Autonomous Robots (2021) 45:997–1012
https://doi.org/10.1007/s10514-021-09987-4

Network offloading policies for cloud robotics: a learning-based approach

Sandeep Chinchali[1] · Apoorva Sharma[2] · James Harrison[3] · Amine Elhafsi[2] · Daniel Kang[1] · Evgenya Pergament[4] · Eyal Cidon[4] · Sachin Katti[1] · Marco Pavone[2]

Received: 30 January 2021 / Accepted: 25 May 2021 / Published online: 3 July 2021

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 and 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.

Keywords Cloud robotics · Edge computing · Multi-robot systems · Robot perception

Special Issue on Robotics: Science and Systems 2019.

✉ Sandeep Chinchali
csandeep@stanford.edu

Apoorva Sharma
apoorva@stanford.edu

James Harrison
jharrison@stanford.edu

Amine Elhafsi
amine@stanford.edu

Daniel Kang
ddkang@stanford.edu

Evgenya Pergament
evgenyap@stanford.edu

Eyal Cidon
ecidon@stanford.edu

Sachin Katti
skatti@stanford.edu

Marco Pavone
pavone@stanford.edu

[1] Department of Computer Science, Stanford University, Stanford, USA

[2] Department of Aeronautics and Astronautics, Stanford University, Stanford, USA

[3] Department of Mechanical Engineering, Stanford University, Stanford, USA

[4] Department of Electrical Engineering, Stanford University, Stanford, USA

1 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* (Kuffner, 2010), 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.

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 Fig. 1, as a (potentially partially-observed) Markov Decision Process (MDP) (Bellman, 1957), where an autonomous system is deciding whether to offload at every time step.

Contributions and Organization: In Sect. 2, 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 Sect. 3, and describe characteristics of this problem that make it difficult to solve with simple heuristics. In Sect. 4, we propose solutions to this problem based on deep reinforcement learning (Sutton and Barto, 1998; Szepesvári, 2010), which are capable of handling diverse network conditions and flexibly trade-off robot and cloud computation. In Sect. 5, 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. Importantly, we show how our learning-based offloading framework extends beyond cloud robotics in Sect. 6. Specifically, we show how we can naturally adapt to advances in hardware-efficient deep learning to dynamically select between compute-efficient DNNs on the *same device*. 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.

An earlier version of this paper was presented at the Robotics: Science and Systems Conference in 2019 (Chinchali et al., 2019). This extended and revised version shows how our learning-based offloading framework generalizes beyond cloud robotics to recent advances in edge computing. Specifically, we (1) add a new section that demonstrates how to use supervised learning, instead of RL, to learn an offloading algorithm, (2) add an experiment to dynamically switch between fast and slow perception models on a state-of-the-art deep neural network hardware accelerator, and (3) discuss the utility of our algorithm for generalized model selection problems.

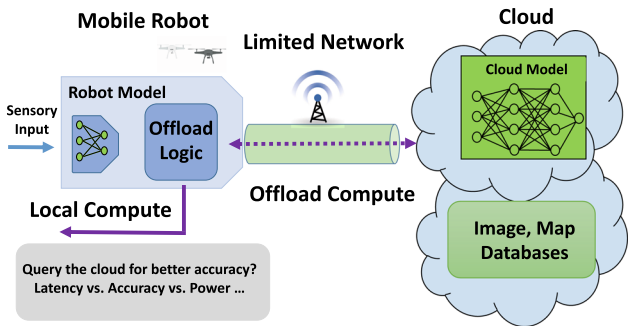


Fig. 1 Autonomous mobile robots are faced with a key tradeoff. Should they rely on local compute models, which could be fast, power-efficient, but less accurate? Or, should they offload computation to a more accurate model in the cloud, which increases latency due to congested networks? We propose a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

2 Background and related work

2.1 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 (Bozcuoğlu et al., 2018; Goldberg and Kehoe, 2013; Kehoe et al., 2015; Kuffner, 2010; Mohanarajah et al., 2015a; Wan et al., 2016). For example, a robot might offload processing of sensory input like video, audio, or LIDAR. Concretely, this approach has been used in mapping (Mohanarajah et al., 2015a, 2015b) and localization (Riazuelo et al., 2014), perception (Salmerón-Garcı et al., 2015), grasping (Kehoe et al., 2013; Tanwani et al., 2019),

Table 1 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 (Lin et al., 2014)

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

visuomotor control (Tian et al., 2018; Wu et al., 2013), and speech processing (Sugiura and Zettsu, 2015). For a review of work in the field, we refer the reader to (Kehoe et al., 2015). Cloud robotics can also include offloading complex decision making to a human, an approach that has been used in path planning (Higuera et al., 2012; Jain et al., 2015), and as a backup option for self-driving cars in case of planning failures (Marshall, 2021).

