systems can only deter crimes and enable forensic analysis. They cannot anticipate events as they unfold in near real time. A recent study [27] shows that such anticipation is possible: many crimes share common signatures such as "a group of people walking together late at night" or "a person getting out of a car and dropping something". Automated systems to identify these signatures will likely increase the effectiveness of surveillance systems.

The retail industry can also use complex activity detection. Today, shop owners install cameras to prevent theft and to track consumer behavior. A complex activity detection system can track customer purchases and browsing habits, providing valuable behavioral analytics to improve sales and design theft countermeasures.

Caesar architecture. Figure 1 depicts the high-level functional architecture of Caesar. Today, video processing and activity detection are well beyond the capabilities of mobile devices or embedded processors on cameras (§4). So Caesar will need to leverage *edge computing*, in which these devices offload video processing to a nearby server cluster. This cluster is a convenient rendezvous point for correlating data from non-overlapping cameras.

Caesar requirements. Caesar should process videos with *high throughput* and *low end-to-end* latency. Throughput, or the rate at which it can process frames, can impact Caesar's accuracy and can determine if it is able to keep up with the video source. Typical surveillance applications process 20 frames per second. The end-to-end latency, which is the time between when a complex activity occurs and when Caesar reports it, must be low to permit fast near real-time response to developing situations. In some settings, such as large outdoor events in locations with minimal infrastructure [6], video capture devices might be un-tethered so Caesar should *conserve wireless bandwidth* when possible. To do this, Caesar can

leverage significant on-board compute infrastructure: over the past year, companies have announced plans to develop surveillance cameras with onboard GPUs [2]. Since edge cluster usage is likely to incur cost (in the same way as cloud usage), Caesar should *scale* well: it should maximize the number of cameras that can be concurrently processed on a given set of resources. Finally, Caesar should have high precision and recall detecting complex activities.

The case for hybrid complex activity detection. Early work on activity detection used a *rule-based* approach [35, 51]. A rule codifies relationships between actors (people); rule specifications can use ontologies [51] or And-Or Graphs [35]. Activity detection algorithms match these rule specifications to actors and objects detected in a video.

More recent approaches are data-driven [46, 54, 61], and train deep neural nets (DNNs) to detect activities. These approaches extract *tubes* (sequences of bounding boxes) from video feeds; these tubes contain the actor performing an activity, as well as the surrounding context. They are then fed into a DNN trained on one or more action data sets (*e.g.,* AVA [1], UCF101 [11], and VIRAT [12]), which output the label associated with the activity. Other work [32] has used a slightly different approach. It learns rules as relationships between actors and objects from training data, then applies these rules to match objects and actors detected in a video feed.

While data-driven approaches are preferable over rule-based ones because they can generalize better, complex activity detection cannot use purely data-driven approaches. By definition, a complex activity comprises individual actions combined together. Because there can be combinatorially many complex activities from a given set of individual activities, and because data-driven approaches require large amounts of training data, it will likely *be infeasible to train neural networks for all possible complex activities of interest*.

Thus, in this paper, we explore a hybrid approach in which rules, based on an extensible vocabulary, describe complex activities. The vocabulary can include atomic actions: *e.g.,* "talking on a phone", or "walking a dog". Using this vocabulary, Caesar users can define a rule for "walking a dog while talking on the phone". Then, Caesar can detect a more complex activity over this new atomic action: "walking a dog while talking on the phone, then checking the postbox for mail before entering a doorway". (For brevity of description, a rule can, in turn, use other rules in its definition.)

Challenges. Caesar uses hybrid complex activity detection to process feeds in near real-time while satisfying the requirements described above. To do this, it must determine: (a) How to specify complex activities across multiple non-overlapping cameras? (b) How to optimize the use of edge compute resources to permit the system to scale to multiple cameras? (c) How to conserve wireless bandwidth by leveraging on-board GPUs near the camera?

3 CAESAR DESIGN

In Caesar, users first specify one or more rules that describe complex activities (Figure 5): this rule definition language includes elements such as objects, actors, and actions, as well as spatial and temporal relationships between them.

Cameras generate video feeds, and Caesar processes these using a three-stage pipeline (Figure 2). In the *object detection* stage, Caesar

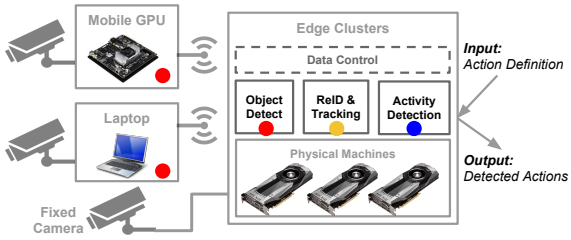


Figure 2: ***The high-level design of Caesar. Dots with different colors represent different DNN modules for specific tasks.***

	Input	Output
Object Detection	Image	Object Bounding Boxes
Track & ReID	Object Bounding Boxes Image	Object TrackID
Action Detection	Object Boxes & TrackID Image	Actions

Table 1: ***Input and output content of each module in Caesar.***

generates bounding boxes of actors and objects seen in each frame. For wireless cameras, Caesar can leverage on board mobile GPUs to run object detection on the device; subsequent stages must run on the edge cluster. The input to, and output of, object detection is the same regardless of whether it runs on the mobile device or the edge cluster. A *re-identification and tracking* module processes these bounding boxes. It (a) extracts *tubes* for actors and objects by tracking them across multiple frames and (b) determines whether actors in different cameras represent the same person. Finally, a *graph matching and lazy action detection* module determines: (a) whether the relationships between actor and object tubes match pre-defined rules for complex activities and (b) when and where to invoke DNNs to detect actions to complete rule matches. Table 1 shows the three modules' data format.

Figure 3 shows an example of Caesar's output for a single camera. It annotates the live camera feed with detected activities. In this snapshot, two activities are visible: one is a person who was using a phone in another camera, another is a person exiting a car. Our demonstration video[1] shows Caesar's outputs for multiple concurrent camera feeds.

Caesar meets the requirements and challenges described in §2 as follows: it processes streams continuously, so can detect events in near-real time; it incorporates robustness optimizations for tracking, re-identification, and graph matching to ensure accuracy; it scales by lazily detecting actions, thereby minimizing DNN invocation.

3.1 Rule Definition and Parsing

Caesar's first contribution is an extensible rule definition language. Based on the observation that complex activity definitions specify relationships in space and time between actors, objects, and/or atomic actions (§2, henceforth simply actions), the language incorporates three different *vocabularies* (Figure 4).

Vocabularies. An *element vocabulary* specifies the list of actors or objects (*e.g.,* "person", "bag", "bicycle") and actions (*e.g.,* "talking

[1] Caesar's demo video: https://vimeo.com/330176833

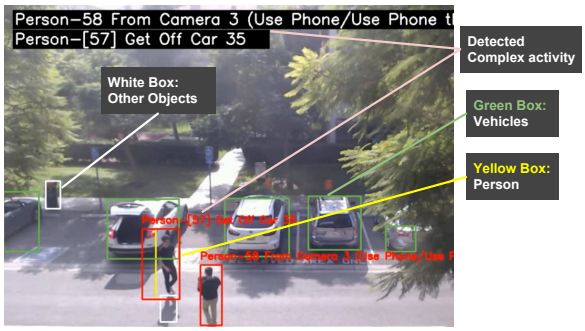


Figure 3: ***The output of Caesar with annotations.***

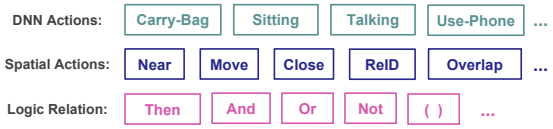


Figure 4: ***Examples of the vocabulary elements.***

on the phone"). As additional detectors for atomic actions become available (*e.g.,* "walking a dog") from new DNNs or new action definition rules, Caesar can incorporate corresponding vocabulary extensions for these.

A *spatial operator vocabulary* defines spatial relationships between actors, objects, and actions. Spatial relationships use binary operators such as "near" and "approach". For example, before a person $p1$ can talk to $p2$, $p1$ must "approach" $p2$ and then come "near" $p2$ (or vice versa). Unary operators such as "stop" or "disappear" specify the dispensation of participants or objects. For example, after approaching $p2$, $p1$ must "stop" before he or she can talk to $p2$. Another type of spatial operator is for describing which camera an actor appears in. The operator "re-identified" specifies an actor recognized in a new camera. The binary operator "same-camera" indicates that two actors are in the same camera.

