Robotics: Science and Systems 2019
Freiburg im Breisgau, June 22-26, 2019

Network Offloading Policies for Cloud Robotics: a Learning-based Approach

Sandeep Chinchali*, Apoorva Sharma†, James Harrison§, Amine Elhafsi†, Daniel Kang*,
Evgenya Pergament‡, Eyal Cidon‡, Sachin Katti*‡, Marco Pavone†
Departments of Computer Science*, Aeronautics and Astronautics†, Electrical Engineering‡, and Mechanical Engineering§
Stanford University, Stanford, CA
{csandeep,apoorva,jharrison,amine,ddkang,evgenyap,ecidon,skatti,pavone}@stanford.edu

***Abstract*—Today's robotic systems are increasingly turning to computationally expensive models such as deep neural networks (DNNs) for tasks like localization, perception, planning, and object detection. However, resource-constrained robots, like low-power drones, often have insufficient on-board compute resources or power reserves to scalably run the most accurate, state-of-the art neural network compute models. *Cloud robotics* allows mobile robots the benefit of offloading compute to centralized servers if they are uncertain locally or want to run more accurate, compute-intensive models. However, cloud robotics comes with a key, often understated cost: communicating with the cloud over congested wireless networks may result in latency or loss of data. In fact, sending high data-rate video or LIDAR from multiple robots over congested networks can lead to prohibitive delay for real-time applications, which we measure experimentally. In this paper, we formulate a novel *Robot Offloading Problem* — how and when should robots offload sensing tasks, especially if they are uncertain, to improve accuracy while minimizing the cost of cloud communication? We formulate offloading as a sequential decision making problem for robots, and propose a solution using deep reinforcement learning. In both simulations and hardware experiments using state-of-the art vision DNNs, our offloading strategy improves vision task performance by between 1.3-2.3× of benchmark offloading strategies, allowing robots the potential to significantly transcend their on-board sensing accuracy but with limited cost of cloud communication.**

I. Introduction

For autonomous mobile robots such as delivery drones to become ubiquitous, the amount of onboard computational resources will need to be kept relatively small to reduce energy usage and manufacturing cost. However, simultaneously, perception and decision-making systems in robotics are becoming increasingly computationally expensive.

To avoid these restrictions, it is possible for a robot to offload computation or storage to the cloud, where resources are effectively limitless. This approach, which is commonly referred to as *cloud robotics* [29], imposes a set of trade-offs that have hitherto only been marginally addressed in the literature. Specifically, while offloading computation (for example) to the cloud reduces the onboard computing requirements, it may result in latency that could severely degrade performance, as well as information loss or total failure if a network is highly congested. Indeed, even economical querying of cloud resources may quickly overload a network in the case where the data transfer requirements are relatively large (such as high definition (HD) video or LIDAR point clouds) or where multiple robots are operating.

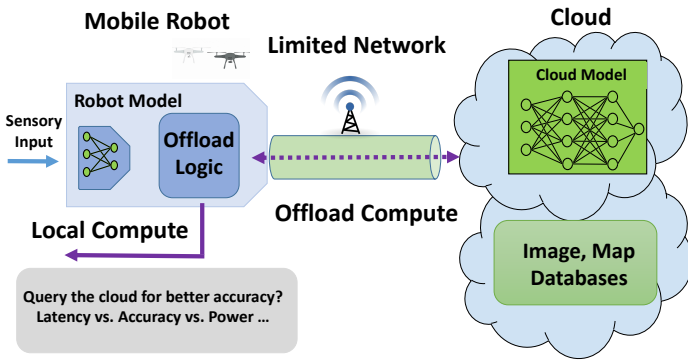


Fig. 1: Autonomous mobile robots are faced with a key tradeoff. Should they rely on local compute models, which could be fast, power-efficient, but less accurate? Or, should they offload computation to a more accurate model in the cloud, which increases latency due to congested networks? We propose a novel algorithmic framework to address such tradeoffs.

In this work, we formally study the decision problem associated with offloading to cloud resources for robotic systems. Given the limitations of real-world networks, we argue that robots should offload only when necessary or highly beneficial, and should incorporate network conditions into this calculus. We view this problem, depicted in Figure 1, as a (potentially partially-observed) Markov Decision Process (MDP) [10], where an autonomous system is deciding whether to offload at every time step.

Contributions and Organization: In Section II, we survey existing work on the offloading problem in robotics and find that it under-emphasizes key costs of cloud communication such as increased latency, network congestion, and load on cloud compute resources, which in turn adversely affect a robot. We further show experimentally that current cloud robotics systems can lead to network failure and/or performance degradation, and discuss how this problem will become more severe in the future without intervention. To address this gap, we formulate a novel cost-based *cloud offloading problem* in Section III, and describe characteristics of this problem that make it difficult to solve with simple heuristics. In Section IV, we propose solutions to this problem based on deep reinforcement learning [49, 50], which are capable of handling diverse

network conditions and flexibly trade-off robot and cloud computation. In Section V, we demonstrate that our proposed approach allows robots to intelligently, but sparingly, query the cloud for better perception accuracy, both in simulations and hardware experiments. To our knowledge, this is the first work that formulates the general cloud offloading problem as a sequential decision-making problem under uncertainty and presents general-purpose, extensible models for the costs of robot/cloud compute and network communication.

II. Background & Related Work

A. Cloud Robotics

Cloud robotics has been proposed as a solution to limited onboard computation in mobile robotic systems, and the term broadly refers to the process of leveraging cloud-based computational resources [29, 19, 54, 28, 37, 12]. For example, a robot might offload processing of sensory input like video, audio, or LIDAR. Concretely, this approach has been used in mapping [38, 37] and localization [43], perception [45], grasping [27, 51], visuomotor control [55, 52], and speech processing [48]. For a review of work in the field, we refer the reader to [28]. Cloud robotics can also include offloading complex decision making to a human, an approach that has been used in path planning [21, 23], and as a backup option for self-driving cars in case of planning failures [33].

In general, the paradigm is useful in any scenario in which there is a tradeoff between performance and computational resources. A notable example of this tradeoff is in perception, a scenario we use as a case-study in this paper. Specifically, vision Deep Neural Networks (DNNs) are becoming the de facto standard for object detection, classification, and localization for robotics. However, as shown in Table I, different DNNs offer varied compute/accuracy tradeoffs. Mobile-optimized vision DNNs, such as MobileNets [46] and ShuffleNets [56], often sacrifice accuracy to be faster and use less power. While MobileNet has lower accuracy, it has significantly fewer parameters and operations than the more accurate Mask R-CNN model, and thus might be favored for an on-robot processing model. A cloud-robotics framework would give improved performance by allowing the robot to query the compute-intensive Mask R-CNN model in the cloud only when the robot model is uncertain.

B. Costs of Offloading

While offloading computation or storage to the cloud has the potential to enable cheap mobile robots to perform increasingly complex tasks, these benefits come at a cost. Querying the cloud is not instant, and there are costs associated with this latency. Furthermore, mobile robots largely use wireless networks (e.g., cellular or WiFi networks), which can be highly stochastic and low bandwidth [44]. Often, the offloaded data can be large relative to this bandwidth: HD video from a single robot can be over 8 megabits per second (Mbps) [57], while cellular networks are often uplink-limited and have between 1-10 Mbps [44, 31] to share across all users.