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 1, different DNNs offer varied compute/accuracy tradeoffs. Mobile-optimized vision DNNs, such as MobileNets (citealtmobilenetcpu) and ShuffleNets (Xiangyu et al., 2017), 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.

2.2 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 (Riiser et al., 2013). 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) (Youtube, 2021), while cellular networks are often uplink-limited and have between 1 and 10 Mbps (Mao et al., 2017; Riiser et al., 2013) 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, Mohanarajah et al. (2015b) 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 Rahman et al. (2016), 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 Salmerón-Garcı et al. (2015), 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. Since the publication of the conference version of this paper, a number of related works (some of which reference the conference paper) have investigated decision-making policies for offloading in cloud robotics, such as Tanwani et al. (2020), Penmetcha and Min (2021).

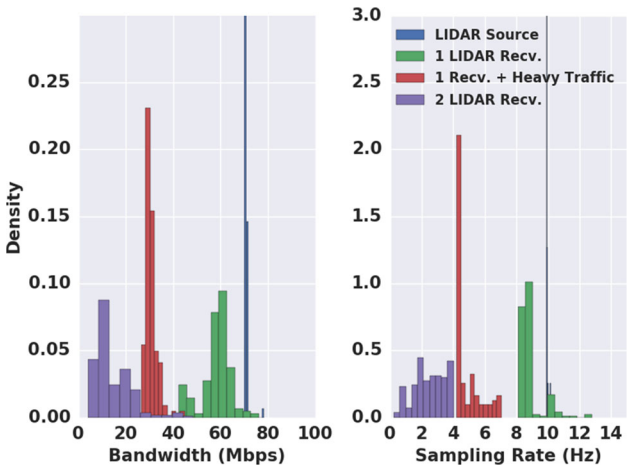


Fig. 2 Streaming LIDAR over WiFi using the Robot Operating System (ROS) produces high-bitrate point cloud streams, which can lead to a server receiving less than half the data (as pictured) or even network failure in the multi-robot setting

Alternative approaches include splitting neural network models across edge devices (in our application, the robot) and the cloud (Kang et al., 2017b), 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.

2.3 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 (ROS Answers, 2017), 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) (Quigley et al., 2009) 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).

Despite the large bandwidth consumption of streaming point-clouds, we also partially 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 statefully encode differences in LIDAR point clouds and use advanced compression, inspired by today's video encoders (Kalva, 2006). However, the point cloud stream of 70.48 Mbps is disconcertingly large, since WiFi networks often attain only 5–100 Mbps (Mitchell, 2018; Verma et al., 2013) while uplink-limited cellular networks often only sustain 1–10 Mbps (Mao et al., 2017; Riiser et al., 2013; Verizon, 2013) 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. These measurements motivate the need to judiciously balance on-board computation with selectively querying the cloud only when necessary for a robotic task.

Our experiments have striking resemblance to issues faced in the computer systems community with streaming HD video to the cloud for computer vision (Chinchali et al., 2018; Pakha et al., 2018). 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 (Intel, 2016). 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, 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 (Marsh, 2018). As more robotics platforms turn to the cloud, such as the *Fetch Robotics* cloud warehouse platform (Fetch Robotics, 2019), 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.

3 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 (Kalva, 2006; Kang et al., 2017a), 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

[1] The post *ROS Ate My Network Bandwidth!* details similar (ROS Answers, 2017) behaviors.

