Machine Learning in Distributed Edge Networks: Optimizing for Security and Latency in Surveillance Systems

Ameena Golding[1], Katie Li[1], and Rosanna Neuhausler[1, 2]

[1]Department of Electrical Engineering and Computer Science, University of California, Berkeley
[2]Department of Geography, University of California, Berkeley
{*agolding, katieyli, rneuhausler*}*@berkeley.edu*

***Abstract*—Machine learning frameworks in conjunction with hardware advancements has allowed IoT applications to support more data-intensive tasks. The proximity of resources not only garners benefits in terms of latency and bandwidth, but also allows developers to address security concerns of sensitive data through reducing risks associated with the transport of data to remote servers (i.e. bandwidth monitoring and malicious database administrators). This is therefore especially beneficial for improving home and office surveillance systems, whose current configurations place its subjects at risk through streaming all recordings of them directly to a cloud-based service for processing. We propose a method for decreasing the amount of data sent to the cloud in this context, specifically for the task of intruder detection, in order to preserve the security of video data, as well as benefit from decreased latency by processing data on the edge. By utilizing caching techniques on-device, in conjunction with Convolutional Neural Network (CNN) abstractions for minimizing stored data per individual, we present a proof-of-concept built on currently available commercial hardware to experiment and evaluate our design. Our results show that facial recognition done on the edge is a viable solution to the security concerns of current surveillance networks, and also reduces the latency of facial recognition tasks in an IoT context.**

I. Introduction

There are increased risks in leaking sensitive information associated with cloud computing, especially in the context of the Internet of Things (IoT). The increase in the number of paths over which data is transported (i.e. data channels from edge sensors to the cloud) and the concentration of millions of users data into a small number of databases (clouds) has provided hotspots for malicious attacks, such as bandwidth monitoring and risky administrative access. The past few years have seen some disastrous examples of IoT hacking and vulnerabilities that arise from insecure firmware and insecure internet connectivity. In 2016, the largest DDoS attack ever was launched on service provider Dyn via an IoT botnet that used the Mirai malware to have computers continually search the internet for vulnerable IoT devices. That same year, the St. Jude Hospital's implantable cardiac devices were discovered to be vulnerable due to the transmitter that sends data from the device to the cloud being insecure.

One solution to this problem is to shift more computation onto the data sources at the edge, thus reducing the amount of data being streamed to the cloud. Until recently, real-time decision making in IoT systems was bottle-necked by the cost, latency, and power consumption of the vast majority of IoT device hardware that did not have enough processing capabilities to perform significant computation on the edge. These devices instead streamed all data from the device to the cloud to carry out computation there. However, recent developments in the hardware of edge devices that augment IoT processing capabilities means that edge computing is experiencing a resurgence of popularity in both academic and consumer device spaces. Furthermore, the optimization of machine learning algorithms and the increasing capabilities of running them on the edge has created a space for inference on the edge, thus reducing the amount of data needing to be sent to and stored in the cloud. Putting the computation closer to the data sources also potentially significantly reduces the latency of common tasks associated with the edge, like object detection in self-driving cars or face detection in surveillance systems, where near-instant decision-making based on real-time data is extremely critical.

On the consumer side, some especially promising examples of recent advances in edge hardware and ML on the edge include the Apple A12 Bionic announced in September of this year, which is the 64-bit ARM-based system on a chip (SoC) successor to the Apple A11 Bionic and is featured in the iPhone XS, XS MAX and XR. The chip includes dedicated neural network hardware called the "Neural Engine" that can perform up to 5 trillion 8-bit operations per second. It is used for Face ID, Animoji and other machine learning tasks on device that utilize the iPhone's cameras for computer vision-related tasks. The neural engine allows Apple to implement neural network and machine learning in a more energy-efficient manner than using either the main CPU or the GPU [26]. Similarly, at the CloudNext '18 conference this year, Google announced their Edge TPU hardware chip for the deployment of ML models to the edge. They also announced their Cloud IoT Edge service, which extends Google Cloud's powerful data-processing and machine learning capabilities to the edge by allowing users to export models trained in Google Cloud to the edge [22]. This announcement only shortly followed Google's earlier release of their commercial Voice Kit and Vision Kit for performing natural language processing and computer vision tasks respectively on the edge [21]. However, only the Voice Kit was packaged with Google Cloud support, leaving the Vision Kit as a standalone device without cloud connectivity. This meant putting all of the ML model training and inference entirely on the Vision Kit's Vision Bonnet that was designed by Google and features the Intel Movidius MA2450 vision processing unit (VPU) [27].

Earlier this year, Intel also announced support for Windows ML on their Movidius VPU [28]. Similarly, Microsoft Research is working on resource-efficient machine learning for IoT devices, including their ProtoNN and Bonsai algorithms for classification, regression, and other common IoT tasks. These models are trained in the cloud, then can make predictions on device without the need for cloud connectivity [41]. Another example of consumer efforts to advance the state of computer vision at the edge are the Net Runner and TensorIO open-source projects from medical AI company doc.ai. Net Runner is doc.ai's environment for prototyping and evaluating computer vision machine learning models for IoT devices, and ships with MobileNet image classification models [13]. TensorIO is doc.ai's Objective-C wrapper for TensorFlow Lite, designed for development and deployment of ML models on iOS devices [14].

Finally, this year marked the public release of Amazon Web Service's (AWS) DeepLens, a deep-learning enabled camera for developers [46]. The device features a 4 megapixel camera with up to 1080p video as well as an Intel Atom Processor, and ships with device-optimized versions of MXNet and Intel's clDNN library, with support for other deep learning frameworks. With the release of the DeepLens coinciding with the increasing popularity of consumer IoT home surveillance systems like those of the Nest [35], Ring [30] and Wink [51] smart cameras, it will be interesting to see whether these devices tend towards the direction of the DeepLens in the future. Especially for Ring, a company owned by Amazon that has received consumer criticism for having cameras that are too slow as the device relies entirely on cloud connection via WiFi. These exclusively-cloud IoT systems also raise concerns regarding the privacy of stored video and image data in the cloud. This month (December 2018), Facebook Research presented a paper detailing their work on ML inference at the edge [6]. Besides minimizing users' network bandwidth, inference at the edge is used by Facebook in certain Instagram features for real-time machine learning at image capture time.

With these recent hardware and software trends in mind, in this paper we focus specifically on face detection and face recognition in edge-based smart camera surveillance systems. Our main objective is to reduce risks associated with sending and processing confidential information in a cloud-dependent IoT surveillance system, whilst ensuring low latency and reduced bandwidth usage by pushing more computation onto the edge. A secondary goal is to further expand on ideas concerning machine learning for IoT on edge networks.

The rest of the paper is structured as follows. In Section 2, we discuss the current state of surveillance systems, machine learning at the edge, privacy concerns and caching at the edge. Section 3 describes the design of our system and Section 4 goes into the details of the implementation. We evaluate the implemented prototype in Section 5 and look at scalability in Section 6. Finally, we discuss the direction of our future work in Section 7.