Current state-of-the-art methods in cloud robotics largely fail to adequately consider these costs. For example, to limit network bandwidth utilization, [38] offload only key-frames (as opposed to all data) in mapping and localization. These key-frames are subsampled at a fixed predetermined frequency, without considering the state of the network connection. In [42], the authors factor in the current state of the system, and hand-design a one-stage decision rule. However, designing such decision rules involves a number of trade-offs and can be difficult. In [45], the authors present a detailed comparison of offloading characteristics in cloud robotics to inform this design process. However, most work, including in the computer systems community, relies on hand-engineering domain-specific offloading solutions without directly optimizing for long-term system-level performance.

Alternative approaches include splitting neural network models across edge devices (in our application, the robot) and the cloud [26], but these may perform poorly under variable network conditions that are likely as a mobile robot navigates through regions of varying signal quality. In this paper, we present an approach to address these hitherto under-emphasized costs in cloud robotics, that incorporates features of the input stream and network conditions in the system-level decision problem of whether or not to offload.

DNN	Accuracy	Size	CPU Infer.	GPU Infer.
MobileNet v1	18	18 MB	270 ms	26 ms
MobileNet v2	22	67 MB	200 ms	29 ms
Mask R-CNN	45.2	1.6GB	325 ms	18 ms

TABLE I: Accuracy, size, and speed tradeoffs of deep neural networks, where accuracy is the standard mean average precision (mAP) metric on the MS-COCO visual dataset [30].

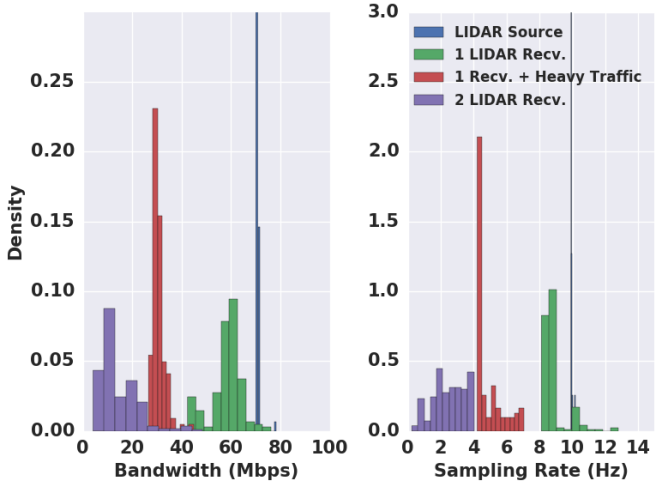


Fig. 2: Streaming LIDAR over WiFi using the Robot Operating System (ROS) produces high-bitrate point cloud streams, which can lead to a server receiving less than half the data (as pictured) or even network failure in the multi-robot setting.

C. Case Study: Costs of Offloading LIDAR Data

To motivate our contributions, we measure the impact of streaming LIDAR over a real WiFi network, which might occur in a cloud-based mapping scenario. The hurdles of streaming

LIDAR over wireless networks have previously been informally described in online forums [3], but, to our knowledge, never rigorously measured in academic literature.

We connected a Velodyne VLP-16 LIDAR to a robotic compute platform (the NVIDIA Jetson Tx2 "source") and streamed LIDAR point clouds in real-time over a *previously uncongested and isolated* WiFi network to a central server (the "receiver") using the standard Robot Operating System (ROS) [41] as a message passing interface.

Figure 2 shows a stark degradation between the sender data-rate (blue) and the data-rate measured at a given receiver (green, red, purple) as we steadily increased network congestion in our controlled experiments. Specifically, at the source (blue), we measured a median 70.48 Mbps data-rate and 9.92 Hz sample frequency, as expected from published Velodyne specifications.

However, at a single receiver without any background network traffic (green), we measured a median data-rate of 59.28 Mbps (only 84% of the sender's). Once we introduced heavy background traffic via simultaneously downloading 69 GB of data over a different machine on the same WiFi network, the congestion became more acute and the LIDAR receiver (red) only measured a data-rate of 30.16 Mbps, which is only 43% of the sender's. Finally, in the most stressful scenario, we simultaneously streamed LIDAR point clouds from two robots to two separate servers on the same network, leading us to only measure 16% of the data-rate at a given receiver (purple).

Since the WiFi link had regular delay statistics, we attribute the low received data-rate to inefficiencies of ROS in handling a large (70.48 Mbps) *sustained* stream. In fact, official ROS documentation for the bandwidth measurement tool `rostopic bw` acknowledges that poor network connectivity and Python, not faster C++, code could be the cause of the receiver not keeping pace with the sender. Though anecdotal, we noticed several users on ROS forums with similar issues for both LIDAR and video.[1]

For one or two LIDAR sender-receiver pairs, the problem we measured in Fig. 2 may be solved by optimizing ROS receiver code. Ideally, one could state-fully encode differences in LIDAR point clouds and use advanced compression, inspired by today's video encoders [24]. However, the point cloud stream of 70.48 Mbps is disconcertingly large, since WiFi networks often attain only 5-100 Mbps [53, 34] while uplink-limited cellular networks often only sustain 1-10 Mbps [1, 44, 31] across users due to congestion or fading environments. Indeed, to stress test the above scenario, we streamed data from several Velodyne sensors over a previously uncongested WiFi network and observed severe delay, dropped ROS messages, and network outages before we could take rigorous measurements.

Our experiments have striking resemblance to issues faced in the computer systems community with streaming HD video to the cloud for computer vision [40, 15]. In the context of robotics, an informal report from Intel estimates that self-driving cars will generate 4 Terabytes of sensor data per day, much more than served by today's cell networks [2]. Even if a subset of this data could be streamed to the cloud, it would place an enormous storage or compute load on cloud services,

[1]The post *ROS Ate My Network Bandwidth!* details similar [3] behaviors.

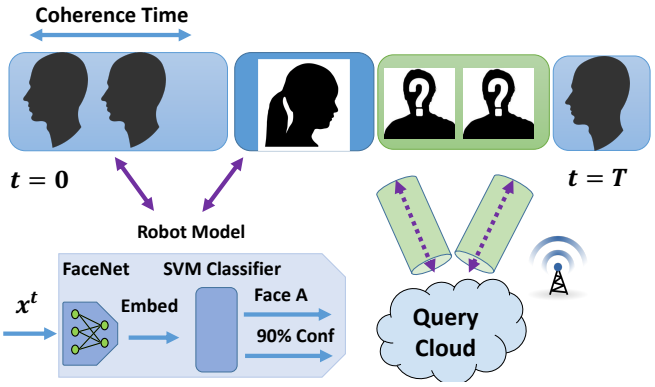


Fig. 3: While our framework is general, we demonstrate it on face recognition from video, a common task for personal assistance robots or search-and-rescue drones. Video surveillance occurs over a finite horizon episode where a robot can use either an optimized local model or query the cloud if uncertain.

potentially exceeding capacity as was the case in the recent widespread outage of Amazon's Alexa speech-processing agent due to an influx of new devices on Christmas day [32]. As more robotics platforms turn to the cloud, such as the *Fetch Robotics* cloud warehouse platform [4], network considerations will play an increasingly important role in system design. We therefore propose an algorithmic framework that allows robotic systems to scalably augment their local compute models with cloud resources.