Finally, a *temporal operator vocabulary* defines concurrent as well as sequential activities. The binary operator "then" specifies that one object or action is visible after another, "and" specifies that two objects or actions are concurrently visible, while "or" specifies that they may be concurrently visible. The unary operator "not" specifies the absence of a corresponding object or action.

A complex activity definition contains three components (Figure 5). The first is a unique name for the activity, and the second is a set of variable names representing actors or objects. For instance, $p1$ and $p2$ might represent two people, and c a car. The third is the definition of the complex activity in terms of these variables. A complex activity definition is a sequence of clauses, where each clause is either an action (*e.g.,* `p1 use-phone`), or a unary or binary spatial operator (*e.g.,* `(p1 close p2)`, or `(p1 move)`). Temporal operators link two clauses, so a complex activity definition is a sequence of clauses separated by temporal operators. Figure 5 shows examples of two complex actions, one to describe a person getting into a car, and another to describe a person who is seen, in two different cameras, talking on the phone while carrying a bag.

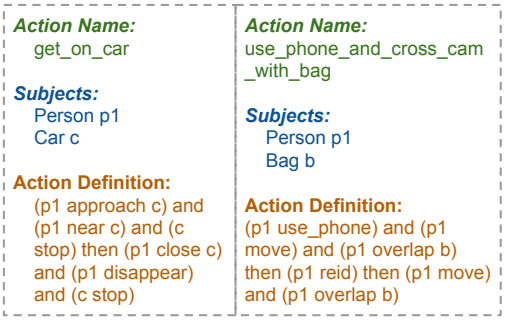


Figure 5: ***Two examples of action definition using Caesar syntax.***

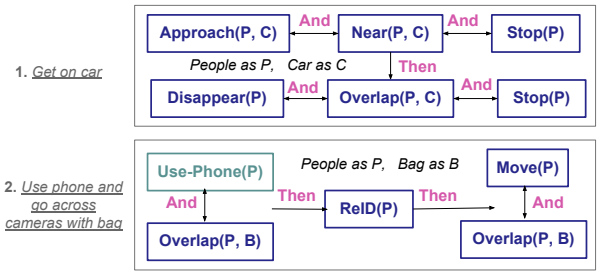


Figure 6: ***Two examples of parsed complex activity graphs.***

The rule parser. Caesar parses each rule to extract an intermediate representation suitable for matching. In Caesar, that representation is a directed acyclic graph (or DAG), in which nodes are clauses and edges represent temporal relationships. Figure 6 shows the parsed graphs of the definition rules. At runtime, Caesar's graph matching component attempts to match each complex activity DAG specification to the actors, objects, and actions detected in the video feeds of multiple cameras (§3.4).

3.2 Object Detection

On-camera object detection. The first step in detecting a complex activity is detecting objects in frames. This component processes each frame, extracts a bounding box for each distinct object within the frame, and emits the box coordinates, the cropped image within the bounding box, and the object label. Today, DNNs like YOLO [40] and SSD [30] can quickly and accurately detect objects. These detectors also have stripped-down versions that permit execution on a mobile device. Caesar allows a camera with on-board GPUs to run these object detectors locally. When this is not possible, Caesar schedules GPU execution on the edge cluster. (The next step in our pipeline involves re-identifying actors across multiple cameras, and cannot be easily executed on the mobile device).

Optimizing wireless bandwidth. When the mobile device runs the object detector, it may still be necessary to upload the cropped images for each of the bounding boxes (in addition to the bounding box coordinates and the labels). Surveillance cameras can see tens to hundreds of people or cars per frame, so uploading images can be bandwidth intensive. In Caesar, the mobile device maintains a cache

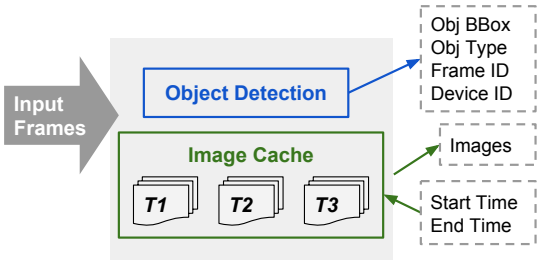


Figure 7: ***Workflow of object detection on mobile device.***

of recently seen images and the edge cluster *lazily retrieves* images from the mobile device to reduce this overhead.

Caesar is able to perform this optimization for two reasons. First, for tracking and re-identification (§3.3), not all images might be necessary; for instance, if a person appears in 20 or 30 successive frames, Caesar might need only the cropped image of the person from one of these frames for re-identification. Second, while all images might be necessary for action detection, Caesar minimizes invocation of the action detection module (§3.4), reducing the need for image transfer.

3.3 Tracking and Re-Identification

The tube abstraction. Caesar's expressivity in capturing complex activities comes from the *tube* abstraction. A tube is a sequence of bounding boxes over successive frames that represent the same object. As such, a tube has a distinct start and end time, and a label associated with the object. Caesar's tracker (Algorithm (1)) takes as input the sequence of bounding boxes from the object detector, and assigns, to each bounding box a globally unique *tube ID*. In its rule definitions (§3.1), Caesar detects spatial and temporal relationships between tubes. Tubes also permit low overhead re-identification, as we discuss below.

Algorithm 1 Cross-Camera Tracking and Re-Identification

```
1:  INPUT : list of bounding boxes
2:  for each person box B in bounding boxes do
3:    ID_B = update_tubes(existing_tubes, B)
4:    if ID_B not in existing_tubes then
5:      frame = get_frame_from_camera()
6:      F_B = get_DNN_feature(frame, B)
7:      for ID_local in local_tubes do
8:        if feature_dist(ID_local, ID_B) then
9:          ID_B = ID_local; Update(ID_local); return
10:       end if
11:     end for
12:     for ID_other in other_tubes do
13:       if feature_dist(ID_other, ID_B) then
14:         ID_B = ID_other; Update(ID_other); return
15:       end if
16:     end for
17:     Update(ID_B);
18:   end if
19: end for
```

Tracking. The job of the tracking sub-component is to extract tubes. This sub-component uses a state-of-the-art tracking algorithm called DeepSORT [57] that runs on the edge server side (line 6, Algorithm (1)). DeepSORT takes as input bounding box positions and extracts features of the image within the bounding box. It then tracks the bounding boxes using Kalman filtering, as well as judging the image feature and the intersection-over-union between successive bounding boxes.

Caesar receives bounding boxes from the object detector and passes it to DeepSORT, which either associates the bounding box with an existing tube or fails to do so. In the latter case, Caesar starts a new tube with this bounding box. As it runs, Caesar's tracking component continuously saves bounding boxes and their tube ID associations to a distributed key-value store within the edge cluster, described below, that enables fast tube matching in subsequent steps.

Caesar makes one important performance optimization. Normally, DeepSORT needs to run a DNN for person re-identification features. This is feasible when object detection runs on the edge cluster (§3.2). However, when object detection runs on the mobile device, feature extraction can require additional compute and network resources, so Caesar relies entirely on DeepSORT's ability to track using bounding box positions alone. As we show in §4, this design choice permits Caesar to conserve wireless network bandwidth by transmitting only bounding box positions instead of uploading the whole frame.

Robust tracking. When in the view of the camera, an object or actor might be partially obscured. If this happens, the tracking algorithm detects two distinct tubes. To be robust to partial occlusions, Caesar retrieves the cropped image corresponding to the first bounding box in the tube (line 5, Algorithm (1)). Then, it applies a re-identification DNN (described below) to match this tube with existing tubes detected in the local camera (lines 7-10, Algorithm (1)). If it finds a match, Caesar uses simple geometric checks (*e.g.,* bounding box continuity) before assigning the same identifier to both tubes.

Cross-camera re-identification. Cross camera *re-identification* is the ability to re-identify a person or object between two cameras. Caesar uses an off-the-shelf DNN [56] which, trained on a corpus of images, outputs a feature vector that uniquely identifies the input image. Two images belong to the same person if the distance between the feature vectors is within a predefined threshold.

To perform re-identification, Caesar uses the image retrieved for robust tracking, and searches a *distributed key-value store* for a matching tube from another camera (line 12-15, §1). Because the edge cluster can have multiple servers, and different servers can process feeds from different cameras, Caesar uses a fast in-memory distributed key-value store [9] to save tubes.