II. Related Work

A. Current Surveillance Systems

The number of surveillance cameras is projected to increase by 20% worldwide every year for the next five years [32]. The British Security Industry Association estimates that in 2015, there were between 4-5.9 billion cameras in the UK alone [4]. In 2016, the global video analytics size was valued at around USD 1.6 billion, and is trending to grow as concerns of safety and security of the public increase [40]. However, more cameras does not necessarily mean increased efficiency as now issues of latency and large amounts of video data come into play. There are also many flavors of video surveillance systems on the market in terms of number of cameras and range of mobility, such as one camera systems, many camera systems and moving camera systems. In conjunction with the applications of categorization, such as object tracking, ID re-identification and behavior analysis, there exist many combinations of how video surveillance systems are used [48].

One of the oldest and most widely used type of camera network is the Close-Circuit Television surveillance system (CCTV). Because of limited storage volume, the majority of these cameras use either 15 FPS or 7.5 FPS [48]. With the introduction of IP-based cameras, the resolution and frame rate became adjustable as compressing algorithms can be used to reduce storage volume. These cameras also make it possible to view footage remotely and perform remote analytics on the video data. We compare mainly with IP cameras in our evaluation. Although our evaluation also focuses primarily on images, video surveillance networks also can carry sound and GPS data [48]. For example, the Radar Video Surveillance System offered by Honeywell is able to detect intruders in user defined Alarm Zones using Automatic Identification System (AIS) and GPS filtering [25].

Often, the installation and hardware of the camera network is separate from the analytics. This is the case for IP-based cameras, where the collection of video data occurs locally and the analytics occurs remotely, allowing businesses to quickly convert their security systems to utilize the newest applications available for analyzing video data using video management software available [18]. For instance, Axis Communications offers different flavors of video management software depending on the use case of the customer [10]. With the growing trend of combining both analytics and hardware as a product, companies are now looking into faster querying of video data for object detection/tracking, face recognition, and ID re-identification. Panasonic is set to release FacePRO, its facial recognition system that can be paired with the available cameras it offers to present a service for face matching and search for video data [39]. However, the video data is still transported to the cloud for processing, which is what we hope to eliminate with our design.

B. Machine Learning on Cameras at the Edge

In-part thanks to some of the aforementioned hardware advancements, optimizing machine learning algorithms for the edge has received recent interest in academia. The main areas of interest are in performing inference at the edge in real-time, and in creating 'smart' edge networks that learn how to adapt to certain parameters that are often in flux such as network bandwidth, as well as efficiently distributing tasks across multiple devices in a network.

The topic of self-adaptive, distributed smart cameras (DSCs) that perform computer vision has been long-investigated, particularly in the scope of video transmission to the cloud or across local networks [42]. Today, novel camera networks are expected to perform adaptive algorithms to account for changes in the scene and objects of varying 'importance' to the task at hand. In their 2015 paper, Zhang and Chowdhery et al. presented a distributed wireless video surveillance system called Vigil that leverages edge computing for real-time tracking and surveillance in enterprise campuses and retail stores [53]. A similar architecture to Vigil was explored in earlier research prototypes like IrisNet[38], Bolt[23], and IBM's S3[47] which attempted to coordinate cameras at scale. Vigil intelligently partitions frame processing across the edge and cloud via novel video frame prioritization techniques based on the level of activity in a frame. Their preliminary results showed that reducing the amount of data being streamed over the wireless network (i.e pushing more computation onto the edge) meant that the surveillance system they studied could support a wider geographical area of coverage, between five and 200 times greater than what was possible with streaming video to the cloud alone. The authors report that such a drastic increase in scalability is made possible due to the fact that, in most real-world video footage, nothing of interest is actually happening in a scene. Thus, it makes sense to utilize object detection or face recognition, depending on the surveillance task at-hand, to limit the number of unimportant frames being sent to the cloud and wasting bandwidth in the process. Vigil's edge compute nodes (ECNs) locally process each camera's video feed with lightweight, stateless vision models like motion and object detection. The Vigil algorithm also relies on geometry and known locations of the ECNs to detect redundant viewpoints without actually exchanging the redundant frames, as well as performs object re-identification to eliminate redundant views of the same object across multiple camera frames. However, Vigil does not store a persistent repository of unique face encodings (as tied to identities) on the device, and therefore is not suited for facial recognition or intruder detection tasks. Rather, they use facial re-identification solely to identify in real-time whether two faces from two different frames are the same person or not.

In November of this year (2018), researchers at North China University of Technology built an extended version of person re-identification across surveillance camera frames that utilizes recent advancements in deep convolutional neural networks (CNNs) for person detection [54]. Researchers at the University of California, Berkeley's RISELab are currently working on ReXCam, a system built to enable at-scale video inference in the context of cross-camera analytics at the edge [29]. ReXCam builds a cross-camera correlation model that encodes the locality observed in historical traffic patterns. Again, though, the motivation here is to achieve person tracking across frames and not necessarily person identification, which is the focus of our work.

C. Privacy in Cameras at the Edge

With storing faces on edge IoT devices comes concerns regarding the accessibility of those images by potentially malicious actors, thus threatening the privacy of the individuals in the footage. The past few years have seen different approaches to enabling video privacy on IoT devices. Recently, Yu and Lim et al. published their work on their Pinto system for producing privacy-protected, forgery-proof videos using low-end IoT cameras [52]. Pinto operates by blurring footage in real-time, thereby obscuring faces and potentially sensitive scene elements, before the footage is sent to the cloud. Although readily implementable in IoT devices today, this kind of video post-processing limits the accuracy of facial recognition that can be achieved in the cloud once the video is sent over. Earlier attempts at preserving privacy of video footage include Chattopadhyay and Boult's PrivacyCam, which uses AES public-key encryption to encrypt and selectively blur detected areas of interest in a video frame [8]. However, since this is a significantly earlier paper, PrivacyCam's face detection algorithms are potentially less accurate than what would be possible with today's CNN-based methods.

This year, Wang et al. at Carnegie Mellon University (CMU) published OpenFace, a new open-source face recognition system nearing state-of-the-art accuracy [50]. In fact, we almost used their system in our work but opted for another Python library. Building on OpenFace, RTFace is a mechanism designed by Wang et al. that integrates OpenFace with inter-frame tracking to achieve real-time video denaturing of edge device video streams. Video privacy is achieved by selectively blurring faces in real-time, in a manner akin to the Pinto system, but according to previously specified policies that can be fine-tuned. Unlike in our work where all video processing happens on the edge device itself, RTFace operates by streaming first video data to a local *cloudlet* and performing the denaturing algorithm there. The term 'cloudlet' refers to small data-centers located near IoT devices. As Wang et al. discuss in their paper, prior studies have shown that denaturing on cloudlets (i.e. nearer to the devices themselves) offers greater privacy assurances than denaturing in the cloud [50][12]. This bodes well for our work, where we carry out out all video denaturing on the edge device itself, side-stepping the need to first transport video to an intermediary cloudlet. Furthermore, we extract and store face encodings (fixed-length arrays) that are potentially more secure than images of blurred faces. In the implementation and evaluations section of this paper, we go into further detail on how we process video streams on the edge devices and extract face encodings.