III. Problem Statement

In this paper, we focus on an abstract cloud robotics scenario, in which a robot experiences a stream of sensory inputs that it must process. At each timestep, it must choose to process the input onboard or to offload the processing to the cloud over a network. In this section, we offer practically motivated abstractions of these ideas, and combine them to formally define the robot-offloading problem.

Sensory Input: We model the raw sensory input into the robot as the sequence $\{x^t\}_{t=0}^{T}$, where x^t represents the data, such as a video frame or LIDAR point cloud, that arrives at time t. While the robot cannot know this sequence in advance, there may be properties of the distribution over these inputs that may guide the robot's offloading policy. For example, this stream may have temporal coherence (see Fig. 3), such as when objects are relatively stationary in video [25, 24], which implies that x^t is similar to x^{t-1}. For example, a person will likely appear in several consecutive frames of a video. However, building a model of coherence can be difficult, since it depends both on the mobility of the robot and dynamism of the background scene. Thus, we turn to model-free approaches, which sidestep modelling temporal coherence and instead can directly learn (comparatively) simple offloading rules.

Computation Models: The computation that we consider offloading to the cloud is the process of estimating some output y^t given some input x^t. For example, in the scenario of processing a video stream, y^t could be a semantically separated version of the input frame x^t (e.g., object detection), useful for

downstream decision making. For the sake of generality, we make no assumptions on what this computation is, and only assume that both the robot and the cloud have models that map a query x^t to predictions of y^t and importantly, a score of their confidence $\texttt{conf}^t$:

$$\hat{y}^t_{\text{robot}}, \texttt{conf}^t_{\text{robot}} = f_{\text{robot}}(x^t)$$
$$\hat{y}^t_{\text{cloud}}, \texttt{conf}^t_{\text{cloud}} = f_{\text{cloud}}(x^t).$$

Typically, f_{robot} is a computationally efficient model suitable for resource-constrained mobile robots. In contrast, f_{cloud} represents a more accurate model which cannot be deployed at scale, for example a large DNN or the decision making of a human operator. The accuracy of these models can be measured through a loss function $\mathcal{L}(y^t, \hat{y}^t)$ that penalizes differences between the predictions and the true results, e.g., the cross entropy loss for classification problems or root mean squared error (RMSE) loss for regression tasks. In the experiments in this paper, we operate in a classification setting, in which confidences are easy to characterize (typically via softmax output layers). However in the regression setting, there are also a wide variety of models capable of outputting prediction confidence [11, 18, 20]. The use of separate, modular robot and cloud models allows a robot to operate independently in case of network failure.

Offload Bandwidth Constraints: The volume of data that can be offloaded is limited by bandwidth, either of the network, or a human operator. We abstract this notion by giving the robot a finite query budget of N_{budget} samples x^t that a robot can offload over a finite horizon of T timesteps. This formalism flexibly allows modeling network bandwidth constraints, or rate-limiting queries to a human. Indeed, the fraction of samples a robot can offload in finite horizon T can be interpreted as the robot's "fair-share" of a network link to limit congestion, a metric used in network resource allocation [17, 39].

These factors impose a variety of tradeoffs to consider when designing an effective offloading policy. Indeed, we can see that the problem of robot offloading can be seen as a sequential decision making problem under uncertainty. Thus, we formulate this problem as a Markov Decision Process (MDP), allowing us to naturally express desiderata for an offloading policy through the design of a cost function, and from there guide the offloading policy design process.

A. The Robot Offloading Markov Decision Process

In this section, we express the generic robot offloading problem as a finite-time MDP

$$\mathcal{M}_{\text{offload}} = (\mathcal{S}_{\text{offload}}, \mathcal{A}_{\text{offload}}, R_{\text{offload}}, \mathcal{P}_{\text{offload}}, T), \quad (1)$$

where $\mathcal{S}_{\text{offload}}$ is the state space, $\mathcal{A}_{\text{offload}}$ is the action space, $R_{\text{offload}} : \mathcal{S}_{\text{offload}} \times \mathcal{A}_{\text{offload}} \to \mathbb{R}$ is a reward function, $\mathcal{P}_{\text{offload}} : \mathcal{S}_{\text{offload}} \times \mathcal{A}_{\text{offload}} \times \mathcal{S}_{\text{offload}} \to [0, 1]$ defines the stochastic dynamics, and T is the problem horizon. In the following section, we define each of these elements in terms of the abstractions of the robot offloading problem discussed earlier. Fig. 4 shows the interplay between the agent (the robot), the offloading policy, and the environment, consisting of the sensory input stream and the robot and cloud prediction models.

Action Space: We consider the offloading decision problem to be the choice of which prediction $\hat{y}$ to use for downstream tasks at time t. The offloading system can either (A) choose to use past predictions and exploit temporal coherence to avoid performing computation on the new input x^t, or (B) incur the computation or network cost of using either the on-device model f_{robot} or querying the cloud model f_{cloud}. Specifically, we have four discrete actions:

$$a^t_{\text{offload}} = \begin{cases} 0, \text{ use past robot prediction } \hat{y}^t = f_{\text{robot}}(x^{\tau_{\text{robot}}}) \\ 1, \text{ use past cloud prediction } \hat{y}^t = f_{\text{cloud}}(x^{\tau_{\text{cloud}}}) \\ 2, \text{ use current robot prediction } \hat{y}^t = f_{\text{robot}}(x^t) \\ 3, \text{ use current cloud prediction } \hat{y}^t = f_{\text{cloud}}(x^t) \end{cases} \quad (2)$$

where $\tau_{\text{robot}} < t$ is the last time the robot model was queried, and $\tau_{\text{cloud}} < t$ is the last time the cloud model was queried.

State Space: We define the state in the offload MDP to contain the information needed to choose between the actions outlined above. Intuitively, this choice should depend on the current sensory input x^t, the stored previous predictions, a measure of the "staleness" of these predictions, and finally, the remaining query budget. We choose to measure the staleness of the past predictions by their age, defining $\Delta t_{\text{robot}} = t - \tau_{\text{robot}}$ and $\Delta t_{\text{cloud}} = t - \tau_{\text{cloud}}$. Formally, we define the state in the offloading MDP to be:

$$s^t_{\text{offload}} = [\ \underbrace{\phi(x^t)}_{\text{features of input}}, \underbrace{f_{\text{robot}}(x^{\tau_{\text{robot}}})}_{\text{past robot}}, \underbrace{f_{\text{cloud}}(x^{\tau_{\text{cloud}}})}_{\text{past cloud}}, \quad (3)$$
$$\underbrace{\Delta t_{\text{robot}}}_{\text{last robot query}}, \underbrace{\Delta t_{\text{cloud}}}_{\text{last cloud query}}, \underbrace{\Delta N_{\text{budget}}}_{\text{remaining queries}}, \underbrace{T - t}_{\text{time left}}\ ].$$

Note that the sensory input x^t may be high-dimensional, and including it directly in the planning problem state could yield an extremely large state-space. Instead, we consider including features $\phi(x^t)$ that are a function of the inputs. We note that in place of our choice of input representation, these state elements may be any summary of the input stream. The specific choice is context dependent and depends on the expense associated with utilizing the chosen features, as well as standard encodings or feature mappings. We describe the choice of practical features ϕ in Section V.