Re-identification can incur a high false positive rate. To make it more robust, we encode the camera topology [38] in the re-identification sub-component. In this topology, nodes are cameras, and an edge exists between two cameras only if a person or a car can go from one camera to another without entering the field of view of any other *non-overlapping* camera. Given this, when Caesar tries to find a match for a tube seen at camera **A**, it applies the re-identification DNN only to tubes at neighbors of **A** in the topology. To scope this search, Caesar uses travel times between cameras [38].

3.4 Action Detection and Graph Matching

In a given set of cameras, users may want to detect multiple complex activities, and multiple instances of each activity can occur. Caesar's rule parser (§3.1) generates an intermediate graph representation for each complex activity, and the *graph matching* component matches tubes to the graph in order to detect when a complex activity has occurred. For reasons discussed below, graph matching dynamically invokes atomic action detection, so we describe these two components together in this section.

Node matching. The first step in graph matching is to match tubes to nodes in one or more graphs. Recall that a node in a graph represents a clause that describes spatial actions or spatial relationships. Nodes consist of a unary or binary operator, together with the corresponding operands. For each operator, Caesar defines algorithm to evaluate the operator.

Matching unary operators. For example, consider the clause `stop c`, which determines whether the car `c` is stationary. This is evaluated to true if the bounding box for `c` has the same position in successive frames. Thus, a tube belonging to a stationary car matches the node `stop c` in each graph, and Caesar binds `c` to its tube ID.

Similarly, the unary operator `disappear` determines if its operand is no longer visible in the camera. The algorithm to evaluate this operator considers two scenarios: an object or person disappearing (visible in one frame and not visible in the next) by (a) entering the vehicle or building, or (b) leaving the camera's field of view. When either of these happen, the object's tube matches the corresponding node in a graph.

Matching binary operators. For binary operators, node matching is a little more involved, and we explain this using an example. Consider the clause `p1 near p2`, which asks: is there a person near another person? To evaluate this, Caesar checks each pair of person tubes to see if there was any instant at which the corresponding persons were close to each other. For this, it divides up each tube into small chunks of duration t (1 second in our implementation), and checks for the *proximity* of all bounding boxes pairwise in each pair of chunks.

To determine proximity, Caesar uses the following metric. Consider two bounding boxes x and y. Let $d(x, y)$ be the smallest pixel distance between the outer edges of the bounding box. Let $b(x)$ (respectively $b(y)$) be the largest dimension of bounding box x (respectively, y). Then, if either $\frac{d(x,y)}{b(x)}$ or $\frac{d(x,y)}{b(y)}$ is less than a fixed threshold δ we say that the two bounding boxes are proximate to each other. Intuitively, the measure defines proximity with respect to object dimensions: two large objects can have a larger pixel distance between them than two small objects, yet Caesar may declare the larger objects close to each other, but not the smaller ones.

Finally, `p1 near p2` is true for two people tubes if there is a chunk within those tubes in which a majority of bounding boxes are proximate to each other. We use the majority test to be robust in bounding box determinations in the underlying object detector.

Caesar includes similar algorithms for other binary spatial operators. For example, the matching algorithm for `p1 approaches p2` is a slight variant of `p1 near p2`: in addition to the proximity check, Caesar also detects whether bounding boxes in successive frames decrease in distance just before they come near each other.

Time of match. In all of these examples, a match occurs within a specific time interval (t_1, t_2). This time interval is crucial for edge matching, as we discuss below.

Edge matching. In Caesar's intermediate graph representation, an edge represents a temporal constraint. We permit two types of temporal relationships: concurrent (represented by `and` which requires that two nodes must be concurrent, and `or` which specifies that two nodes may be concurrent), and sequential (one node occurs strictly after another).

To illustrate how edge matching works, consider the following example. Suppose there are two matched nodes `a` and `b`. Each node has a time interval associated with the match. Then `a and b` are concurrent if their time intervals overlap. Otherwise, `a then b` is true if `b`'s time interval is strictly after `a`'s.

Detecting atomic actions. Rule-based activity detection has its limits. Consider the atomic action "talking on the phone". One could specify this action using the rule `p1 near m`, where `p1` represents a person and `m` represents a mobile phone. Unfortunately, phones are often too small in surveillance videos to be captured by object detectors. DNNs, when trained on a large number of samples of people using phones, can more effectively detect this atomic action.

Action matching. For this reason, Caesar rules can include clauses matched by a DNN. For example, the clause `talking_phone(p1)` tries to find a person tube by applying each tube to a DNN. For this, Caesar uses the DNN described in [54]. We have trained this on Google's AVA [1] dataset which includes 1-second video segments from movies and action annotations. The training process ensures that the resulting model can detect atomic actions in surveillance videos without additional fine-tuning; see [54] for additional details. The model can recognize a handful of actions associated with person tubes, such as: "talking on the phone", "sitting", and "opening a door". For each person tube, it uses the Inflated 3D features [20] to extract features which represent the temporal dynamics of actions, and returns a list of possible action labels and associated confidence levels. Given this, we say that a person tube matches `talking_phone(p1)` if there is a video chunk in which "talking on the phone" has a higher confidence value than a fixed threshold τ.

Efficiency considerations. In Caesar's rule definition language, an action clause is a node. Matching that node requires running the DNN on every chunk of every tube. This is inefficient for two reasons. The first is *GPU inefficiency*: the DNN takes about 40 ms for each chunk, so a person who appears in the video for 10 s would require 0.4 s to process (each chunk is 1 s) unless Caesar provisions multiple GPUs to evaluate chunks in parallel. The second is *network inefficiency*. To feed a person tube to the DNN, Caesar would need to retrieve *all* images for that person tube from the mobile device.

Lazy action matching. To address these inefficiencies, Caesar matches actions lazily: it first tries to match all non-action clauses in the graph, and only then tries to match actions. To understand how this works, consider the rule definition `a then b then c`, where `a` and `c` are spatial clauses, and `b` is a DNN-based clause. Now, suppose `a` occurs at time t_1 and `c` at t_2, Caesar executes the

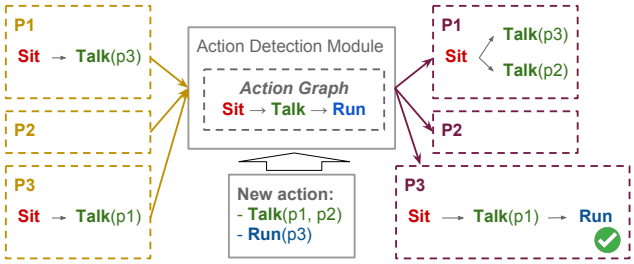


Figure 8: ***An example of graph matching logic: the left three graphs are unfinished graphs of each tube, and the right three graphs are their updated graphs.***

DNN	*Speed (FPS)*
Object Detection [30, 40]	40~60
Tracking & ReID [57]	30~40
Action Detection [54]	50~60 (per tube)

Table 2: ***The runtime frame rate of each DNN model used by Caesar (evaluated on a single desktop GPU [7]).***

DNN on all tubes that start after t_1 and end before t_2 in order to determine if there is a match. This addresses both the GPU and network inefficiency discussed above, since the DNN executes fewer tubes and Caesar retrieves fewer images.

Algorithm (2) shows the complete graph matching algorithm. The input contains the current frame number as timestamp and a list of positions of active tubes with their tube IDs and locations. If Caesar has not completed assembling a tube (*e.g.,* the tube's length is less than 1 second), it appends the tube images to the temporary tube videos and records the current bounding box locations (lines 2-3). When a tube is available, Caesar attempts to match the spatial clauses in one or more complex activity definition graphs (line 5). Once it determines a match, Caesar marks the vertex in that graph as done, and checks the all its neighbor nodes in the graph. If the neighbor is a DNN node, it adds a new entry to the DNN checklist of the graph, and moves on to its neighbors. The entry contains the tube ID, DNN action label, starting time, and the ending time. The starting time is the time when the node is first visited. The end time is the time when Caesar matches its next node. In our example above, when `a` is matched at timestamp $T1$, Caesar creates an entry for `b` in this graph, with the starting time as $T1$. When `c` is matched at $T2$, the algorithm adds $T2$ to `b`'s entry as the ending timestamp.