D. Caching at the Edge

Besides concerns around the privacy of footage being stored on device, another key aspect of face recognition on the edge is the latency associated with real-time face detection. The need that exists in current state-of-the-art systems, which is to offload both requests and image data to a local cloudlet or the cloud itself, adds extra latency on top of the time it takes to perform inference on an image. For real-time scenarios such as face recognition and even other tasks like people detection in self-driving cars, it is extremely critical that requests be served with as low user-perceived latency as possible. Since those inference results are then used to make real-time decisions, late results are useless and potentially dangerous.

Due to the general unsuitability of the cloud for serving real-time information back to the edge device, given network connection dependencies, leveraging resources closer to the device is key in order to reduce the response latency of requests. A promising approach that is receiving recent interest is on-device content-level caching [7] [34] to store data or inference results that may be useful (e.g. the identity of a person), much like how a browser might cache often-visited web pages. Caching data on the edge is not a new idea [3], however caching inference results for recognition applications is novel. Drolia et al. have recently presented one of the first works on modelling edge-servers as caches for compute-intensive recognition applications [17]. They have since built Cachier, a system for face recognition that applies adaptive load-balancing between the cloud and a purpose-built edge server cache [16]. Cachier utilizes load-balancing by leveraging the spatiotemporal locality of requests and online analysis of network conditions. Their initial results show a 3x speedup in responsiveness for face recognition tasks on the edge, as compared to non-caching systems. Drolia et al. extend the Cachier idea to build a system called PreCog, for pre-fetching image recognition classifiers from edge servers to perform inference on-device [15]. This kind of pre-model fetching is especially applicable in a camera scenario where multiple different kinds of objects are of interest besides just faces. PreCog uses Markov chains to make predictions about what data to prefetch onto to the device. Markov chains are also used in Sadeghi et al.'s reinforcement-learning-based scheme for caching content in base-station units [43], although clearly the domain is slightly different than IoT cameras. PreCog's initial results are promising, with reduced latency by up to 5 and increased recognition accuracy. However, both PreCog and Cachier operate on best-effort, meaning that they do not provide any real-time guarantees. This is addressed by Hnetynka et al. in their paper on the importance of guaranteed latency applications in edge-cloud environments [24].

In our work, we deviate from the Cachier and PreCog concepts by side-stepping the dedicated edge server and instead building an in-software cache onto the edge camera itself. Unlike in the PreCog scenario, where multiple different types of objects might be of interest (dog, cat, person, car) and therefore different specialized classifiers required, we are for now only concerned with guaranteeing highly accurate face recognition. Therefore we are able to store the inference model on device due to the fact that face recognition only requires one dedicated model to carry out the task. In our setup, pre-fetching specialized models from either the cloud or a nearby edge server would be overkill for the task we are carrying out and would reduce unnecessary overhead. The vast majority today's modern smart cameras, including all the devices mentioned in the introduction of this paper, ship with enough on-device storage and memory to store a single pre-trained model on device.

Glimpse is another real-time object recognition system for camera-equipped mobile devices (i.e not surveillance systems specifically) that uses an 'active cache' of video frames on the mobile device [9]. However, they cache entire frames as their focus is on object tracking, which is a significantly different task than face recognition. Venugopal et al. at IBM recently put forth a proof-of-concept for an edge caching system that is similar to ours [49] (performing inference and caching feature-level identifiers on-device), however their work addresses object detection more broadly, whereas we focus on caching in the context of face recognition specifically.

III. Design

A. State-of-the-art workflows for face recognition at the edge

We chose to build our system on the Amazon Web Services (AWS) DeepLens. The DeepLens hardware comes with 8Gb of RAM, with 6G free under light load, meaning that we have plenty of space to build our cache in memory. AWS DeepLens also comes with a 16GB micro-SD card out-of-the-box; our device is left with 7GB free, which we intend to utilize as our on-device back-up database of face encodings. The Intel Atom E3930 is used for processing, which has 2 cores clocking at 1.30GHz, as well as the Intel i915 graphics card which boasts 100 billion GFLOPS. The camera used for capturing video data is 4-megapixels with MJPEG, and able to record up to 1080p of resolution. AWS DeepLens also supports Wi-Fi connectivity for both 2.4 GHz and 5GHz with standard dual-band networking.

Our decision to use the AWS DeepLens stems from its integration with AWS, which allows for fast prototyping. Unlike the Google Vision Kit, the DeepLens has built-in cloud connectivity with access to the Amazon Rekognition service [44], which is a state of the art facial recognition application that operates in the cloud. It boasts high performance in the cloud in terms of speed and accuracy, so we concluded that it would serve as a worthy comparison to our on-device system. It is likely that home surveillance systems like the Ring, also owned by Amazon, will trend towards smart cameras like the DeepLens as hardware becomes cheaper to manufacture and more available to the public. This is why we decided to evaluate our design using this type of device, as our targeted use case is for office and private area settings where a fully-fledged network of smart cameras is feasible.

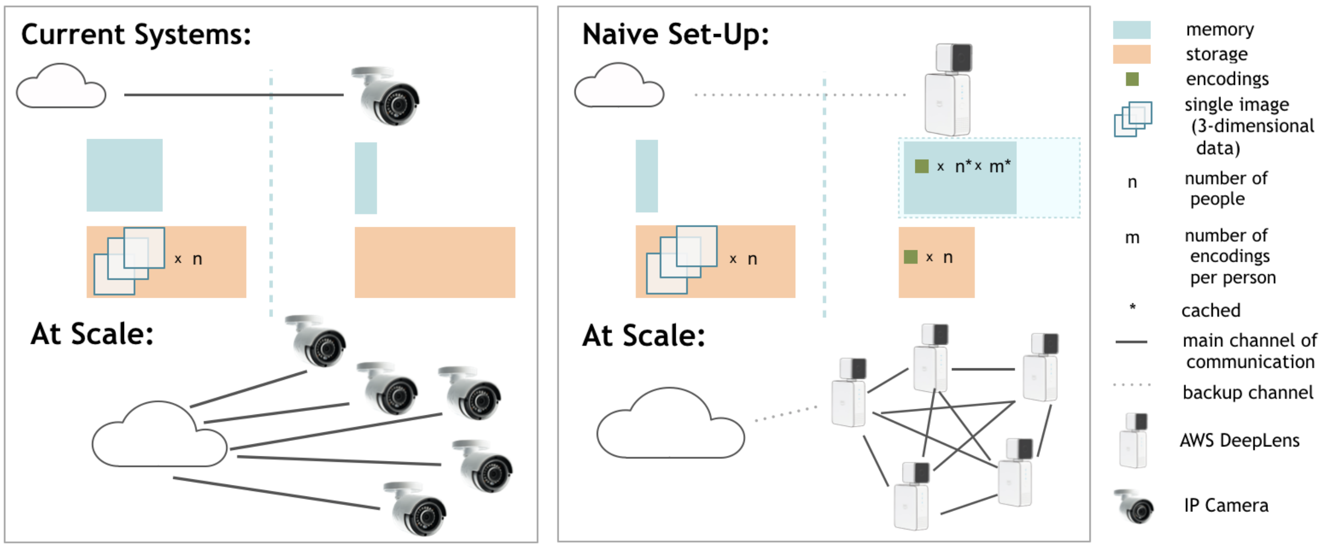


Fig. 1. Current surveillance systems (left) and our proposed setup (right).