Dynamics: The dynamics in the robot offloading MDP capture both the stochastic evolution of the sensory input, as well as how the offloading decisions impact the other state elements such as the stored predictions and the query budget. The evolution of x^t is independent of the offloading action, and follows a stochastic transition model that is domain-specific. For example, the evolution of video frames or LIDAR point clouds depends on the coherence of the background scene and robot mobility. The time remaining $T - t$ deterministically decrements by 1 at every timestep. The other state variables' transitions depend on the chosen action.

If $a^t_{\text{offload}} \in \{0, 1\}$, then the past prediction elements of the state do not change, but we increment their age by one. If $a^t_{\text{offload}} = 2$, meaning we used the current on-robot model, then we update the stored robot model prediction f_{robot} and

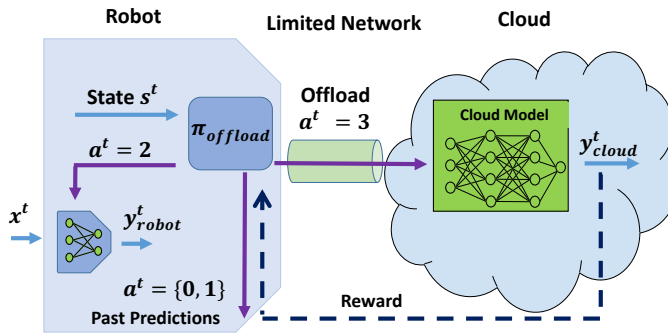


Fig. 4: We depict our novel Robot Offloading MDP, where a robot uses an on-board offloading policy to select if it should use cached predictions, query a local model, or incur the cost, but also reap the accuracy benefits, of querying the cloud.

reset its age to $\Delta t_{\text{robot}} = 0$. Similarly, if we choose to query the cloud model, $a^t_{\text{offload}} = 3$, then we update the stored f_{cloud} prediction and reset its age to $\Delta t_{\text{cloud}} = 0$, and also decrement the query budget N_{budget} by 1.

The modelling of the network query budget is directly based on our measurements (Fig. 2) and recent work in the systems community on network congestion [44, 40, 15]. Our use of sequential features is inspired by the coherence of video frames [25, 24], which we also measured experimentally and observed for LIDAR point clouds.

Reward: We choose the reward function in the MDP to express our objective of achieving good prediction accuracy while minimizing both on-robot computation and network utilization. We can naturally express the goal of high prediction accuracy by adding a penalty proportional to the loss function $\mathcal{L}(y^t, \hat{y}^t)$ under which the cloud and robot models are evaluated. We note, however, that this choice of loss is arbitrary, and a loss derived from some downstream application may be used instead. Indeed, if a scenario is such that misclassification will result in high cost (e.g., misclassifying a human as a stationary object during path planning), this may be incorporated into the MDP reward function. To model the cost of network utilization and computation, we add action costs. This gives us the reward function

$$R^t_{\text{offload}}(s^t, a^t) = -\alpha_{\text{accuracy}} \underbrace{\mathcal{L}(y^t, \hat{y}^t)}_{\text{model error}} - \underbrace{\beta_{\text{cost}}\ \texttt{cost}(a^t)}_{\text{latency, compute}}, \tag{4}$$

where α_{accuracy}, β_{cost} are weights. The costs for network utilization are best derived from the economic analysis of onboard power usage and the cost of bandwidth utilization. For example, a mobile robot with a small battery might warrant a higher cost for querying the onboard model than a robot with a higher battery capacity.

B. The Robot Offloading Problem

Having formally defined the robot offloading scenario as an MDP, we can quantify the performance of an offloading policy in terms of the expected total reward it obtains in this MDP. This allows us to formally describe the general robot offloading problem as:

Problem 1 (Robot Offloading Problem): Given robot model f_{robot}, cloud model f_{cloud}, a cloud query budget of N_{budget} over a finite horizon of T steps, and an offloading MDP $\mathcal{M}_{\text{offload}}$ (Equation 1), find optimal offloading control policy $\pi^*_{\text{offload}} : \mathcal{S}_{\text{offload}} \rightarrow \mathcal{A}_{\text{offload}}$ that maximizes expected cumulative reward R_{offload}:

$$\pi^*_{\text{offload}} \in \underset{\pi_{\text{offload}}}{\text{argmax}}\, \mathbb{E}_{x^0,\ldots,x^T}\left(\sum_{t=0}^{T} R_{\text{offload}}(s^t_{\text{offload}}, a^t_{\text{offload}})\right), \tag{5}$$

where $a^t_{\text{offload}} = \pi_{\text{offload}}(s^t_{\text{offload}})$.

Our MDP formulation, depicted in Fig. 4, is based both on experimental insights and practical engineering abstractions. A key abstraction is the use of separate, modular robot and cloud perception models. Thus, a designer can flexibly trade-off accuracy, speed, and hardware cost, using a suite of pre-trained models available today [46], as alluded to in Table I. Importantly, the robot can always default to its local model in case of network failure, which provides a guarantee on minimum performance.

While we have framed this problem as an MDP, we cannot easily apply conventional tools for exactly solving MDPs such as dynamic programming, as many of the aspects of this problem are hard to analytically characterize, notably the dynamics of the sensory input stream. This motivates studying approximate solution techniques to this problem, which we discuss in the following section.

We emphasize that the framework we present is agnostic to the sensory input modality, and is capable of handling a wide variety of data streams or cost functions. Moreover, the action space can be simply extended if multiple offloading options exist. As such, it describes the generic offloading problem for robotic systems.

IV. A Deep RL Approach to Robot Offloading

Our Approach: We approach the offloading problem using deep reinforcement learning (RL) [49, 50, 35] for several reasons. First, model-free policy search methods such as RL avoid needing to model the dynamics of the system, especially the complex evolution of the robot's incoming sensory inputs. The model-free approach is capable of learning optimal offloading policies based solely on the features included in the state, and avoids needing to predict incoming images. Moreover, the use of a recurrent policy allows better estimation of latent variables defining the possibly non-Markovian context of the incoming images.

In addition to the above benefits, RL enables simple methods to handle stochastic rewards, which may arise in offloading due to variable costs associated with network conditions or load on cloud compute servers. Finally, an RL based approach allows inexpensive evaluation of the policy, as it is not necessary to evaluate dynamics and perform optimization-based action selection as in, e.g., model predictive control [14]. In contrast to these approaches, a deep RL-based approach requires only

evaluating a neural network. Because this policy evaluation is performed as an intermediate step to perception onboard the robot, efficient evaluation is critical to achieving low latency.

We represent the RL offloading policy as a deep neural network and train it using the Advantage Actor-Critic (A2C) algorithm [36]. We discuss the details of the training procedure in the next section. We refer to the policy trained via RL as $\pi^{\mathrm{RL}}_{\mathrm{offload}}$. Note that the training happens pre-deployment, where we are able to provide true labels for the inputs and compute the reward function directly. Upon deployment, $\pi^{\mathrm{RL}}_{\mathrm{offload}}$ is held fixed and not trained further, as the true labels are not available.