4 EVALUATION

In this section, we evaluate Caesar's accuracy and scalability on a publicly available multi-camera data set.

4.1 Methodology

Implementation and experiment setup. Our experiments use an implementation of Caesar which contains (a) object detection and image caching on the camera, (b) tracking, re-identification, action detection, and graph matching on the edge cluster. Caesar's demo code is available at https://github.com/USC-NSL/Caesar.

Algorithm 2 Activity Detection with Selective DNN Activation

```
1:  INPUT : incoming tube
2:  if tube_cache not full then
3:      tube_cache.add(tube); return
4:  end if
5:  spatial_acts = get_spatial_actions (tube_cache)
6:  for sa in spatial_acts do
7:      for g in tube_graph_mapping [sa.tube_id] do
8:          if sa not in g.next_acts then
9:              continue
10:         end if
11:         if g.has_pending_nn_act then
12:             nn_acts = get_nn_actions (g.nn_start, cur_time())
13:             if g.pending_nn_act not in nn_acts then
14:                 continue
15:             end if
16:         end if
17:         g.next_acts = sa.neighbors()
18:         tube_graph_mapping.update()
19:         if g.last_node_matched then
20:             add g to output activities
21:         end if
22:     end for
23: end for
```

In our experiments, we use multiple cameras equipped with Nvidia's TX2 GPU boards [8]. Each platform contains a CPU, a GPU, and 4 GB memory shared between the CPU and the GPU. Caesar runs a DNN-based object detector, SSD-MobilenetV2 [43], on the camera. As described earlier, Caesar caches the frames on the camera, as well as cropped images of the detected bounding boxes. It sends box coordinates to the edge cluster using RPC [5]. The server can subsequently request a camera for additional frames, permitting lazy retrieval.

A desktop with three Nvidia RTX 2080 GPUs [7] runs Caesar on the server side. One of the GPUs runs the state-of-the-art online re-identification DNN [56] and the other two execute the action detection DNNs [54]. Each action DNN instance has its own input data queue so Caesar can load-balance action detection invocations across these two for efficiency. We use Redis [9] as the in-memory key-value storage. Our implementation also includes Flask [4] web server that allows users to input complex activity definitions, and visualize the results.

DNN model selection. The action detector and the ReID DNN require 7.5 GB of memory, far more than the 4 GB available on the camera. This is why, as described earlier, our mobile device can only run DNN-based object detection. Among the available models for object detection, we have evaluated four that fit within the camera's GPU memory. Table 3 shows the accuracy and speed (in frames per second) of these models on our evaluation dataset. Our experiments use SSD-Mobilenet2 because it has good accuracy with high frame rate, which is crucial for Caesar because a higher frame rate can lead to higher accuracy in detecting complex activities.

We use [56] for re-identification because it is lightweight enough to not be the bottleneck. Other ReID models [50, 62] are more accurate than [56] on our dataset (tested offline), but are too slow (<

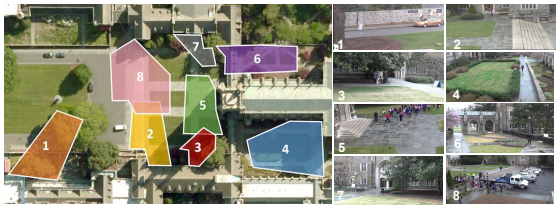


Figure 9: ***Camera placement and the content of each camera.***

10 fps) to use in Caesar. For atomic actions, other options [25, 49] have slight higher accuracy than [54] on the AVA dataset, but are not publicly available yet and their performance is not reported.

Dataset. We use DukeMTMC [3] for our evaluations. It has videos recorded by eight non-overlapping surveillance cameras on campus. The dataset also contains annotations for each person, which gives us the ground truth of each person tube's position at any time. We selected 30 minutes of those cameras' synchronized videos for testing Caesar. There are 624 unique person IDs in that period, and each person shows up in the view of 1.7 cameras on average. The average number of people showing up in all cameras is 11.4, and the maximum is 69.

Atomic action ground truth. DukeMTMC was originally designed for testing cross-camera person tracking and ReID, so it does not have any action-related ground truth. Therefore, we labeled the atomic actions in each frame for each person, using our current action vocabulary. Our action ground truth contains the action label, timestamp, and the actor's person ID. We labeled a total of 1,289 actions in all eight cameras. Besides the atomic actions, we also labeled the ground truth traces of cars and bikes in the videos. Figure 9 shows the placement of these cameras and a sample view from each camera.

Complex activity ground truth. We manually identified 149 complex activities. There are seven different categories of these complex activities as shown in Table 4. This table also lists two other attributes of the complex activity type and the dataset. The third column of the table shows the number of instances in the data set of the corresponding complex activity, broken down by how many of them are seen on a single camera vs. multiple cameras. Thus, for the first complex activity, the entry *12/1* means that our ground-truth contains 12 instances that are visible only on a single camera, and one that is visible across multiple cameras.

These complex activities are of three kinds. #1's clauses, labeled "NN-only", are all atomic actions detected using a NN. #2 through #5, labeled "Mixed", have clauses that are either spatial or are atomic actions. The last two, #6 and #7, labeled "Spatial-only", have only spatial clauses.

Metrics. We replay the eight videos at 20 fps to simulate the real-time camera input on cameras. Caesar's server takes the input from all mobile nodes, runs the action detection algorithm for a graph, and outputs the result into logs. The results contain the activity's name, timestamp, and the actor or object's bounding box location when the whole action finishes. Then we compare the log with the annotated ground truth. A *true positive* is when the detected activity matches the ground truth's complex activity label, overlaps with the ground truth's tubes for the complex activity, and has timestamp difference within a 2-second threshold. We report: *recall*, which is the fraction of the ground truth classified as true positives, and *precision*, which is the fraction of true positives among all Caesar-detected complex activities. We also evaluate the Caesar's scalability, as well as the impact of its performance optimizations; we describe the corresponding metrics for these later.

DNN	*Speed (FPS)*	*Accuracy (mAP)*
SSD [30]	3.7	91
YOLOv3 [40]	4.1	88
TinyYOLO [39]	8.5	84
SSD-MobileNetv2 [43]	11.2	83

Table 3: ***Speed and accuracy of different DNNs on mobile GPU.***

ID	*Complex Activity*	*# of Samples (Single/Multi)*	*Type*
1	Use phone then talk	12 / 1	NN-only
2	Stand, use phone then open door	9 / 2	Mixed
3	Approach and give stuff	10 / 0	Mixed
4	Walk together then stop and talk	6 / 2	Mixed
5	Load stuff and get on car	2 / 0	Mixed
6	Ride with bag in two cams	0 / 8	Spatial-only
7	Walk together in two cams	0 / 97	Spatial-only

Table 4: ***Summary of labeled complex activities***

4.2 Accuracy

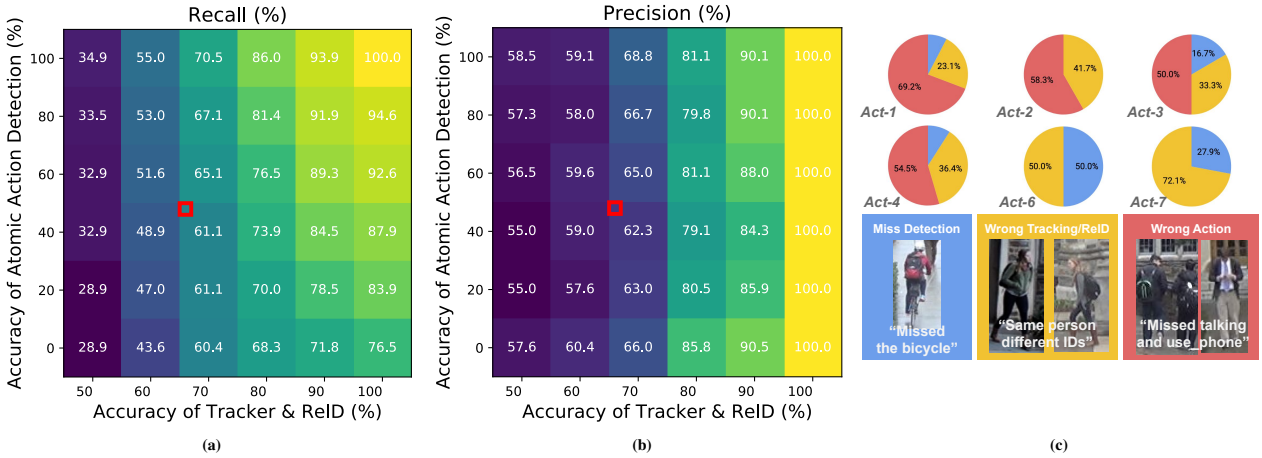
Overall. Table 5 shows the recall and precision of all complex activities. #1 (using the phone and then talking to a person) and #4 (walking together then stopping to talk) have the lowest recall at 46.2% and the lowest precision at 36.4%. At the other end, #5's two instances achieve 100% recall and precision. Across all complex activities, Caesar has a recall of %61.0 and a precision of %59.5 precision.