Currently, the AWS workflow for doing ML on edge devices involves either storing images on device or sending images to the cloud. Both of these options are expensive, and bad for privacy concerns. Furthermore, AWS currently only supports face *detection* on device (i.e True or False as to whether a face exists in the frame), and does not support face *recognition* on device. Face recognition is a significantly different task, as it involves identifying whether two faces belong to the same person, not just whether or not a face exists in the frame. In order to do face recognition with the pre-existing AWS services, you must make a request from the edge device to the AWS cloud to use their Rekognition face recognition service, which involves sending the image frame containing a face to the cloud.

B. Other Benefits

The AWS face detection model that runs on the DeepLens edge device require images to be saved to the device, which can quickly become expensive as either the images increase in resolution or the device sees more faces over time. This is also bad from a privacy and security perspective, as the faces aren't encoded so it would be relatively easy to hack into the device and steal the face frames.

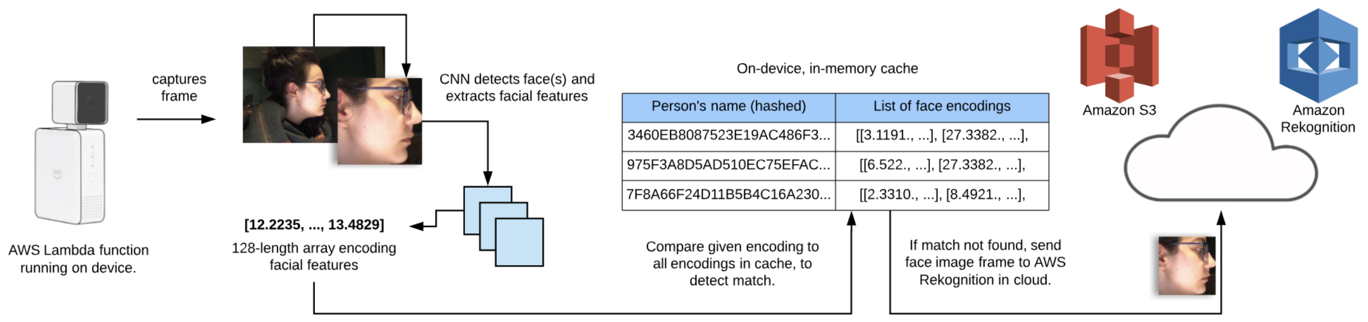
The AWS face detection and face recognition models that run in the cloud require images of faces to be sent to and from the device to the cloud, which has clear privacy concerns (streaming images over WiFi with little to no encryption). Furthermore, as discussed in the section below, Amazon's Rekognition service that operates on images of faces quickly becomes expensive (i.e. latency increases dramatically) as more faces are requested from the service in succession. For example, even the task of having Rekognition identify four faces in a frame takes considerably longer than several seconds (up to 10s), which is simply not fast enough for the kinds of situations in which quick and fast facial recognition via an edge device would be required. Latency is critical in the kinds of applications like surveillance and even home security cameras, where is it critical to identify intruders as fast as possible.

C. Modifications to perform ML at the Edge

We use the Amazon DeepLens hardware and its AWS cloud connection to simulate cloud-independence and dependence. We eliminate the sending of confidential data to the cloud by instead computing the task of intruder detection at the edge. Confidential data is defined as real-time information of individuals that the security system is meant to protect. (i.e. video streams of residents and workers in homes and office settings, respectively). Cloud independence is possible by making the edge smart through running a face recognition machine learning framework, the `face_recognition` module powered by `dlib`, on the cameras generating the surveillance data. This data-heavy task is made possible through the following two techniques:

1) Creating an on-device, software cache in memory to store the most recently seen people (or most frequently seen, depending on the cache policy).
2) Storing face encodings in the cache, rather than image frames of faces.

Technique 1 helps alleviate latency concerns, as now the edge device does not have to make an expensive call to a face recognition service in the cloud every single time it detects a face in the image. Now, if a face is detected, a maximum of one call to the cloud is needed, if the face is one that the device does not yet have stored in the cache. Once this face is stored in the cache, the face recognition can happen entirely on the device, which is orders of magnitudes faster. For example, in situations where a security camera or system of cameras is stationed in the same place and sees many of the same faces


captures frame
AWS Lambda function running on device.
CNN detects face(s) and extracts facial features
[12.2235, ..., 13.4829]
128-length array encoding facial features
On-device, in-memory cache
Compare given encoding to all encodings in cache, to detect match.
If match not found, send face image frame to AWS Rekognition in cloud.
Amazon S3
Amazon Rekognition

Person's name (hashed)	List of face encodings
3460EB8087523E19AC486F3...	[[3.1191., ...], [27.3382., ...],
975F3A8D5AD510EC75EFAC...	[[6.522., ...], [27.3382., ...],
7F8A66F24D11B5B4C16A230...	[[2.3310., ...], [8.4921., ...],

Fig. 2. Edge-cloud face recognition workflow

throughout the day, caching these faces on device significantly reduces both the latency to detect a face and the bandwidth used throughout the day that would have otherwise been spent servicing requests to and from the cloud.

Technique 1 helps alleviate privacy concerns, as now the overall rate and number of images of faces that are sent to and from the cloud is reduced significantly.

Technique 2 helps alleviate latency concerns, because storing just the encodings, as opposed to images, means that checking whether a face exists in the cache is order of magnitudes faster than having to re-calculate encodings every time a face is detected in frame.

Technique 2 helps alleviate privacy concerns, because it is impossible to reverse-engineer an array of encodings back into the original face (in the case of someone attacking a device in an attempt to steal its cache contents).

IV. Implementation

A. *Edge-meets-cloud workflow*

See Figure 2 for an overview of the workflow. A single AWS Lambda function runs on the DeepLens device that handles the capture and display of frames, the image processing of those frames to extract facial encodings, and the caching of those encodings. This Lambda function also facilitates communication with the cloud when needed, via the boto3 library [45], which is the AWS SDK for Python. To make face recognition as fast as possible, we have the capture and display of image frames occur in its own thread, separate from the image processing. For face recognition on the edge device, we utilize the `face_recognition` Python module, which recognizes and manipulates faces, built using `dlibs` state-of-the-art face recognition built with deep learning. The model has an accuracy of 99.38% on the Labeled Faces in the Wild (LFW) benchmark. The module does support GPU acceleration, however it requires nvidia's CUDA library, which cannot be installed on the DeepLens due to insufficient hardware. When installing the `face_recognition` module on the DeepLens device, we drew inspiration from the OneEye Deeplens hackathon project built by medcv [33]. The `face_recognition` module utilizes a pre-trained CNN to extract face-level features from an image of a face, which are then stored in the form of 128-length arrays of floats. These feature arrays are called 'encodings' or 'embeddings'. The CNN will always output feature arrays that are 128-length, irrespective of input image size. We store these face encodings on device, instead of storing entire image frames. If a match cannot be found for a given face encoding that the device has just extrapolated from a scene, a request is sent to the cloud to identity the face.