Baseline Approaches: We compare this RL-trained policy against the following baseline policies:

1) **Random Sampling** $\pi^{\mathrm{random}}_{\mathrm{offload}}$
This extremely simple benchmark chooses a random $a^t_{\mathrm{offload}} \in \{0, 1, 2, 3\}$ when the cloud query budget is not saturated and, afterwards, chooses randomly from actions $0 - 2$.
2) **Robot-only Policy** $\pi^{\mathrm{all-robot}}_{\mathrm{offload}}$
The robot-only policy chooses $a^t_{\mathrm{offload}} = 2$ at every time-step to query the robot model and can optionally use past robot predictions $a^t_{\mathrm{offload}} = 0$ in between.
3) **Cloud-only Policy** $\pi^{\mathrm{all-cloud}}_{\mathrm{offload}}$ The cloud-only policy chooses $a^t_{\mathrm{offload}} = 3$ uniformly every $\frac{N_{\mathrm{budget}}}{T}$ steps (queries the cloud model) and uses the past cloud predictions $a^t_{\mathrm{offload}} = 1$ in between. Essentially, we periodically sample the cloud model and hold the prediction.
4) **Robot-uncertainty Based Sampling** $\pi^{\mathrm{robot-heuristic}}_{\mathrm{offload}}$
This policy uses robot confidence $\mathtt{conf}^t_{\mathrm{robot}}$ to offload the qth percentile least-confident samples to the cloud as long as the remaining cloud query budget allows.

While approaches 2 and 3 may seem simple, we note that these are the de-facto strategies used in either standard robotics (all robot computations) or standard cloud robotics (all offloading with holds to reduce bandwidth requirements). Robot-uncertainty based sampling is a heuristic that may be used for key-frame selection, analogously to [38].

V. Experimental Performance of Our Deep RL Offloader

We benchmark our proposed RL-based cloud offloading policy within a realistic and representative setting for cloud robotics. Specifically, we focus on a face detection scenario using cutting edge vision DNNs. This scenario is prevalent in robotics applications ranging from search and rescue to robots that assist humans in commercial or industrial settings. More generally, it is representative of an object detection task that is a cornerpiece of virtually any robotics perception pipeline. We test this system with both a simulated input image stream with controlled temporal coherence as well as on a robotic hardware platform with real video streams, and find that in both cases, the RL policy intelligently, and sparingly, queries the cloud to achieve high prediction accuracy while incurring low query costs, outperforming baselines.

Face-detection Scenario: We formulate this scenario, depicted in Fig. 3, in terms of the general abstractions we introduced in Section III. Here, the sensory input stream is a video, where each x^t is a still frame from that video. To avoid training a policy over the large image space directly, we choose the feature encoding ϕ that is used in the state space of the offloading MDP to be the sum of absolute differences between sequential frames. For the on-robot prediction model f_{robot}, we use a combination of FaceNet [47], a widely-used pre-trained face detection model which embeds faces into embedding vectors, together with an SVM classifier over these embeddings. This model has seen success for face detection in live streaming video on embedded devices [8]. For the cloud model f_{cloud}, we use a human oracle, which always gives an accurate prediction with a high confidence measure. We used a zero-one loss function to measure the accuracy of the predictions, with $\mathcal{L}(y^t, \hat{y}^t) = 1$ if the prediction was incorrect, and $\mathcal{L}(y^t, \hat{y}^t) = 0$ if it was correct.

We choose the reward function to balance prediction accuracy and minimize onboard computation, as well as queries to the human operator through the network. The cost of past robot model and cloud queries, denoted by actions $0, 1$, was set to zero ($\mathtt{cost}(0) = \mathtt{cost}(1) = 0$), while the robot model cost was set to $\mathtt{cost}(2) = 1.0$ and the cost of the cloud model was chosen to be $\mathtt{cost}(3) = 5.0$, to especially penalize querying the human oracle who will have limited bandwidth. We tested with different weightings in the reward function (Eqn. 4), and found $\alpha_{\mathrm{accuracy}} = 1.0$ and $\beta_{\mathrm{cost}} = 10.0$ to yield representative performance for our specific cost setup, and therefore report results for this parameter setting. These costs were chosen to incentivize reasonably rational behavior; in real robotic systems they could be computed through an economic cost-benefit analysis [2].

Offloading Policy Architecture: In practice, the input query sequence may show time-variant patterns and the MDP may become nonstationary if the agent only knows the current state. To address this problem using a recurrent policy, we use a Long Short Term Memory (LSTM) [22] as the first hidden layer in the offloader policy to extract a representation over a short history of states. In particular, the actor (or critic) DNN has a LSTM first layer of 64 units, a fully-connected second layer of 256 units, and a softmax (or linear) output layer. We softly enforce the action constraint of disallowing the offloading action when the budget has depleted by having action 3 map to action 2 when $N_{\mathrm{budget}} = 0$. Further, we used standard hyper-parameters for A2C training (described in our extended technical report), and observed stable convergence for a wide variety of MDP reward settings.

A key aspect of this problem is how the *coherence* of the input stream allows the offloading policy to leverage cached predictions to avoid excessively querying the cloud model. In order to test this, we apply the deep RL approach in two scenarios: a synthetic stream of images where coherence was controlled, as well as in a hardware demo which used real video data. In the following subsections, we detail the training and testing procedure for each scenario, and discuss the results.

[2]We provide offloading DNN models and an OpenAI *gym* [13] offloading simulator at https://github.com/StanfordASL/cloud_robotics. An extended technical report is available at https://arxiv.org/abs/1902.05703.

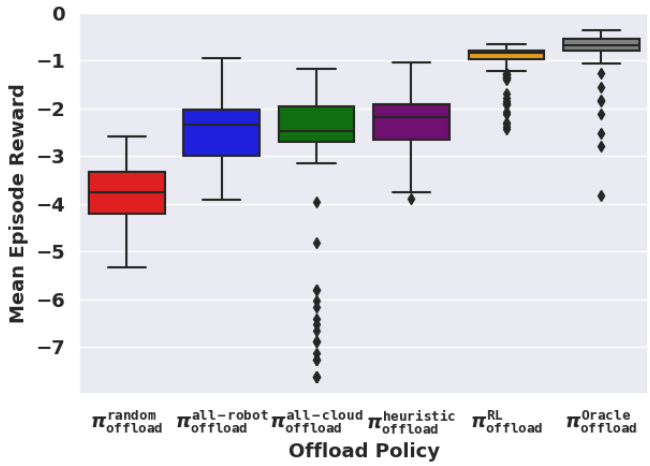


Fig. 5: RL beats benchmark offloading policies by over 2.3× in diverse test episodes over a mixture of network conditions.

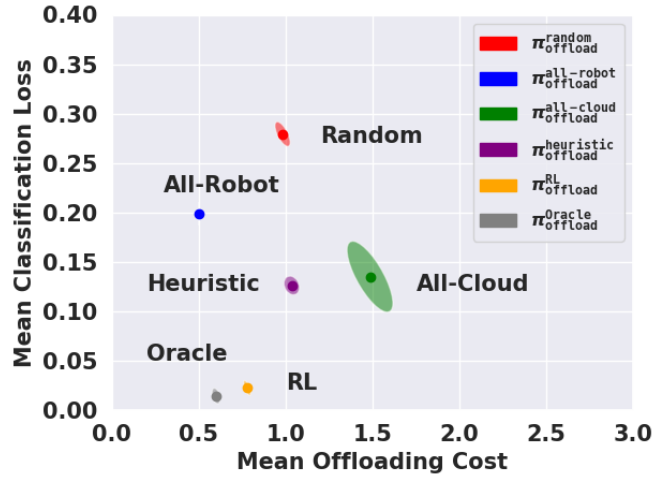


Fig. 6: The reward trades off offloading costs, which penalize network and cloud usage, with classification loss.