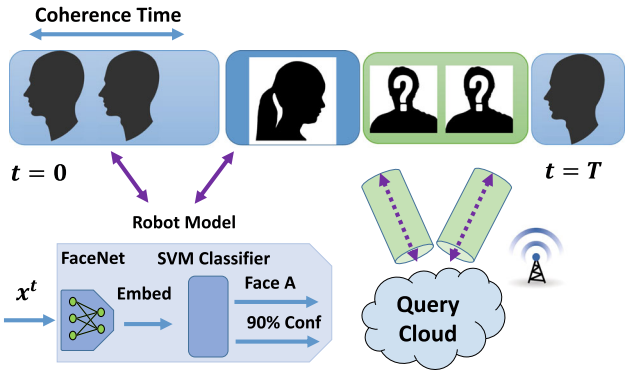


Fig. 3 While our framework is general, we demonstrate it on face recognition from video, a common task for personal assistance robots or search-and-rescue drones. Video surveillance occurs over a finite-horizon episode where a robot can use either an optimized local model or query the cloud if uncertain. To emphasize that video has temporal coherence, we depict how a certain face appears for some time and then the video transitions to another face, as in our synthetic experiments in Sect. 5.1. We also tested our offloading policy on real, complex video scenes, as described in Sect. 5.2

mobility of the robot and dynamism of the background scene. Thus, we turn to model-free approaches, which allow us to use past experience and data to directly learn an offloading policy.

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 (Blundell et al., 2015; Gal et al., 2017; Harrison et al., 2018). 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 (Forouzan and Fegan, 2002; Padhye et al., 1999).

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.

3.1 The robot offloading Markov decision process

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

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

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. Figure 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:

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

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:

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

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 Sect. 5.

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 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 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 (Chinchali et al., 2018; Pakha et al., 2018; Riiser et al., 2013). Our use of sequential features is inspired by the coherence of video frames (Kalva, 2006; Kang et al., 2017a), 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. Furthermore, the cost of an action $\texttt{cost}(a^t)$ could include the compute latency associated with running that specific model, which can be measured as in Table 1.

3.2 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 $\mathcal{M}_{\text{offload}}$ (Eq. 1), find optimal offloading control policy $\pi^*_{\text{offload}} : \mathcal{S}_{\text{offload}} \rightarrow \mathcal{A}_{\text{offload}}$ that maximizes expected cumu-

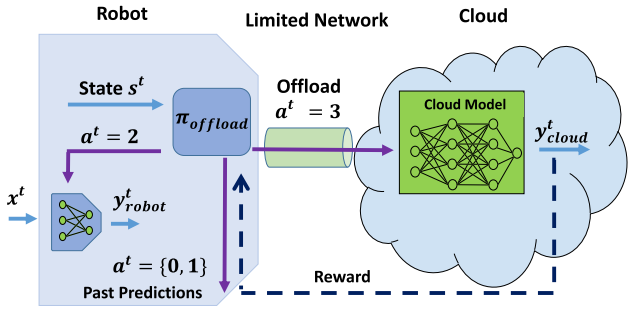


Fig. 4 We depict our novel Robot Offloading MDP, where a robot uses an on-board offloading policy to select if it should use cached predictions, query a local model, or incur the cost, but also reap the accuracy benefits, of querying the cloud. The outputs of a selected computation model, such as y^t_{robot} or y^t_{cloud}, can be used by a robot's planning and decision-making modules

lative 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 (citealtmobilenetcpu), as alluded to in Table 1. 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.

4 A deep RL approach to robot offloading

Our Approach: We approach the offloading problem using deep reinforcement learning (RL) (Mnih et al., 2013; Sutton and Barto, 1998; Szepesvári, 2010) 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 the temporal evolution of a video stream.

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 (MPC) (Camacho and Alba, 2013). In contrast to classical model predictive control 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. While we focus on model-free RL for the above reasons, our general MDP framework could also be solved using model-based RL methods. These methods could employ video dynamics models approximated by deep neural networks.

We represent the RL offloading policy as a deep neural network and train it using the Advantage Actor-Critic (A2C) algorithm (Mnih et al., 2016). We discuss the details of the training procedure in the next section. We refer to the policy trained via RL as $\pi^{\text{RL}}_{\text{offload}}$. 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, $\pi^{\text{RL}}_{\text{offload}}$ is held fixed and not trained further, as the true labels are not available.