For face recognition in the cloud, we utilize Amazon's cloud-based Rekognition service, which operates on images of faces. When learning how to make requests to AWS from the DeepLens, we referenced Github user darwaishx's tutorial, which we are grateful for [11]. The Rekognition Collection that is used to identify familiar faces is linked to an S3 database [47] we created that stores known faces as jpeg files. So, in order to make a request to Amazon Rekognition, we have to send the image of the unknown face to the cloud. On the device, we resize the image frame from its full-size to a considerably smaller 300x300 pixels, as this is the frame size that the `face_recognition` model operates on on-device. To keep things consistent, this is also the size of the image that gets sent to the cloud (as opposed to the original frame size captured by the device). An image of this size, at the lowest resolution on the DeepLens (480p) results in an image whose size would be, on average, 30KB. In comparison, the size of one face encoding array stored on-device is around 1120 bytes, which is order of magnitudes smaller than the image size. The pseudocode for our algorithm is below:

B. *Creating the cache*

Our cache is a software cache that is stored in memory with the code for the fastest access. The default capacity of the cache is 10 entries, for 10 unique people. A cache entry consists of a {key:value} pair that represents a unique person, where the key is the hashed name of the person (if known) and the value is a list of face encodings for that person that the device has seen over its lifetime. We store multiple encodings per person to account for the various parameters that might change across images (lighting, angle, position of face) so as to best capture the potential different presentations of a single person. The more encodings per person we store, the

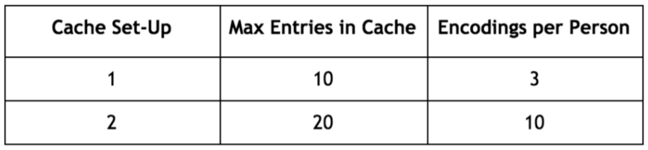
Algorithm 1 On-device recognition algorithm pseudocode

```
procedure RUNINFINITE                                  ▷ Recognizes faces
    $cache \leftarrow Cache$(evictionPolicy is LFU or LRU)
    while $True$ do                                     ▷ While camera capturing
        $locations \leftarrow getFaceLocations(frame)$
        $encodings \leftarrow getFaceEncodings(faceLocations)$
        for face in encodings do
            if face in cache then
                retrieve face
            else
                Send face to cloud
                if face not in cloud then
                    $databaseInCloud \leftarrow$ add face
                    $cache \leftarrow$ add face
```

more likely it is that the cache will score a hit for any given unknown encoding we are attempting to match. However, the length of the encoding lists is capped to ensure that we do not store an encoding for every single frame in which the same person is detected, as this list would quickly grow too large and be very slow to iterate over. We experimented with different values for the optimal encodings list size, and settled on somewhere between 10-20 encodings per person in a cache with 10 entries (for 10 distinct people). This way, we are able to strike a balance between the advantage of having cached results stored close to the code, and the time it takes to linearly iterate over all of the face encodings stored in the cache and call the face_recognition.compare_faces() function for each encoding.

We implemented and tested two caching schemes, least-recently-used (LRU) and least-frequently-used (LFU). We chose these as they made the most sense given our context (surveillance at the edge). In certain situations like in homes, offices and academic settings, it may be the case that many faces pass by an area but that there is a consistent set of faces that appear frequently. In this case, LFU caching makes sense. In other situations like in cafes and shopping malls, there is no concept of 'favorite faces' that continually reappear but rather the same person may linger in the same area for some time before moving on to another area and never returning back to the previous area. In these kinds of situations, LRU caching makes sense because once a camera no longer sees a face, after a certain amount of time it is unlikely to ever see it again. Although we implemented the LRU cache ourselves in order to fine-tune certain parameters, for our LFU cache we used the lfu_cache Python module from Laurent Luce which supports O(1) deletion and insertion [31].

Searching for a matching face in the cache consists of comparing the currently unknown face encoding, grabbed from the current frame, against each list of known encodings per each cache item. Currently, we do this linearly by iterating over each item in the cache. However, given the relatively small cache capacity (10-20 items) and the limit on the number of encodings saved per face (3-20), this process is actually very quick. To compare an unknown encoding to a



Cache Set-Up	Max Entries in Cache	Encodings per Person
1	10	3
2	20	10

Fig. 3. The parameters for the cache are described above.

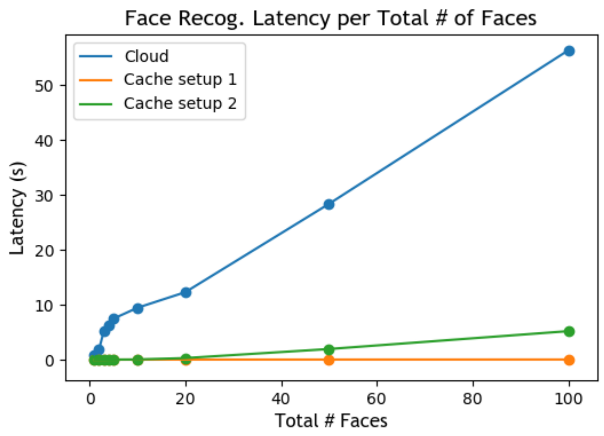


Fig. 4. The above plot compares a pure cloud set-up camera system with two edge-compute models.

known encoding, we use the 'face_recognition.compare_faces' function that returns True if the encodings correspond to the same person, or False if they do not.

C. Security

In addition to storing the encodings of each face instead of the image, the corresponding names of each individual is cryptographically hashed using a SHA256 hash in order to minimize leaking sensitive information under an attack of a malicious attacker. In order for our caching-encoding system to operate, the edge device itself does not 'care' about whether the names associated with encodings have any semantic meaning. All that matters is that the key values of items in the cache are unique, so that our cache can be fully-associative (moderate search speed, but best hit-rate). Hashing not only ensures that each key is distinct, but that individual's names are protected in the instance of an information leak.

D. Failure Tolerance

The DeepLens comes with a 32GB SD card for additional storage once mounted. Initially, we were going to use this as a 'secondary cache' to store even more face encodings. However, due to the linear fashion in which we're currently iterating over the cache to find potential matches, it became clear via some basic experimentation that using persistent storage as a secondary cache would be too slow.

Instead, we opted to use the ample persistent storage to store 'backups' of the face encodings. Since the cache is emptied and reset every time the device boots up, we lose all the face encodings we've ever seen if the device should have to restart.

The benefit of storing the encodings in persistent storage is that in the case of device failure, we can quickly recover all those encodings rather than have to wait for the same person to come back into frame again. This way, we don't lose potentially vital information about who was in a given frame at a given time. We also don't have to send a bunch of requests to the cloud in order to restore the cache back to its previous state.

V. Evaluation

As Figure 4 shows, the scaling behavior of a pure cloud model is not only unsustainable for real-life application, but also heavily dependent on network bandwidth. The experiment was conducted on UC Berkeleys campus network, but relative performance between cloud and edge devices should behave similarly under different network conditions. Figure 3 shows two different cache set-ups utilized in order to determine the optimal configuration. Main concerns with having too many encodings per person involved the delay incurred by the linear comparisons done for each face. The on-device storage is used as a backup for the database of faces in case of power or other failures. Figure 5 highlights that by only storing the encodings, we are able to store around 6 million unique encodings more as opposed to storing images of faces at a given resolution of 480p at 300x300 pixels.