A. Synthetic Input Stream Experiments

To model the coherence of video frames we observed experimentally, we divided an episode of T steps into "coherent" sub-intervals, where only various frames of the *same person* appear within one contiguous sub-interval, albeit with different background and lighting conditions. Then, the input stochastically switches to a new individual, who could be unknown to the robot model. As such, we simulate a coherent stream of faces which are a diverse mixture of known and unknown faces to the robot, as shown at the top of Fig. 3. The length of a coherence interval was varied between $1/10 - 1/12$ of an episode duration T to show a diversity of faces in an episode.

Each training trace (episode of the MDP) lasted $T = 80$ steps where a face image (query x^t) arrived at each timestep t. To test RL on a diverse set of network usage limits, we randomly sampled a query budget $\frac{N_{\text{budget}}}{T} \in [0.10, 0.20, 0.50, 0.70, 1.0]$ at the start of each trace.

Evaluation: We evaluated the RL policy and the benchmarks on 100 diverse testing traces each, where the face pictures present in each trace were distinct from those in the set of training traces. To test an offloader's ability to adapt to various network bandwidth constraints, we evaluated it on each trace for every query budget fraction in $\frac{N_{\text{budget}}}{T} \in [0.05, 0.15, 0.30, 0.45, 0.80, 0.90, 0.95]$ (different from the training budgets), simulating novel highly-congested to uncongested networks. We show RL test results for the same representative reward function parameters described above in Section V-A.

RL Intelligently, but Sparingly, Queries the Cloud: Figure 5 shows the distribution of rewards attained by the different offloader policies on all test traces, where our RL approach is depicted in the yellow boxplot. Then, we break down the mean episode reward into its components of prediction accuracy and offloading cost, and show the mean performance over all test traces for each policy in Fig. 6.

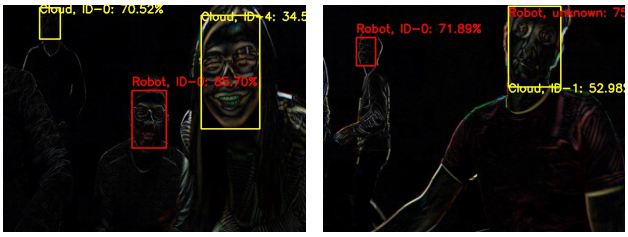
Benchmark policies of random-sampling ($\pi^{\text{random}}_{\text{offload}}$), all-robot compute ($\pi^{\text{all-robot}}_{\text{offload}}$), periodic cloud-compute ($\pi^{\text{all-cloud}}_{\text{offload}}$), and the *best* confidence-threshold based heuristic policy ($\pi^{\text{robot-heuristic}}_{\text{offload}}$) are shown in the left four boxplots (red to purple). An oracle upper-bound solution, which is unachievable in practice since it *perfectly* knows the robot and cloud predictions and future timeseries x^t, is depicted in gray in Figs. 5 - 6.

Fig. 5 shows that RL has at least 2.3× higher median episode reward than the benchmarks. Further, it is competitive with the upper-bound oracle solution, achieving $0.60 - 0.70\times$ its reward depending on experimental settings. This is because the RL policy sparingly queries the costly cloud model, in contrast to an all-cloud policy that incurs significantly higher model query and network cost, as shown in Fig. 6, which plots the mean reward terms and the 95% standard error estimates to display our certainty in the mean estimate. Essentially, RL learns to judiciously query the cloud when the robot model is highly uncertain, which allows it to improve the overall system accuracy and achieve a low prediction loss (Fig. 6). Interestingly, it has better prediction accuracy than an "all-cloud" scheme since bandwidth limits cause this policy to periodically, but sparsely, sample the cloud and hold past cloud predictions. RL learns to conserve precious cloud queries when the query budget $\frac{N_{\text{budget}}}{T}$ is low and use them when they help prediction accuracy the most, thereby achieving low prediction error in Fig. 6.

B. Hardware Experiments

Inspired by deep RL's promising performance on synthetic input streams, we built an RL offloader that runs on the NVIDIA *Jetson Tx2* embedded computer, which is optimized for deep learning and used in mobile robotics. The RL offloader takes in live video from the Jetson camera and runs the small `nn4.small2.v1`[3] FaceNet DNN from the popular OpenFace project [8] and an SVM classifier on selected frames as the

[3]available at https://cmusatyalab.github.io/openface/models-and-accuracies/



(a) FaceNet on a live video stream. (b) Offload yellow faces.

Fig. 7: **Hardware Experiments:** Our offloader, depicted on frame pixel differences, interleaves FaceNet predictions on a robot (red box) and cloud model (yellow) when uncertain.

robot model. OpenFace [8] provides four pre-trained FaceNet models, and we chose to use the smallest, fastest model on the robot, to consider cases where a robot has limited hardware. The smallest model has half the parameters and is $1.6\times$ faster on a GPU than the largest FaceNet model [8].

If the RL agent deems a face needs to be offloaded due to local uncertainty, it can either send the concise Facenet embedding (128-dimensional vector) or the face image to a central server that can run a larger FaceNet DNN and/or SVM classifier trained on many more humans of interest as the cloud model. We use OpenCV for video frame processing, a PyTorch `nn4.small2.v1` OpenFace model, and TensorFlow [5] for the RL offloader neural network policy to achieve real-time processing performance on the embedded Jetson platform.

Data Collection: We captured 6 training and 3 testing videos spanning 9 volunteers. The nine volunteers were known to the robot model but our dataset had 18 distinct, annotated people. Collectively, the training and test datasets showed diverse scenarios, ranging from a single person moving slowly, to dynamic scenarios where several humans move amidst background action.

The Jetson was deployed with a robot FaceNet model and an SVM trained on a subset of images from only 9 people, while the cloud model was trained on several more images of all 9 volunteers to be more accurate. The state variables in Eq. 3 were input to our Deep RL offloader, where the sum of frame *differences* $\sum_{\text{pixels}} |x^t - x^{t-1}|$, rather than a full image, was an important indicator of how quickly video content was changing. Frame differences are depicted in Fig. 7, which helps the offloader subtract background noise and hone in on rapidly-changing faces.

Evaluation and Discussion: As expected, the trained RL offloader queries the cloud when robot model confidence is low, the video is chaotic (indicated by large pixel difference scores), and several hard-to-detect faces appear in the background.

However, it weights such factors together to decide an effective policy and hence is significantly better than a confidence-threshold based heuristic. In fact, the decision to offload to the cloud is only 51.7% correlated (Spearman correlation coefficient) with robot model confidence, showing several factors influence the policy.

In our hardware experiments on real streaming video, our offloader achieved $1.6\times$ higher reward than an all-robot policy, $1.35\times$ better reward than an all-cloud policy, and $0.82\times$ that of an oracle upper bound solution. Further, it attained $1.1 - 1.5\times$ higher reward than confidence-based threshold heuristics, where we did a linear sweep over threshold confidences, which depend on the specific faces in the video datasets.