Understanding the accuracy results. Our results show that most NN-only and Mixed activities have lower position and recall than those in the Spatial-only category. Recall that Caesar uses off-the-shelf neural networks for action detection and re-identification. This suggests that the action detection DNN, used in the first two categories but not in the third, is the larger source of detection failures than the re-identification DNN. Indeed, the reported mean average precision for these two models are respectively 45% and 65% in our dataset.

We expect the overall accuracy of complex activity detection to increase in the future for two reasons. We use off-the-shelf networks for these activities that are not customized for this camera. There is significant evidence that customization can improve the accuracy of neural networks [37] especially for surveillance cameras since their viewing angles are often different from the images used for training these networks. Furthermore, these are highly active research areas, so with time we can expect improvements in accuracy.

Two natural questions arise: (a) as these neural networks improve, how will the overall accuracy increase? and (b) to what extent does Caesar's graph matching algorithm contribute to detection error? We address both of these questions in the following analysis.

Projected accuracy improvements. We evaluate Caesar with the tracker and the action detector at different accuracy levels. To do this,



(a) (b) (c)

Figure 10: ***Caesar's (a) recall rate and (b) precision rate with different action detection and tracker accuracy. (c) The statistics and sample images of failures in all complex activities.***

for each of these components, we vary the accuracy p (expressed as a percentage) as follows. We re-run our experiment, but instead of running the DNNs when Caesar invokes the re-identification and action detection components, we return the ground truth *for that DNN* (which we also have) $p\%$ of the time, else return an incorrect answer. By varying p, we can effectively simulate the behavior of the system as DNN accuracy improves. When p is 100%, the re-identification DNN works perfectly and Caesar always gets the correct person ID in the same camera and across cameras, so tracking is 100% accurate. Similarly, when the action DNN has 100% accuracy, it always captures every atomic action correctly for each person. We then compare the end-to-end result with the complex activity ground-truth to explore precision and accuracy.

Figure 10a and Figure 10b visualize the recall and precision of Caesar with different accuracy in the tracker and the atomic action detector. In both of these figures, the red box represents Caesar's current performance (displayed for context).

The first important observation from these graphs is that, when the action detection and re-identification DNNs are perfect, Caesar achieves 100% precision and recall. This means that the rest of the system, which is a key contribution of the paper, works perfectly; this includes matching the spatial clauses, and the temporal relationships, while lazily retrieving frames and bounding box contents from the camera, and using shared key-value store to help perform matching across multiple cameras.

The second observation from this graph is that the re-identification DNN's accuracy affects overall performance more than that of the action detector. Recall that the re-identification DNN tracks people both within the same camera and across cameras. There are two reasons for why it affects performance more than the action detector. The first is the dependency between re-identification, action detection, and graph matching. If the re-identification is incorrect, then regardless of whether action detection is correct or not, matching will fail since those actions do not belong to the same tube. Thus, when re-identification accuracy increases, correct action detection will boost overall accuracy. This is also why, in Figure 10b, the overall precision is perfect even when the action detector is less than perfect. Second, 70% (105) samples of complex activities in the current dataset are Spatial-only, which relies more heavily on re-identification, making the effectiveness of that DNN more prominent.

Action ID	1	2	3	4	5	6	7
Recall (%)	46.2	54.5	60	50	100	50	65.3
Precision (%)	43.8	41.7	42.8	36.4	100	100	63

Table 5: ***Caesar's recall and precision on the seven complex activities shown in Table 4.***

From those two figures, for a complex activity detector with >90% in recall and precision, the object detector/tracker must have >90% accuracy and the action detector should have >80% accuracy. We observe that object detectors (which have been the topic of active research longer than activity detection and re-identification) have started to approach 90% accuracy in recent years.

Finally, as an aside, note that the precision projections are non-monotonic (Figure 10b): for a given accuracy of the Re-ID, precision is non-monotonic with respect to accuracy of atomic action detection. This results from the dependence observed earlier: if Re-ID is wrong, then even if action detection is accurate, Caesar will miss the complex activity.

Failure analysis. To get more insight into these macro-level observations, we examined all the failure cases, including false positives and false negatives; Figure 10c shows the error breakdown for each activity (#5 is not listed because it does not have an error case).

The errors fall into three categories. First, the object detection DNN is not perfect and can miss the boxes of some actors or objects such as bags and bicycles, affecting tube construction and subsequent re-identification. This performance is worse in videos with rapid changes in the ratio of object size to image scale, large within-class variations of natural object classes, and background clutter [52]. Figure 10c shows the object detector missing a frontal view of a bicycle.

Complex Activity	# of Samples (Single/Multi)	Recall(%) /Precision(%)
Eat then drink	5/0	80/80
Shake hands then sit	6/0	83.3/100
Use laptop then take photo	2/2	75/100
Carry bag and sit then read	2/4	66.7/80
Use laptop, read then drink	0/4	75/100
Read, walk then take photo	0/4	75/100
Carry bag, sit, then eat, then drink, then read, then take photo	0/4	50/100

Table 6: ***Activities in a three-camera dataset, and Caesar's accuracy.***

Re-ID either within a single camera, or across multiple cameras, is error-prone. Within a camera, a person may be temporarily occluded. Tracking occluded people is difficult especially in a crowded scene. Existing tracking algorithms and Re-ID DNNs have not completely solved the challenge, and result in many *ID-switches* within a single camera. Similarly, Re-ID can fail across cameras even with our robustness enhancements which encode camera topology. This occurs because of different lighting conditions, changes in pose between different cameras, different percentage of occlusions, and different sets of detectable discriminative features [18, 48]. An incorrect re-identification results in Caesar missing all subsequent atomic actions performed by a person.

Action detection is the third major cause of detection failures. Blurry frames, occlusions, and an insufficient number of frames in a tube can all cause high error rates for the action DNN, resulting from incorrect labeling [34]. As described earlier, errors in other modules can propagate to action detection: object detection failures can result in shorter tubes for action detection; the same is true of re-identification failures within a single camera.

Object detection failure is the least important factor, although it affects #6 because it requires detecting a bicycle. For the graphs that require DNN-generated atomic actions, the action detection error is more influential than tracking. In the Spatial-Only cases, the tracking issue is the major cause of errors.

Caesar performance on more complex activities. Since the MTMC dataset has few complex activities, we recorded another dataset to test Caesar on a variety of cross-camera complex activities Table 6. We placed three non-overlapping cameras in a plaza on our campus. Volunteers acted out activities not included in Table 4, such as "shake hands", "eat", and "take a photo"; these invoke our action DNN. We observe (Table 6) that Caesar can capture these complex activities with precision >80% (100% for 5 of the 7 activities) and recall ≥75% for 5 of the activities. All failures are caused by incorrect re-ID due to varying lighting conditions.

4.3 Scalability

We measure the scalability of Caesar by the number of concurrent videos it can support with fixed server-side hardware, assuming cameras have on-board GPUs for running action detection.

Strawman approach for comparison. Caesar's lazy action detection is an important scaling optimization. To demonstrate its effectiveness, we evaluate a strawman solution which disables lazy action detection. The strawman retrieves images from the camera for every bounding box, and runs action detection on every tube.

Recall that lazy invocation of action detection does not help for NN-only complex activities. Lazy invocation first matches spatial clauses in the graph, then selectively invokes action detection. But, NN-only activities do not contain spatial clauses, so Caesar invokes action detection on all tubes, just like the strawman. However, for Mixed or Spatial-only complex activities lazy invocation conserves the use of GPU resources.

To highlight these differences, we perform this experiment by grouping the complex activities into these three groups: Strawman (which is the same as NN-only for the purposes of this experiment), Mixed, and Spatial-Only. Thus, for example, in the Mixed group experiment, Caesar attempts to detect all Mixed complex activities.