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

1. **Random Sampling** ($\pi^{\text{random}}_{\text{offload}}$)
   This extremely simple benchmark chooses a random $a^t_{\text{offload}} \in \{0, 1, 2, 3\}$ when the cloud query budget is not saturated and, afterwards, chooses randomly from actions 0–2.
2. **Robot-only Policy** ($\pi^{\text{all-robot}}_{\text{offload}}$)
   The robot-only policy chooses $a^t_{\text{offload}} = 2$ at every time-step to query the robot model and can optionally use past robot predictions $a^t_{\text{offload}} = 0$ in between.
3. **Cloud-only Policy** ($\pi^{\text{all-cloud}}_{\text{offload}}$)
   The cloud-only policy chooses $a^t_{\text{offload}} = 3$ uniformly every $\frac{N_{\text{budget}}}{T}$ steps (queries the cloud model) and uses the past cloud predictions $a^t_{\text{offload}} = 1$ in between. Essen-

tially, we periodically sample the cloud model and hold the prediction.

4. **Robot-uncertainty Based Sampling** ($\pi_{\text{offload}}^{\text{robot-heuristic}}$) This policy uses robot confidence $\texttt{conf}_{\text{robot}}^{t}$ to offload the q^{th} 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 (Mohanarajah et al., 2015b).

5 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 Sect. 3. 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. The on-robot prediction model f_{robot} consists of a combination of FaceNet (Schroff et al., 2015), 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 (Amos et al., 2016). 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 ($\texttt{cost}(0) = \texttt{cost}(1) = 0$), while the robot model cost was set to $\texttt{cost}(2) = 1.0$ and the cost of the cloud model was chosen to be $\texttt{cost}(3) = 5.0$, 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 $\alpha_{\text{accuracy}} = 1.0$ and $\beta_{\text{cost}} = 10.0$ 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) (Hochreiter and Schmidhuber, 1997) 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 $N_{\text{budget}} = 0$.

We use standard hyper-parameters for A2C training, with an orthogonal initializer and RMSprop gradient optimizer. Specifically, we set the actor learning rate to 10^{-4}, the critic learning rate to 5×10^{-5}, the minibatch size to 20, the entropy loss coefficient to 0.01, and gradient norm clipping to 40. We train A2C over 1 million episodes, with discount factor 0.99 and episode length $T = 80$. We observed stable convergence after 350,000 episodes, consistent over different weightings of the accuracy and loss terms in the reward.

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* (Brockman et al., 2016) offloading simulator at https://github.com/StanfordASL/cloud_robotics. An extended technical report is available at https://arxiv.org/abs/1902.05703.

5.1 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 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 episode.

Evaluation: We evaluated the RL policy and the benchmarks on 100 diverse testing episodes each, where the face pictures present in each episode were distinct from those in the set of training episodes. To test an offloader's ability to adapt to various network bandwidth constraints, we evaluated it on each episode 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 Sect. 5.1.

RL Intelligently, but Sparingly, Queries the Cloud: Fig. 5 shows the distribution of rewards attained by the different offloader policies on all test episodes, 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 episodes for each policy in Fig. 6.

Benchmark policies of random-sampling ($\pi_{\text{offload}}^{\text{random}}$), all-robot compute ($\pi_{\text{offload}}^{\text{all-robot}}$), periodic cloud-compute ($\pi_{\text{offload}}^{\text{all-cloud}}$), and the *best* confidence-threshold based heuristic policy ($\pi_{\text{offload}}^{\text{robot-heuristic}}$) 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 and 6.

Figure 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× 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

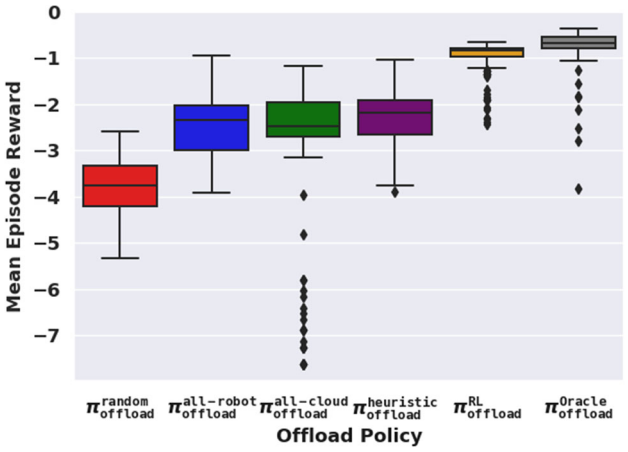


Fig. 5 RL beats benchmark offloading policies by over 2.3× in diverse test episodes over a mixture of network conditions