Videos of our offloading policy show how robot computation can be effectively juxtaposed with cloud computation in Fig. 7. Finally, since the offloader has a concise state space and does not take in full images, but rather a sum of pixel differences as input, it is extremely small (1.5 MB). Essentially, it is an order of magnitude smaller than even optimized vision DNNs (Table I), allowing it to be scalably run on a robot without interfering in perception or control.

VI. Discussion and Conclusions

In this work we have presented a general mathematical formulation of the cloud offloading problem, tailored to robotic systems. Our formulation as a Markov Decision Problem is both general and powerful. We have demonstrated deep reinforcement learning may be used within this framework effectively, outperforming common heuristics. However, we wish to emphasize that RL is likely an effective choice to optimize offloading policies even for modifications of the offloading problem as stated.

Future Work: While there are many theoretical and practical avenues of future work within the cloud robotics setting (and more specifically within offloading), we wish to herein emphasize three problems that we believe are important for improved performance and adoption of cloud robotics techniques. First, we take a finite-time approach, and while the resulting policy could be applied in a receding horizon fashion for applications with sustained robotic deployment, an infinite-time formulation may be more theoretically appealing and is a promising avenue for future work.

Second, we have characterized the offloading problem as an MDP, in which factors such as latency correspond to costs. However, for safety critical applications such as self-driving cars, one may want to include hard constraints, such as bounding the distribution of latency times. This approach would fit within the scope of Constrained MDPs [7], which has seen recent research activity within deep reinforcement learning [16, 6].

Third, we have dealt with input streams that are independent of our decisions in this work. However, the input streams that robotic systems receive are a consequence of the actions that they take. Therefore, a promising extension to improve performance of cloud robotics systems is considering the offloading problem and network characteristics during action selection (e.g., planning or control). Conceptually, this is related to active perception [9], but also incorporates information about network conditions or input stream staleness.

VII. Acknowledgements

Toyota Research Institute ("TRI") provided funds to assist the authors with their research but this article solely reflects the opinions and conclusions of its authors and not TRI or any other Toyota entity.

References

[1] 4g lte speeds vs. your home network. https://www.verizonwireless.com/articles/4g-lte-speeds-vs-your-home-network/, 2013. [Online; accessed 31-Jan.-2019].

[2] Data is the new oil in the future of automated driving. https://newsroom.intel.com/editorials/krzanich-the-future-of-automated-driving/#gs.LoDUaZ4b, 2016. [Online; accessed 30-Jan.-2019].

[3] Ros ate my network bandwidth! https://answers.ros.org/question/256080/ros-ate-my-network-bandwidth/, 2017. [Online; accessed 31-Jan.-2019].

[4] Introducing the fetch cloud robotics platform. https://fetchrobotics.com/products-technology/cloud-robotics-platform-for-warehouse-automation/, 2019. [Online; accessed 14-May.-2019].

[5] Martín Abadi, Paul Barham, Jianmin Chen, Zhifeng Chen, Andy Davis, Jeffrey Dean, Matthieu Devin, Sanjay Ghemawat, Geoffrey Irving, Michael Isard, et al. Tensorflow: A system for large-scale machine learning. In *Proceedings of the OSDI 2016. Savannah, Georgia, USA*, 2016.

[6] Joshua Achiam, David Held, Aviv Tamar, and Pieter Abbeel. Constrained policy optimization. In *International Conference on Machine Learning*, pages 22–31, 2017.

[7] Eitan Altman. *Constrained Markov decision processes*. CRC Press, 1999.

[8] Brandon Amos, Bartosz Ludwiczuk, and Mahadev Satyanarayanan. Openface: A general-purpose face recognition library with mobile applications. Technical report, CMU-CS-16-118, CMU School of Computer Science, 2016.

[9] Ruzena Bajcsy. Active perception. *Proceedings of the IEEE*, 1988.

[10] R. Bellman. A markovian decision process. Technical report, DTIC Document, 1957.

[11] Charles Blundell, Julien Cornebise, Koray Kavukcuoglu, and Daan Wierstra. Weight uncertainty in neural networks. *International Conference on Machine Learning (ICML)*, 2015.

[12] Asil Kaan Bozcuoğlu, Gayane Kazhoyan, Yuki Furuta, Simon Stelter, Michael Beetz, Kei Okada, and Masayuki Inaba. The exchange of knowledge using cloud robotics. *IEEE Robotics and Automation Letters*, 3(2):1072–1079, 2018.

[13] Greg Brockman, Vicki Cheung, Ludwig Pettersson, Jonas Schneider, John Schulman, Jie Tang, and Wojciech Zaremba. Openai gym. *arXiv:1606.01540*, 2016.

[14] Eduardo F Camacho and Carlos Bordons Alba. *Model predictive control*. Springer Science & Business Media, 2013.

[15] Sandeep P. Chinchali, Eyal Cidon, Evgenya Pergament, Tianshu Chu, and Sachin Katti. Neural networks meet physical networks: Distributed inference between edge devices and the cloud. In *ACM Workshop on Hot Topics in Networks (HotNets)*, 2018.

[16] Yinlam Chow, Ofir Nachum, Edgar Duenez-Guzman, and Mohammad Ghavamzadeh. A lyapunov-based approach to safe reinforcement learning. *Neural Information Processing Systems (NIPS)*, 2018.

[17] Behrouz A Forouzan and Sophia Chung Fegan. *TCP/IP protocol suite*. McGraw-Hill Higher Education, 2002.

[18] Yarin Gal, Riashat Islam, and Zoubin Ghahramani. Deep bayesian active learning with image data. In *International Conference on Machine Learning*, 2017.

[19] Ken Goldberg and Ben Kehoe. Cloud robotics and automation: A survey of related work. *EECS Department, University of California, Berkeley, Tech. Rep. UCB/EECS-2013-5*, 2013.

[20] James Harrison, Apoorva Sharma, and Marco Pavone. Meta-learning priors for efficient online bayesian regression. *Workshop on the Algorithmic Foundations of Robotics (WAFR)*, 2018.

[21] Juan Camilo Gamboa Higuera, Anqi Xu, Florian Shkurti, and Gregory Dudek. Socially-driven collective path planning for robot missions. *IEEE Conference on Computer and Robot Vision*, 2012.

[22] Sepp Hochreiter and Jürgen Schmidhuber. Long short-term memory. *Neural computation*, 9(8):1735–1780, 1997.

[23] Ashesh Jain, Debarghya Das, Jayesh K Gupta, and Ashutosh Saxena. Planit: A crowdsourcing approach for learning to plan paths from large scale preference feedback. *IEEE International Conference on Robotics and Automation (ICRA)*, 2015.

[24] Hari Kalva. The h. 264 video coding standard. *IEEE multimedia*, 13(4):86–90, 2006.

[25] Daniel Kang, John Emmons, Firas Abuzaid, Peter Bailis, and Matei Zaharia. Noscope: Optimizing neural network queries over video at scale. *Proc. VLDB Endow.*, 2017.

[26] Yiping Kang, Johann Hauswald, Cao Gao, Austin Rovinski, Trevor Mudge, Jason Mars, and Lingjia Tang. Neurosurgeon: Collaborative intelligence between the cloud and mobile edge. *ACM SIGPLAN Notices*, 52(4):615–629, 2017.

[27] Ben Kehoe, Akihiro Matsukawa, Sal Candido, James Kuffner, and Ken Goldberg. Cloud-based robot grasping with the google object recognition engine. In *Robotics and Automation (ICRA), 2013 IEEE International Conference on*, pages 4263–4270. IEEE, 2013.