Latency. Figure 11(a) shows Caesar's *detection latency* is a function of the number of cameras for each of these three alternatives. The detection latency is the time between when a complex activity completes in a camera to when Caesar detects it.

The results demonstrate the impact of the scaling optimization in Caesar. As the number of cameras increases, detection latency can increase dramatically for the strawman, going up to nearly 1000 seconds with eight cameras. For Mixed complex activities, the latency is an order of magnitude less at about 100 seconds; this illustrates the importance of our optimization, without which Caesar's performance would be similar to the Strawman, an order of magnitude worse. For Spatial-only activities that do not involve action detection, the detection latency is a few seconds; Caesar scales extremely well for these types of complex activities.

Recall that in these experiments, we fix the number of GPU resources. In practice, in an edge cluster, there is likely to be some elasticity in the number of GPUs, so Caesar can target a certain detection latency by dynamically scaling the number of GPUs assigned for action detection. We have left this to future work.

Finally, we note that up to 2 cameras, all three approaches perform the same; in this case, the bottleneck is the object detector on the camera with a frame rate of 20 fps.

Frame rate. Figure 11(b) shows the average frame rate at which Caesar can process these different workloads, as a function of the number of cameras. This figure explains why Strawman's latency is high: its frame rate drops precipitously down to just two frames per second with eight concurrent cameras. Caesar scales much more gracefully for other workloads: for both Mixed and Spatial-only workloads, it is able to maintain over 15 frames per second even up to eight cameras.

These results highlight the importance of hybrid complex activity descriptions. Detecting complex activities using neural networks can be resource-intensive, so Caesar's ability to describe actions using spatial and temporal relationships while using DNNs sparingly is the key to enabling scalable complex activity detection system.

Cache size. Caesar maintains a cache of image contents at the camera. The longer the detection latency, the larger the cache size. Thus, another way to examine Caesar's scalability is to understand the cache size required for different workloads with different number of concurrent cameras. The camera's cache size limit is 4 GB.

Figure 11(c) plots the largest cache size observed during an experiment for each workload, as a function of the number of cameras.

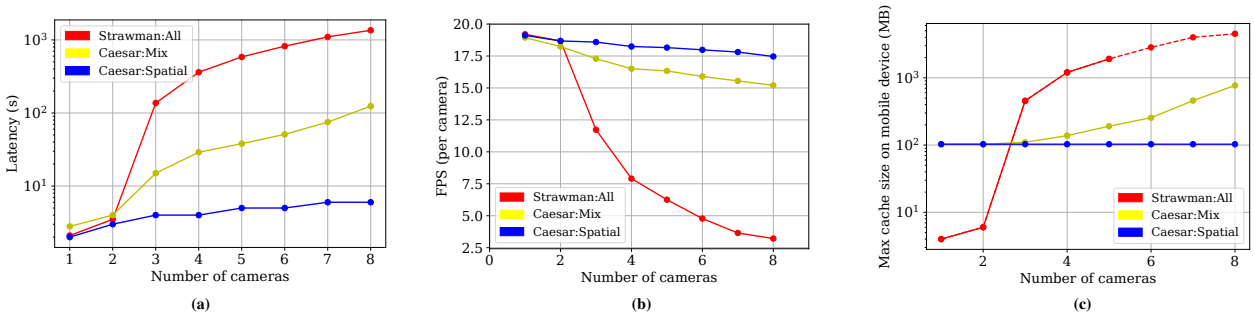


Figure 11: ***(a) Latency of Caesar and the strawman solution with different number of inputs. (b) Throughput of Caesar and the strawman solution with different number of Inputs. (c) Maximum cache size needed for Caesar and the strawman solution to reach the best accuracy, with different number of Inputs.***

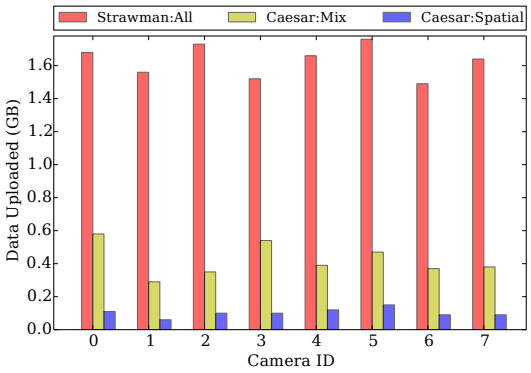


Figure 12: ***The total amount of data to be uploaded from each camera, with different uploading schemes.***

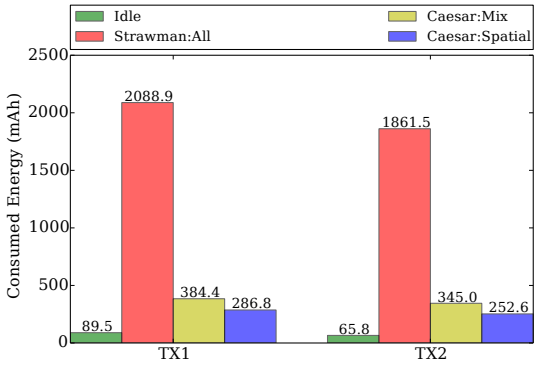


Figure 13: ***The average energy consumption of cameras in Caesar, with different uploading scheme and action queries.***

Especially for the Strawman, this cache size exceeded the 4 GB limit on the camera, so we re-did these experiments on a desktop with more memory. The dotted-line segment of the Strawman curve denotes these desktop experiments. When Caesar exceeds memory on the device, frames can be saved on persistent storage on the mobile device, or be transmitted to the server for processing, at the expense of latency and bandwidth.

Strawman has an order of magnitude higher cache size requirements precisely because its latency is an order of magnitude higher than the other schemes; Caesar needs to maintain images in the cache until detection completes. In contrast, Caesar requires a modest and fixed 100 MB cache for Spatial-only workloads on the camera: this supports lazy retrieval of frames or images for re-identification. The cache size for Mixed workloads increases in proportion to the increasing detection latency for these workloads.

4.4 Data Transfer

Caesar's lazy retrieval of images conserves wireless bandwidth, and to quantify its benefits, we measure the total amount of data uploaded from each camera (the edge cluster sends small control messages to request image uploads; we ignore these). In Figure 12, the strawman solution simply uploads the whole 30-min video with metadata (more than 1.5 GB for each camera). Mixed transfers >3× fewer data, and Spatial-only is an order of magnitude more efficient than Strawman. Caesar's data transfer overhead can be further optimized by transferring image deltas, leveraging the redundancy in successive frames or images within successive bounding boxes; we have left this to future work.

4.5 Energy Consumption

Even for wireless surveillance cameras with power sources, it may be important to quantify the energy required to run these workloads on the camera. The TX2 GPUs' onboard power supply chipset provides instantaneous power consumption readings at 20 Hz. We plot, in Figure 13, the total energy required for each workload by integrating the power consumption readings across the duration of the experiment. For context, we also plot the idle energy consumed by the device when run for 30 mins.

Strawman consumes 1800 mAh for processing our dataset, comparable to the battery capacity of modern smart phones. For Mixed and Spatial workloads, energy consumption is, respectively, 6× to 10× lower, for two reasons: (a) these workloads upload fewer images, reducing the energy consumed by the wireless network interface; (b)

Strawman needs to keep the device on for longer to serve retrieval requests because its detection latency is high.

5 RELATED WORK

Action detection pipelines. Existing pipelines detect the person's location (bounding box) in the video, extract representative features for the person's tube, and output the probability of each action label. Recent pipelines [46, 54, 61] leverage DNNs to extract features from person tubes and predict actions. Other work [26, 31] estimates human behavior by detecting the head and hand positions and analyzing their relative movement. Yet others analyze the moving trajectories of objects near a person to predict the interaction between the person and the object [15, 32]. By doing this, the action detector can describe more complex actions. The above approaches achieve high accuracy only with sufficient training samples, which limits their applications for more complex activities that involve multiple subjects and long duration.

Rather than analyzing a single actor's frames, other complementary approaches [14, 17] present their solutions to detect group behavior such as "walk in group", "stand in queue", and "talk together". The approach is to build a monolithic model that takes input both the behavior feature of each actor and the spatial-temporal relation (e.g. distance change), and outputs the action label. The model could be an recurrent neural network ([17]) or a handcrafted linear-programming model ([14]). However, both models require training videos to work properly because the models need training, rendering these approaches unsuitable for Caesar.