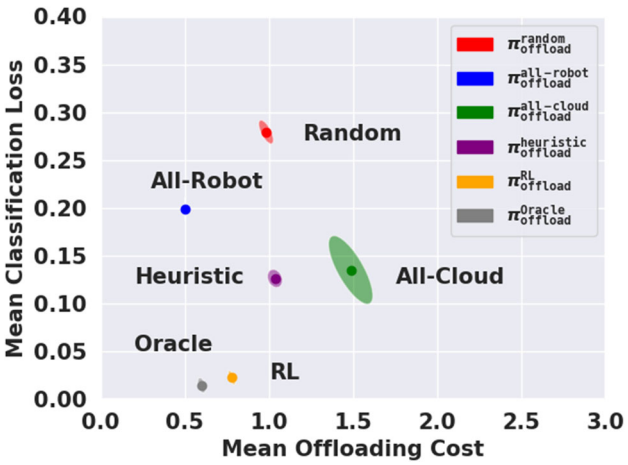


Fig. 6 The reward trades off offloading costs, which penalize network and cloud usage, with classification loss

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.

5.2 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 opti-

mized 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 (Amos et al., 2016) and an SVM classifier on selected frames as the robot model. OpenFace (Amos et al., 2016) 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× faster on a GPU than the largest FaceNet model (Amos et al., 2016).

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 (Abadi et al., 2016) for the RL offloader neural network policy to achieve real-time processing performance on the embedded Jetson platform.

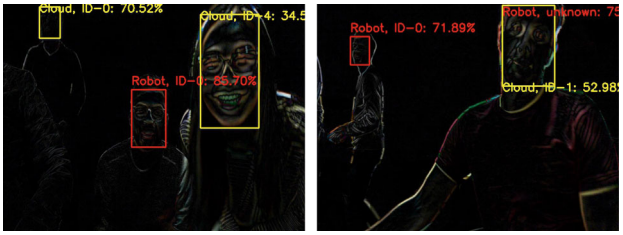
Data Collection: We captured 6 training and 3 testing videos spanning 9 volunteers. Collectively, the training and test datasets showed diverse scenarios, ranging from a single person moving slowly, to dynamic scenarios where several humans move amidst background action.

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 the 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. A hard-to-detect face is one where the local robot model has poor confidence due to insufficient training data or is partially occluded and thus hard to classify in the scene.

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× higher reward than an all-robot policy, 1.35× better reward than an all-cloud policy, and 0.82× that of an oracle upper bound solution. Further, it attained 1.1–1.5× 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.



(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 (Color figure online)

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 1), allowing it to be scalably run on a robot without interfering with perception or control.

6 Beyond cloud robotics: on-device vision model selection

We now demonstrate that our offloading framework also applies to selecting between compute models that are both running on the *same robot*. Our analysis is especially relevant given recent advancements in on-device AI, where techniques such as neural architecture search, weight pruning, and weight quantization are enabling increasingly larger DNNs to run on low-power platforms. Despite these advances, there will still be a fundamental trade-off between model complexity/accuracy and inference latency. This is because more DNN layers, arithmetic operations, and memory access on the same hardware platform will, in general, add to compute delay and consume more power. Thus, our offloading algorithm equally applies to the scenario when we offload from a small DNN to a larger DNN on the *same device*. Ideally, our learning algorithm should achieve the high-accuracy of a large DNN, but the low inference latency of a smaller model through judiciously querying the large DNN. We emphasize that offloading to the cloud will still be extremely useful in robotics, especially for natural language processing tasks that utilize large transformer DNNs.

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

Table 2 Edge TPU DNN models: classification accuracy, size, and speed tradeoffs of quantized DNN vision models

DNN	Accuracy %	Size (MB)	TPU infer. time (ms)
MobileNet v2	83	4	2.67
EfficientNet-Small	92	7	5.18
EfficientNet-Large	93	13	30.1
Inception v3	87	25	51.1

The models all run on the Google Edge TPU accelerator and were trained to classify road scenes using the data in Chinchali et al. (2020)

In this context, our on-device offloader serves to illustrate the generality of our algorithmic offloading framework.