[28] Ben Kehoe, Sachin Patil, Pieter Abbeel, and Ken Goldberg. A survey of research on cloud robotics and automation. *IEEE Trans. Automation Science and Engineering*, 12(2): 398–409, 2015.

[29] J Kuffner. Cloud-enabled robots in: Ieee-ras international conference on humanoid robots. *Piscataway, NJ: IEEE*, 2010.

[30] Tsung-Yi Lin, Michael Maire, Serge Belongie, James Hays, Pietro Perona, Deva Ramanan, Piotr Dollár, and C Lawrence Zitnick. Microsoft coco: Common objects in context. In *European conference on computer vision*, pages 740–755. Springer, 2014.

[31] Hongzi Mao, Ravi Netravali, and Mohammad Alizadeh. Neural adaptive video streaming with pensieve. In *Proceedings of the Conference of the ACM Special Interest Group on Data Communication*, pages 197–210. ACM,

2017.

[32] Sarah Marsh. Amazon alexa crashes after christmas day overload. https://www.theguardian.com/technology/2018/dec/26/amazon-alexa-echo-crashes-christmas-day-overload, 2018. [Online; accessed 20-Jan.-2019].

[33] Aarian Marshall. Starsky robotics unleashes its truly driverless truck in florida. *Wired Magazine*. URL https://www.wired.com/story/starsky-robotics-truck-self-driving-florida-test.

[34] Bradley Mitchell. Learn exactly how fast a wi-fi network can move. https://www.lifewire.com/how-fast-is-a-wifi-network-816543, 2018. [Online; accessed 31-Jan.-2019].

[35] Volodymyr Mnih, Koray Kavukcuoglu, David Silver, Alex Graves, Ioannis Antonoglou, Daan Wierstra, and Martin Riedmiller. Playing atari with deep reinforcement learning. *arXiv preprint arXiv:1312.5602*, 2013.

[36] Volodymyr Mnih, Adria Puigdomenech Badia, Mehdi Mirza, Alex Graves, Timothy Lillicrap, Tim Harley, David Silver, and Koray Kavukcuoglu. Asynchronous methods for deep reinforcement learning. In *International Conference on Machine Learning*, pages 1928–1937, 2016.

[37] Gajamohan Mohanarajah, Dominique Hunziker, Raffaello D'Andrea, and Markus Waibel. Rapyuta: A cloud robotics platform. *IEEE Transactions on Automation Science and Engineering*, 12(2):481–493, 2015.

[38] Gajamohan Mohanarajah, Vladyslav Usenko, Mayank Singh, Raffaello D'Andrea, and Markus Waibel. Cloud-based collaborative 3d mapping in real-time with low-cost robots. *IEEE Transactions on Automation Science and Engineering*, 2015.

[39] Jitendra Padhye, Victor Firoiu, and Don Towsley. A stochastic model of tcp reno congestion avoidence and control. 1999.

[40] Chrisma Pakha, Aakanksha Chowdhery, and Junchen Jiang. Reinventing video streaming for distributed vision analytics. In *10th USENIX Workshop on Hot Topics in Cloud Computing (HotCloud 18)*, Boston, MA, 2018. USENIX Association. URL https://www.usenix.org/conference/hotcloud18/presentation/pakha.

[41] Morgan Quigley, Ken Conley, Brian Gerkey, Josh Faust, Tully Foote, Jeremy Leibs, Rob Wheeler, and Andrew Y Ng. Ros: an open-source robot operating system. In *ICRA workshop on open source software*, volume 3, page 5. Kobe, Japan, 2009.

[42] Akhlaqur Rahman, Jiong Jin, Antonio Cricenti, Ashfaqur Rahman, and Dong Yuan. A cloud robotics framework of optimal task offloading for smart city applications. *IEEE Global Communications Conference (GLOBECOM)*, 2016.

[43] Luis Riazuelo, Javier Civera, and JM Martínez Montiel. C2tam: A cloud framework for cooperative tracking and mapping. *Robotics and Autonomous Systems*, 62(4):401–413, 2014.

[44] Haakon Riiser, Paul Vigmostad, Carsten Griwodz, and Pål Halvorsen. Commute path bandwidth traces from 3g networks: Analysis and applications. In *Proceedings of the 4th ACM Multimedia Systems Conference*, MMSys '13, pages 114–118, New York, NY, USA, 2013. ACM. ISBN 978-1-4503-1894-5.

[45] Javier Salmerón-Garcı, Pablo Íñigo-Blasco, Fernando Dı, Daniel Cagigas-Muniz, et al. A tradeoff analysis of a cloud-based robot navigation assistant using stereo image processing. *IEEE Transactions on Automation Science and Engineering*, 12(2):444–454, 2015.

[46] Mark Sandler and Andrew Howard. Mobilenetv2: The next generation of on-device computer vision networks. https://ai.googleblog.com/2018/04/mobilenetv2-next-generation-of-on.html.

[47] Florian Schroff, Dmitry Kalenichenko, and James Philbin. Facenet: A unified embedding for face recognition and clustering. In *Proceedings of the IEEE conference on computer vision and pattern recognition*, pages 815–823, 2015.

[48] Komei Sugiura and Koji Zettsu. Rospeex: A cloud robotics platform for human-robot spoken dialogues. *IEEE International Conference on Intelligent Robots and Systems (IROS)*, 2015.

[49] R.S. Sutton and A.G. Barto. Reinforcement learning: an introduction. *Neural Networks, IEEE Transactions on*, 9 (5):1054–1054, 1998.

[50] C. Szepesvári. Algorithms for reinforcement learning. *Synthesis Lectures on Artificial Intelligence and Machine Learning*, 4(1):1–103, 2010.

[51] Ajay Kumar Tanwani, Nitesh Mor, John Kubiatowicz, Joseph E Gonzalez, and Ken Goldberg. A fog robotics approach to deep robot learning: Application to object recognition and grasp planning in surface decluttering. *arXiv preprint arXiv:1903.09589*, 2019.

[52] Nan Tian, Jinfa Chen, Mas Ma, Robert Zhang, Bill Huang, Ken Goldberg, and Somayeh Sojoudi. A fog robotic system for dynamic visual servoing. *arXiv preprint arXiv:1809.06716*, 2018.

[53] Lochan Verma, Mohammad Fakharzadeh, and Sunghyun Choi. Wifi on steroids: 802.11 ac and 802.11 ad. *IEEE Wireless Communications*, 20(6):30–35, 2013.

[54] J. Wan, S. Tang, H. Yan, D. Li, S. Wang, and A. V. Vasilakos. Cloud robotics: Current status and open issues. *IEEE Access*, 4:2797–2807, 2016. ISSN 2169-3536.

[55] Haiyan Wu, Lei Lou, Chih-Chung Chen, Sandra Hirche, and Kolja Kuhnlenz. Cloud-based networked visual servo control. *IEEE Transactions on Industrial Electronics*, 2013.

[56] Z Xiangyu, Z Xinyu, L Mengxiao, and S Jian. Shufflenet: an extremely efficient convolutional neural network for mobile devices. In *Computer Vision and Pattern Recognition*, 2017.

[57] Youtube. Youtube: Recommended upload encoding settings. https://support.google.com/youtube/answer/1722171?hl=en.