A. Increase in Privacy

By reducing the amount of traffic sent to the cloud for processing, multiple security concerns are addressed. Consider the case where a constant stream of video data is sent to the cloud for facial recognition processing. From network traffic, the current location of individuals appearing in the data can be easily accessed. Although this might be desirable for CCTV video surveillance networks, our targeted use case of intruder detection assumes that the set of people with granted access is known, and that the whereabouts of these individuals should not be trivially revealed regardless of intruders in the system. As we also choose to store the image encodings of individuals instead of pictures, the identity of each permitted person can also be obfuscated as faces cannot be reverse-engineered from encodings.

B. Other Benefits

Our approach also addresses computing concerns, namely latency and failure tolerance of the system. By storing the list of encodings for permitted individuals in persistent storage, intruder detection is able to continue in the event of network failure or powering off of a device. This is especially important in rural areas where network connection is limited. Our results have shown that although it is slower to go to persistent storage first before the cloud, it is nonetheless useful to keep an on-device list of encodings to improve failure tolerance.

VI. Scalability and Multi-camera Networks

So far we have evaluated our system based on the behavior of one device.

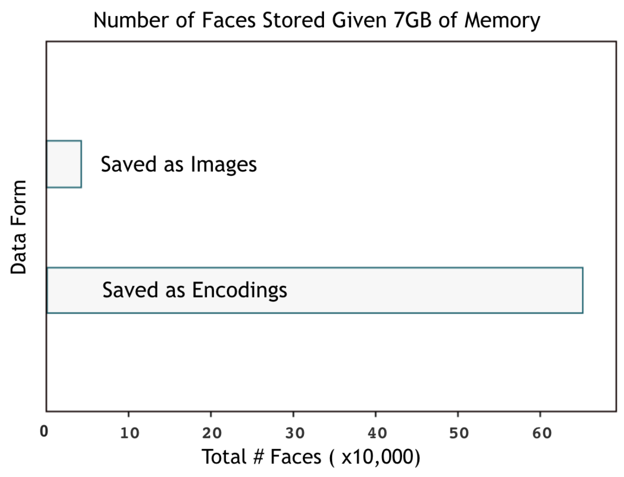


Fig. 5. Number of faces gained from only storing encodings instead of images given 7GB of free memory (this number was chosen based on the amount of memory we had left after loading the model and code onto our device).

In this section we present ideas on how our design might scale through the addition of more cameras, and potentially optimal configurations.

A. Parallelizing for Performance

As the task of intruder detection through face recognition increases in terms of absolute individuals accounted for by the system, both in the cache and from incoming faces, a network of communicating cameras becomes necessary to keep performance at current standards. As previously discussed, current systems have multiple cameras, with each typically set to stream data to the cloud independent of the activities of adjacent cameras. Given the linear behavior of how the facial recognition program is implemented (every new face is compared to each cached encoding), we propose the utilization of the multitude of hardware (i.e. cameras or edge servers) to parallelize our task for reduced time based on number of cameras.

We envision multiple sub-networks of cameras that share the workload of computing, dividing and partitioning the encodings among cameras in the same room. How this data is partitioned, however, is greatly dependent upon the current number of cameras in a room and the frame of view that each camera has. We explore two simple types of potential existing camera topology and provide our recommended architecture for optimal performance.

B. Topology and Configurations

The two types of topologies we explore are as follows, per room:

1) x cameras overseeing the whole room with equal coverage
2) one camera overseeing 70 percent of the room, $x - 1$ adjacent cameras covering hidden angles.

For the first case, we propose the network to follow a general distributed scheme, where encodings are equally distributed among the cameras. Given n individuals stored in the system, each camera will be responsible for testing incoming frames to be compared to n/x encoding sets (a set being the m encodings stored per person). Given a positive match for one of the cameras, the camera with the matching encoding will send said encoding to all the other cameras to be cached in memory for the reminder of the day. This reduces the number of encodings that are known by each camera to purely the encodings being matched that day. The time component is flexible and needs to be further investigated for optimization purposes.

For the latter case, we propose a more centralized system, where the camera with the best coverage (preferably with sight of the entrance to the space), utilizes the processing capabilities of the adjacent cameras, sending requests to the adjacent cameras as and when those backup cameras are needed. This leader camera, which sees the majority of traffic in a room, might distribute its face recognition tasks to those secondary cameras whilst they are not picking up any frames of interest i.e. are idle. This leader camera might also be the sole facilitator of communication between the network of edge devices and the cloud. Further testing needs to be done to determine whether this leader camera should store face encodings of its own, based on the latency of camera to camera communication time (i.e. whether it is efficient for the leader to make recognition requests to the other cameras), or if it is faster for the leader camera to perform face recognition as well. A reminder that this is only centralized at the single room scale; the edge system is still distributed in terms of covering the building at large.

C. Locality

For each target, we can consider how locality of reference can be exploited in order to predict where individuals are located. If we assume the same model of a sub-network of cameras, we examine how **spatial** locality can influence our system. Given the appearance of an individual at Camera A, we can further decrease requests to the cloud if the name-encoding pair is sent to cameras within close proximity of Camera A. This preemptive action can decrease requests sent to the cloud per individual, and thus minimize traffic within potentially untrusted networks.

D. Architecture Evaluation based on Untrusted vs Trusted Networks

Per use-case, the architecture and topology of our system changes depending on whether the network can be trusted or not. In this paper, we focus primarily on a trusted camera network, with an untrusted cloud network. However, if both were non-malicious, we can simplify our design as we would not have to worry about requests sent to the cloud from a privacy perspective. For an untrusted cloud network with a trusted camera network, we envision dedicated nodes whose sole purpose is for communicating with untrusted networks. These nodes will be reinforced with additional security precautions as they serve as the entry point into our trusted network. For instance, these nodes could have hardware-assisted Trusted Execution Environments (TEE) that would ensure isolation of the executed environment even if software on the device is compromised [36]. As an example usage, an application running with Intel's Software Guard eXtension allows for communication between trusted and untrusted components through `ECalls` and `OCalls`.

E. Cost

The Amazon DeepLens currently retails for $249 with AWS services charged separately depending on usage. For our experiment, we used Amazon S3 and Amazon Lambda. For the first 50TB per month, S3 charges $.023 per GB of storage [2]. AWS Lambda offers an initial free tier up to one million requests per month, which translates to 400,000GB-seconds of compute time. After the free request tier expires, each subsequent one million requests gets charged $0.20 [1]. For this experiment, approximately $20 was used to perform our evaluations. A typical video security system for a business can run around $300 per camera, with monthly monitoring fees of around $30 [19]. For a professional monitoring system, additional labor costs may also increase the total price. A more targeted system with facial recognition capabilities such as the Panasonic FacePRO WV-ASF950 is estimated to cost upwards of $1000 simply for the software license [39] [5].

F. Maintainability

As the network of camera grows, the maintainability of our system should remain the same factoring out hardware installations. In order to update software on each edge device, AWS Lambda helps facilitate this process by automatically distributing the code to each connected machine. We envision a central hub that would evaluate the status of each camera through a simple heartbeat protocol.