Zero-shot action detection is closely related to Caesar, and targets near real-time detection even when there are very few samples for training. Some approaches [36, 58] train a DNN-based feature extractor with videos and labels. The feature extractor can generate similar outputs for the actions that share similar attributes. For example, "surfing" and "swimming" have more in common than "surfing" and "basketball". When an unknown action tube arrives, these approaches cluster it with existing labels, and evaluate its similarity with the few positive samples. Another approach [24] further decomposes an action query sentence into meaningful keywords which have corresponding features clusters, and waits for those clusters to be matched together at runtime. However, these zero-shot detection approaches suffer from limited vocabulary and low accuracy (<40%).

Cross-camera tracking and re-identification. Several approaches [33, 47, 59] track multiple people across cameras but require overlapped cameras, which may be too restrictive for most surveillance systems. In non-overlapping scenarios, other approaches [22, 42, 53] leverage the visual similarity between people's traces in different cameras to match them. They also run a bipartite matching algorithm globally to minimize the ID assignment error. However, these approaches can only work offline for best performance and are unsuitable for Caesar's realtime needs.

Scaling DNN pipelines. More and more applications rely on a chain of DNNs running on edge clusters. This raises challenges for scaling well with fixed number of computation resources. Recent work [60] addresses the problem by tuning different performance settings (frame rate and resolution) for task queries to maximize the server utilization while keeping the quality of service. Downgrading frame rates and DNNs is not a good choice for Caesar because both options will adversely impact accuracy. [28] proposes a scheduler on top of TensorFlow Serving [10] to improve the GPU utilization with different DNNs on it. Caesar could leverage such a model serving system, but is complementary to it. Recent approaches [23, 29] cache the intermediate results to save GPU cycles. Caesar goes one step further with lazily activating the action DNN. [23] also batches the input for higher per-image processing speed on GPU, which Caesar also adopts to perform object detection on the mobile device.

Wireless camera networks. Wang *et al.* [55] discuss an edge-computing based approach in which a group of camera-equipped drones efficiently livestream a sporting event. However, they focus on controlling drone movements and efficiently transmitting the video frame over a shared wireless medium in order to maintain good quality live video streaming with low end to end latency. Other work [16] presents a new FPGA architecture and a communication protocol for that architecture to efficiently transmit images in a wireless camera network. San Miguel *et al.* [44] present a vision of a smart multi-camera network and the required optimization and properties, but discuss no specific detection techniques. A related work [41] proposes a method for re-configuring the camera network over time based on the description of the camera nodes, specifications of the area of interest and monitoring activities, and a description of the analysis tasks. Finally, MeerKats [19] uses different image acquisition policies with resource management and adaptive communication strategies. No other prior work has focused on cross-camera complex activity detection, as Caesar has.

Query optimization. Database query optimization [21, 45] focuses on deriving an optimal sequence of database operations to execute a query. Caesar's lazy evaluation of DNNs is similar in spirit, but is determined when it matches spatial operators.

6 CONCLUSION

This paper presents Caesar, a *hybrid* multi-camera complex activity detection system that combines traditional rule based activity detection with DNN-based activity detection. Caesar supports an extensible vocabulary of actions and spatio-temporal relationships and users can specify complex activities using this vocabulary. To satisfy the network bandwidth and low latency requirements for near real-time activity detection with a set of non-overlapping (possibly wireless) cameras, Caesar partitions activity detection between a camera and a nearby edge cluster that lazily retrieves images and lazily invokes DNNs. Through extensive evaluations on a public multi-camera dataset, we show that Caesar can have high precision and recall rate with accurate DNN models, while keeping the bandwidth and GPU usage an orders of magnitude lower that a strawman solution that does not incorporate its performance optimizations. Caesar also reduces the energy consumption on the mobile nodes by 7X. Future work with Caesar includes evaluations with a larger dataset that has more cameras and a larger set of complex activities; deploying Caesar as a public accessible service and extending Caesar to recognize complex activities in moving cameras (e.g. drones, and cars).

REFERENCES

[1] AVA Actions Dataset. https://research.google.com/ava/.

[2] Dawn of the Smart Surveillance Camera. https://www.zdnet.com/article/dawn-of-the-smart-surveillance-camera/.

[3] Duke Multi-Target, Multi-Camera Tracking Project. http://vision.cs.duke.edu/DukeMTMC/.

[4] Flask Framework. http://flask.pocoo.org/.

[5] GRPC: A High-performance, Open-source Universal RPC Framework. https://grpc.io/.

[6] Interior Wants Wi-Fi At Burning Man. https://www.nextgov.com/cio-briefing/2018/04/interior-wants-wi-fi-burning-man/147852/.

[7] Nvidia GeForce RTX 2080. https://www.nvidia.com/en-us/geforce/graphics-cards/rtx-2080/.

[8] Nvidia Jetson TX2. https://www.nvidia.com/en-us/autonomous-machines/embedded-systems/jetson-tx2/.

[9] Redis. https://redis.io/.

[10] TensorFlow Serving. https://github.com/tensorflow/serving.

[11] UCF 101 Action Dataset. https://www.crcv.ucf.edu/data/UCF101.php.

[12] VIRAT Action Dataset. http://www.viratdata.org/.

[13] WHAT'S WRONG WITH PUBLIC VIDEO SURVEILLANCE? https://www.aclu.org/other/whats-wrong-public-video-surveillance.

[14] M. R. Amer, P. Lei, and S. Todorovic. Hirf: Hierarchical Random Field for Collective Activity Recognition in Videos. In *European Conference on Computer Vision*, pages 572–585. Springer, 2014.

[15] B. B. Amor, J. Su, and A. Srivastava. Action Recognition using Rate-invariant Analysis of Skeletal Shape Trajectories. *IEEE transactions on pattern analysis and machine intelligence*, 38(1):1–13, 2016.

[16] S. M. Aziz and D. M. Pham. Energy Efficient Image Transmission in Wireless Multimedia Sensor Networks. *IEEE communications letters*, 17(6):1084–1087, 2013.

[17] T. Bagautdinov, A. Alahi, F. Fleuret, P. Fua, and S. Savarese. Social Scene Understanding: End-to-end Multi-person Action Localization and Collective Activity Recognition. In *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*, pages 4315–4324, 2017.

[18] S. Bak. *Human re-identification through a video camera network*. PhD thesis, Université Nice Sophia Antipolis, 2012.

[19] J. Boice, X. Lu, C. Margi, G. Stanek, G. Zhang, R. Manduchi, and K. Obraczka. Meerkats: A Power-aware, Self-managing Wireless Camera Network for Wide Area Monitoring. In *Proc. Workshop on Distributed Smart Cameras*, pages 393–422. Citeseer, 2006.

[20] J. Carreira and A. Zisserman. Quo Vadis, Action Recognition? a New Model and the Kinetics Dataset. In *proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*, pages 6299–6308, 2017.

[21] S. Chaudhuri. An overview of query optimization in relational systems. In *Proceedings of the seventeenth ACM SIGACT-SIGMOD-SIGART symposium on Principles of database systems*, pages 34–43. ACM, 1998.

[22] W. Chen, L. Cao, X. Chen, and K. Huang. An Equalized Global Graph Model-based Approach for Multicamera Object Tracking. *IEEE Transactions on Circuits and Systems for Video Technology*, 27(11):2367–2381, 2017.

[23] D. Crankshaw, X. Wang, G. Zhou, M. J. Franklin, J. E. Gonzalez, and I. Stoica. Clipper: A Low-latency Online Prediction Serving System. In *14th USENIX Symposium on Networked Systems Design and Implementation (NSDI 17)*, pages 613–627, 2017.

[24] C. Gan, Y. Yang, L. Zhu, D. Zhao, and Y. Zhuang. Recognizing an Action using Its Name: A Knowledge-based Approach. *International Journal of Computer Vision*, 120(1):61–77, 2016.

[25] R. Girdhar, J. Carreira, C. Doersch, and A. Zisserman. Video Action Transformer Network. *CoRR*, abs/1812.02707, 2018.

[26] D. Gowsikhaa, S. Abirami, et al. Suspicious Human Activity Detection From Surveillance Videos. *International Journal on Internet & Distributed Computing Systems*, 2(2), 2012.