VII. Future Work

Our work proves that face recognition can be run on a simple edge device set-up. What remains to be accounted for are the other factors involving scalability to completely ensure the feasibility of our system.

A. Intra-Camera Connectivity

A large concern of the current evaluation is how it will scale as more cameras and individuals are added. We theorize that one way of ensuring an efficient system is introducing camera-to-camera communication. This can not only help with decreasing traffic to the cloud as mentioned in part C of Section VI, but also in terms of performance.

By introducing camera connectivity, we can also introduce edge server nodes with a sole purpose of communicating with the cloud. These nodes would be designed with more security precautions in mind than camera nodes within a trusted network as they serve as the entry points for our system. In order to evaluate such a system, it would be useful to examine the different scheduling options in order to not overload one single server node as the system scales. However, we foresee that communication with the cloud should decrease overall as other than the initial sync operation for initializing a new camera, intra-camera connectivity should be used instead to identify unknown targets upon a miss in any given camera's cache.

B. Energy Consumption Evaluations

Our current evaluation of our system does not look at energy consumption and the costs associated with it. The Amazon DeepLens consumes 20 Watts and requires 5V and 4Amps to run. We would like to measure the power consumption under the workload of facial recognition and evaluate how this scales in terms of number of cameras, number of individuals in a given time range, and the monetary cost of these operations.

The Amazon DeepLens currently does not have an external battery, so if power if cut off, there is no way of continuing intruder detection. This introduces a major security concern in our system as intruders can thus cut the power to avoid detection. We would also like to analyze and compare the performance of other edge devices that are configurable with batteries or try to provide external sources of power to prevent complete loss of functionality to the Amazon DeepLens and observe how this scales.

C. Accuracy Analysis

The library used for facial recognition boasts an accuracy of 99.38% on the Labeled Faces in the Wild benchmark [20]. However, depending on the different environments in which our system might implemented, there might be a need to tune the sensitivity of the neural net. We would like to evaluate the accuracy of the library on different sets of faces compiled based on varying degrees of similar or distinct characteristics of facial features. For instance, in populations where facial features are more homogeneous, the tolerance parameter should be adjusted according. Based on these different populations, we aim to show that the number of false positives should decrease after adjusting for irregularities in our known data set. More specifically, as the list of people assigned access to an area is generally always known in an office or campus scenario, the recognition program can always be tuned for optimal performance during installation.

D. Other Specialized Hardware

Although the DeepLens does ship with a GPU, it does not support the NVIDIA CUDA toolkit [37], and so we could not enable GPU acceleration on our model. Furthermore, the DeepLens does not come with any specialized vision processing units (VPUs) either, so we were limited to the CPU only in terms of processing power. We attempted to implement our system on the Google Vision Kit's Intel Movidius VPU, however it turns out that the C++ `dlib` library that powers the `face_recognition` Python module is not compatible with the Movidius hardware. In the future, we would like to explore other options in terms of porting our system to VPUs with specialized ML model support.

VIII. Conclusion

We introduce the current state of surveillance systems, as well as look into trends of machine learning, hardware, and privacy and caching all in the context of edge computing. We also present our implementation and evaluation of facial recognition at the edge using caching and reduced data storage through encodings generated by the `dlib` library. By reducing dependence on the cloud, we address the main concerns of security and privacy by reducing sensitive data sent to untrusted networks. We are also able to decrease latency through our design of caching and improve storage efficiency by saving encodings instead of images. Finally, we present a theoretical analysis into this system at scale in terms of performance, intra-camera connectivity, and cost.

IX. Acknowledgements

We would like to thank Joey Gonzalez and Marcel Neuhausler for introducing us to the issues of privacy in regards security camera streaming and to the concepts around AI at the edge. We would also like to thank Paul Golding for his contributions of the Amazon DeepLens and AWS account resources. Finally, we would like to thank John Kubiatowicz for teaching us advanced topics in computer systems this semester.

References

[1] Amazon Web Services, *AWS Lambda Pricing*, 2018. https://aws.amazon.com/lambda/pricing/.

[2] ______, *AWS S3 Pricing*, 2018. https://aws.amazon.com/s3/pricing/.

[3] Ejder Bastug, Mehdi Bennis, and Merouane Debbah, *Living on the edge: The role of proactive caching in 5g wireless networks*. https://ieeexplore.ieee.org/document/6871674.

[4] BBC News, *CCTV: Too many cameras useless, warns surveillance watchdog Tony Porter*, 2015. https://www.bbc.com/news/uk-30978995/.

[5] BH, *Panasonic FacePro 1-Channel Expansion License*, 2018. https://www.bhphotovideo.com/c/product/1395460-REG/panasonic_wv_asfe901w_facepro_1_channel_expansion_license.html.

[6] Kevin Chen Douglas Chen Sy Choudhury Marat Dukhan Kim Hazelwood Eldad Isaac Yangqing Jia Bill Jia Tommer Leyvand Hao Lu Yang Lu Lin Qiao Brandon Reagen Joe Spisak Fei Sun Andrew Tulloch Peter Vajda Xiaodong Wang Yanghan Wang Bram Wasti Yiming Wu Ran Xian Sungjoo Yoo Peizhao Zhang Facebook Inc. Carole-Jean Wu David Brooks, *Machine learning at facebook: Understanding inference at the edge*, 2018. https://research.fb.com/publications/machine-learning-at-facebook-understanding-inference-at-the-edge/.

[7] Zheng Chang, Lei Lei, Zhenyu Zhou, Shiwen Mao, and Tapani Ristaniemi, *Learn to cache: Machine learning for network edge caching in the big data era*. http://www.eng.auburn.edu/~szm0001/papers/Chang_Cache18.pdf.

[8] Ankur Chattopadhyay and Terrance Boult, *Privacycam: a privacy preserving camera using uclinux on the blackfin dsp*, 200706.

[9] Tiffany Yu-Han Chen, Lenin S. Ravindranath, Shuo Deng, Paramvir Victor Bahl, and Hari Balakrishnan, *Glimpse: Continuous, Real-Time Object Recognition on Mobile Devices*, 13th acm conference on embedded networked sensor systems (sensys), 2015November.

[10] Axis Communications, *Video management software*. https://www.axis.com/products/video-management-software.

[11] darwaishx, *Deep learning with deep lens' github tutorial*. https://github.com/darwaishx/Deep-Learning-With-Deep-Lens.

[12] Nigel Davies, Nina Taft, Mahadev Satyanarayanan, Sarah Clinch, and Brandon Amos, *Privacy mediators: Helping iot cross the chasm*, 201602, pp. 39–44.

[13] doc.ai, *Net runner by doc.ai: Machine learning on the edge*, 2018. https://itunes.apple.com/us/app/net-runner-by-doc-ai/id1435828634?ls=1&mt=8.

[14] ———, *Tensor io: Objective-c wrapper for tensorflow lite*, 2018. https://github.com/doc-ai/TensorIO.

[15] Utsav Drolia, Katherine Guo, and Priya Narasimhan, *Precog: p refetching for image recog nition applications at the edge*, 201710, pp. 1–13.

[16] Utsav Drolia, Katherine Guo, Jiaqi Tan, Rajeev Gandhi, and Priya Narasimhan, *Cachier: Edge-caching for recognition applications*, 201706, pp. 276–286.

[17] ———, *Towards edge-caching for image recognition*, 2017. https://ieeexplore.ieee.org/document/7917629.

[18] Patrick Dugan, *When worlds collide: Ip-based video surveillance on an it network*, 2014. https://er.educause.edu/articles/2014/8/when-worlds-collide-ipbased-video-surveillance-on-an-it-network.

[19] fixr, *Install Video Surveillance Cameras Cost*, 2018. [Online; accessed 11-December-2018].

[20] Adam Geitgey, *face_recognition*, GitHub, 2017. [https://github.com/ageitgey/face_recognition].

[21] Google, *Google vision kit: Do-it-yourself intelligent camera. experiment with image recognition using neural networks.*, 2017. https://aiyprojects.withgoogle.com/vision/.

[22] ———, *Cloud iot edge: Deliver google ai capabilities at the edge.*, 2018. https://cloud.google.com/iot-edge/.

[23] Trinabh Gupta, Rayman Preet Singh, Amar Phanishayee, Jaeyeon Jung, and Ratul Mahajan, *Bolt: Data management for connected homes*, Nsdi, 2014.

[24] Petr Hnetynka, Petr Kubat, Rima Al-Ali, Ilias Gerostathopoulos, and Danylo Khalyeyev, *Guaranteed latency applications in edge-cloud environment*, 201809, pp. 1–4.

[25] Honeywell, *Radar video surveillance (rvs) system*. https://www.honeywellintegrated.com/products/integrated-security/video/97630.html.

[26] Apple Inc, *A12 bionic: The smartest, most powerful chip in a smartphone.*, 2018. https://www.apple.com/iphone-xs/a12-bionic/.

[27] Intel, *Intel movidius myriad vpu 2: A class-defining processor* (2017). https://www.movidius.com/myriad2.

[28] ———, *Intel and Microsoft Enable AI Inference at the Edge with Intel Movidius Vision Processing Units on Windows ML*, 2018. https://www.movidius.com/news/intel-and-microsoft-enable-ai-inference-at-the-edge-with-intel-movidius-vis.

[29] Samvit Jain, Junchen Jiang, Yuanchao Shu, Ganesh Ananthanarayanan, and Joseph Gonzalez, *Rexcam: Resource-efficient, cross-camera video analytics at enterprise scale*, 2018.

[30] Ring Labs, *Ring: Video doorbells and security cameras for your smartphone*, 2018. https://ring.com/.

[31] Laurent Luce, *lfu_cache python module by laurent luce*. https://github.com/laurentluce/lfu-cache.

[32] Martinez, Michael and Cameron, Lori, *Real-Time Video Analytics: The Killer App For Edge Computing* (Computer, ed.), 2017. https://publications.computer.org/computer-magazine/2017/11/14/real-time-video-analytics-for-camera-surveillance-in-edge-computing/.

[33] medcv, *Oneeye deeplens hackathon project*. https://github.com/medcv/OneEyeFaceDetection.

[34] Anselme Ndikumana, Nguyen H. Tran, and Choong Seon Hong, *Deep learning based caching for self-driving car in multi-access edge computing*, CoRR **abs/1810.01548** (2018).

[35] Nest, *Nest: Create a connected home*, 2018. https://nest.com/.

[36] Zhenyu Ning, Jinghui Liao, Fengwei Zhang, and Weisong Shi, *Preliminary study of trusted execution environments on heterogeneous edge platforms*, 201810.

[37] NVIDIA, *Cuda toolkit 10.0 download — nvidia developer*. https://developer.nvidia.com/cuda-downloads.

[38] Y. Ke S. Nath P. B. Gibbons B. Karp and S. Seshan, Irisnet: An architecture for a worldwide sensor web., 2003.

[39] Panasonic, *FacePRO:Panasonic Facial Recognition System*, 2018. https://security.panasonic.com/Face_Recognition/.

[40] Grand View Research, *Video Analytics Market Size Report By Type (Software, Hardware), By Deployment (Cloud, On-premise), By Application (Intrusion Detection, Crowd Management, Facial Recognition), By End Use, And Segment Forecasts, 2018 - 2025*, 2018. https://www.grandviewresearch.com/industry-analysis/video-analytics-market.

[41] Microsoft Research, *Resource-efficient ml for edge and endpoint iot devices*, 2018. https://www.microsoft.com/en-us/research/project/resource-efficient-ml-for-the-edge-and-endpoint-iot-devices/.

[42] Bernhard Rinner and Wayne Wolf, *An introduction to distributed smart cameras*, 200810.

[43] Alireza Sadeghi, Fatemeh Sheikholeslami, and Georgios B. Giannakis, *Optimal and scalable caching for 5g using reinforcement learning of space-time popularities*, IEEE Journal of Selected Topics in Signal Processing (201707).

[44] Amazon Web Services, *Amazon Rekognition: Easily add intelligent image and video analysis to your applications.* https://aws.amazon.com/rekognition/.

[45] ———, *boto3: Python package for aws sdk*. https://boto3.amazonaws.com/v1/documentation/api/latest/guide/quickstart.html.

[46] ———, *Amazon web services deeplens*, 2018. https://aws.amazon.com/deeplens/.

[47] Ying-li Tian, Lisa M. G. Brown, Arun Hampapur, Max Lu, Andrew Senior, and Chiao-Fe Shu, *Ibm smart surveillance system (s3): Event based video surveillance system with an open and extensible framework*, Mach. Vis. Appl. **19** (200810), 315–327.

[48] Vassilios Tsakanikas and Tasos Dagiuklas, *Video surveillance systems-current status and future trends*, Computers Electrical Engineering **70** (2018), 736 –753.

[49] Srikumar Venugopal, Michele Gazzetti, Yiannis Gkoufas, and Kostas Katrinis, *Shadow puppets: Cloud-level accurate AI inference at the speed and economy of edge*, USENIX workshop on hot topics in edge computing (hotedge 18), 2018.

[50] Junjue Wang, Brandon Amos, Anupam Das, Padmanabhan Pillai, Norman Sadeh, and Mahadev Satyanarayanan, *Enabling live video analytics with a scalable and privacy-aware framework*, ACM Transactions on Multimedia Computing, Communications, and Applications **14** (201806), 1–24.

[51] Labs Inc. Wink, *Wink: A simpler, smarter home*, 2018. https://www.wink.com/products/.

[52] Hyunwoo Yu, Jaemin Lim, Kiyeon Kim, and Suk-Bok Lee, *Pinto: Enabling video privacy for commodity iot cameras*, 201810, pp. 1089–1101.

[53] Aakanksha Victor) Bahl Paramvir Jamieson Kyle Banerjee Suman Zhang Tan Chowdhery, *The design and implementation of a wireless video surveillance system.*, 2015. https://www.cs.princeton.edu/~kylej/papers/com287-zhang.pdf.

[54] Shilin Zhang and Hangbin Yu, *Person re-identification by multi-camera networks for internet of things in smart cities*, IEEE Access **PP** (201811), 1–1.