[27] H. Haelterman. *Crime Script Analysis: Preventing Crimes Against Business*. Springer, 2016.

[28] Y. Hu, S. Rallapalli, B. Ko, and R. Govindan. Olympian: Scheduling Gpu Usage in a Deep Neural Network Model Serving System. In *Proceedings of the 19th International Middleware Conference*, pages 53–65. ACM, 2018.

[29] Y. Lee, A. Scolari, B.-G. Chun, M. D. Santambrogio, M. Weimer, and M. Interlandi. PRETZEL: Opening the Black Box of Machine Learning Prediction Serving Systems. In *13th USENIX Symposium on Operating Systems Design and Implementation (OSDI 18)*, pages 611–626, 2018.

[30] W. Liu, D. Anguelov, D. Erhan, C. Szegedy, S. Reed, C.-Y. Fu, and A. C. Berg. Ssd: Single Shot Multibox Detector. In *European conference on computer vision*, pages 21–37. Springer, 2016.

[31] X. Liu, Y. Jiang, P. Jain, and K.-H. Kim. Tar: Enabling Fine-grained Targeted Advertising in Retail Stores. In *Proceedings of the 16th Annual International Conference on Mobile Systems, Applications, and Services*, pages 323–336. ACM, 2018.

[32] P. Mettes and C. G. Snoek. Spatial-aware Object Embeddings for Zero-shot Localization and Classification of Actions. In *Proceedings of the IEEE International Conference on Computer Vision*, pages 4443–4452, 2017.

[33] K. Nithin and F. Brémond. Globality–locality-based Consistent Discriminant Feature Ensemble for Multicamera Tracking. *IEEE Transactions on Circuits and Systems for Video Technology*, 27(3):431–440, 2017.

[34] R. Poppe. A survey on vision-based human action recognition. *Image and vision computing*, 28(6):976–990, 2010.

[35] S. Qi, S. Huang, P. Wei, and S.-C. Zhu. Predicting Human Activities using Stochastic Grammar. In *Proceedings of the IEEE International Conference on Computer Vision*, pages 1164–1172, 2017.

[36] J. Qin, L. Liu, L. Shao, F. Shen, B. Ni, J. Chen, and Y. Wang. Zero-shot Action Recognition with Error-correcting Output Codes. In *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*, pages 2833–2842, 2017.

[37] H. Qiu, K. Chintalapudi, and R. Govindan. Satyam: Democratizing Groundtruth for Machine Vision. *CoRR*, abs/1811.03621, 2018.

[38] H. Qiu, X. Liu, S. Rallapalli, A. J. Bency, K. Chan, R. Urgaonkar, B. S. Manjunath, and R. Govindan. Kestrel: Video Analytics for Augmented Multi-camera Vehicle Tracking. In *2018 IEEE/ACM Third International Conference on Internet-of-Things Design and Implementation (IoTDI)*, pages 48–59, 2018.

[39] J. Redmon and A. Farhadi. Yolo9000: Better, Faster, Stronger. *arXiv preprint arXiv:1612.08242*, 2016.

[40] J. Redmon and A. Farhadi. Yolov3: An Incremental Improvement. *arXiv preprint arXiv:1804.02767*, 2018.

[41] B. Rinner, B. Dieber, L. Esterle, P. R. Lewis, and X. Yao. Resource-aware Configuration in Smart Camera Networks. In *2012 IEEE Computer Society Conference on Computer Vision and Pattern Recognition Workshops*, pages 58–65. IEEE, 2012.

[42] E. Ristani and C. Tomasi. Features for Multi-target Multi-camera Tracking and Re-identification. *arXiv preprint arXiv:1803.10859*, 2018.

[43] M. Sandler, A. Howard, M. Zhu, A. Zhmoginov, and L.-C. Chen. Mobilenetv2: Inverted Residuals and Linear Bottlenecks. In *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*, pages 4510–4520, 2018.

[44] J. C. SanMiguel, C. Micheloni, K. Shoop, G. L. Foresti, and A. Cavallaro. Self-reconfigurable Smart Camera Networks. *Computer*, 47(5):67–73, 2014.

[45] T. K. Sellis. Multiple-query optimization. *ACM Transactions on Database Systems (TODS)*, 13(1):23–52, 1988.

[46] Z. Shou, J. Pan, J. Chan, K. Miyazawa, H. Mansour, A. Vetro, X. Giro-i Nieto, and S.-F. Chang. Online Detection of Action Start in Untrimmed, Streaming Videos. In *Proceedings of the European Conference on Computer Vision (ECCV)*, pages 534–551, 2018.

[47] F. Solera, S. Calderara, E. Ristani, C. Tomasi, and R. Cucchiara. Tracking Social Groups Within and Across Cameras. *IEEE Transactions on Circuits and Systems for Video Technology*, 2016.

[48] C. Su, S. Zhang, J. Xing, W. Gao, and Q. Tian. Deep attributes driven multi-camera person re-identification. In *European conference on computer vision*, pages 475–491. Springer, 2016.

[49] C. Sun, A. Shrivastava, C. Vondrick, K. Murphy, R. Sukthankar, and C. Schmid. Actor-centric Relation Network. In *Proceedings of the European Conference on Computer Vision (ECCV)*, pages 318–334, 2018.

[50] Y. Sun, L. Zheng, Y. Yang, Q. Tian, and S. Wang. Beyond Part Models: Person Retrieval with Refined Part Pooling (and a Strong Convolutional Baseline). In *Proceedings of the European Conference on Computer Vision (ECCV)*, pages 480–496, 2018.

[51] M. Y. K. Tani, A. Lablack, A. Ghomari, and I. M. Bilasco. Events Detection using a Video-surveillance Ontology and a Rule-based Approach. In *European Conference on Computer Vision*, pages 299–308. Springer, 2014.

[52] P. M. Tank and H. A. Patel. Survey on Human Detection Techniques in Real Time Video. *International Journal of Innovative Research in Science, Engineering and Technology*, 7(5):5852–5858, 2018.

[53] Y. T. Tesfaye, E. Zemene, A. Prati, M. Pelillo, and M. Shah. Multi-target Tracking in Multiple Non-overlapping Cameras using Constrained Dominant Sets. *arXiv preprint arXiv:1706.06196*, 2017.

[54] O. Ulutan, S. Rallapalli, C. Torres, M. Srivatsa, and B. Manjunath. Actor Conditioned Attention Maps for Video Action Detection. In *the IEEE Winter Conference on Applications of Computer Vision (WACV)*. IEEE, 2020.

[55] X. Wang, A. Chowdhery, and M. Chiang. Networked Drone Cameras for Sports Streaming. In *2017 IEEE 37th International Conference on Distributed Computing Systems (ICDCS)*, pages 308–318. IEEE, 2017.

[56] N. Wojke and A. Bewley. Deep Cosine Metric Learning for Person Re-identification. *CoRR*, abs/1812.00442, 2018.

[57] N. Wojke, A. Bewley, and D. Paulus. Simple Online and Realtime Tracking with a Deep Association Metric. In *2017 IEEE International Conference on Image Processing (ICIP)*, pages 3645–3649. IEEE, 2017.

[58] X. Xu, T. M. Hospedales, and S. Gong. Multi-task Zero-shot Action Recognition with Prioritised Data Augmentation. In *European Conference on Computer Vision*, pages 343–359. Springer, 2016.

[59] Y. Xu, X. Liu, L. Qin, and S.-C. Zhu. Cross-view People Tracking by Scene-centered Spatio-temporal Parsing. In *AAAI*, pages 4299–4305, 2017.

[60] H. Zhang, G. Ananthanarayanan, P. Bodik, M. Philipose, P. Bahl, and M. J. Freedman. Live Video Analytics At Scale with Approximation and Delay-tolerance. In *14th USENIX Symposium on Networked Systems Design and Implementation (NSDI 17)*, pages 377–392, 2017.

[61] Y. Zhao, Y. Xiong, L. Wang, Z. Wu, X. Tang, and D. Lin. Temporal Action Detection with Structured Segment Networks. In *Proceedings of the IEEE International Conference on Computer Vision*, pages 2914–2923, 2017.

[62] Z. Zheng, X. Yang, Z. Yu, L. Zheng, Y. Yang, and J. Kautz. Joint Discriminative and Generative Learning for Person Re-identification. In *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*, pages 2138–2147, 2019.