6.1 Compute-efficient vision models

In the subsequent experiment, we demonstrate how our offloading algorithm can select between two vision DNNs running on the Google Edge Tensor Processing Unit (TPU) (Edge TPU, 2019) *without* any cloud support. We use the TPU to complement our prior experiments on the NVIDIA Jetson TX2 platform (Sect. 5.2) and demonstrate the generality of our framework. Our experiments use MobileNet v2 alongside EfficientNet vision DNNs, which are a suite of models that trade-off inference latency and classification accuracy (Tan and Le, 2019). Specifically, the "robot" or small DNN is a MobileNet v2 classifier, and the larger "cloud" DNN is an EfficientNet-Large classifier. Both the small and large EfficientNet variants follow the standard architecture in Tan and Le (2019).

Using transfer learning, we adapt each DNN from a standard model pre-trained on ImageNet to classify challenging road scenes from a prior robotics paper (Chinchali et al., 2020). Specifically, we have three classes consisting of road scenes that have a rare object of a self-driving car, construction site, or standard road scenarios devoid of the first two classes. Table 2 illustrates the trade-off between classification accuracy and inference latency for the small and large EfficientNet variants as well as other models. All models were quantized and compressed using Tensorflow Lite to run on the Edge TPU.

6.2 Training the offloader

To demonstrate the generality of our learning-based approach, we show how our framework can also be used to learn an offloader by a simple supervised learning algorithm. Such a simple learning algorithm is useful for practitioners who might want to avoid training an RL agent, which often has poor sample complexity. Given the extremely low-latency of running the MobileNet v2 classifier on the TPU (Table 2), it is feasible to run the small model for every frame. Rather than learning the temporal coherence of the video stream, we formulate a supervised learning problem where the offloader learns to choose the fast model with action $a_{\text{offload}} = 0$ or slow model with $a_{\text{offload}} = 1$. Further, we do not have a cloud-querying budget restriction from the robot offloading MDP, so $\frac{N_{\text{budget}}}{T} = 1.0$. This is because a budget is suited to tasks requiring communication over a bandwidth-limited wireless link. Instead, we express a roboticist's preference for accuracy and latency via the factors α_{accuracy} and β_{cost} in Eq. 4. Given that we can run the fast model per frame and do not have to allocate a cloud-querying budget across several steps, we can make decisions independently across time and solely based on the predictions for a given image x. Hence, we drop the dependence on time t in the following description of our supervised-learning offloader.

Given a set of training videos, we can record images x and outputs of both the slow and fast models $f_{\text{robot}}(x)$ and $f_{\text{cloud}}(x)$. Further, we can define an optimal action a^*_{offload} by applying the objective in Eq. 4 to the case with two, time-independent actions of choosing either the fast robot or slow "cloud" model. First, since we operate with classification DNNs, the loss is defined to be unity when a model prediction (y_{robot} or y_{cloud}) is different from the true class y_{true} and is zero when the model prediction is correct. Then, as per Eq. 4, we should only select the slow model if the accuracy gain is higher than the weighted additional inference cost:

$$a^*_{\text{offload}} = \begin{cases} 1, & \Big(\mathcal{L}(y_{\text{robot}}, y_{\text{true}}) - \mathcal{L}(y_{\text{cloud}}, y_{\text{true}}) \\ & \quad > \frac{\beta_{\text{cost}}}{\alpha_{\text{accuracy}}}(c_{\text{cloud}} - c_{\text{robot}})\Big) \\ 0, & \text{otherwise.} \end{cases} \tag{6}$$

In the above condition, c_{cloud} and c_{robot} are the inference times of the slow and fast DNNs respectively.

Given the desired action a^*_{offload}, we can assemble a training dataset $\mathcal{D}$ with N images indexed by i. The dataset has images x^i, both models' predictions, and the optimal action:

$$\mathcal{D} = \left\{x^i, \texttt{conf}^i_{\text{robot}}, y^i_{\text{robot}}, \texttt{conf}^i_{\text{cloud}}, y^i_{\text{cloud}}, a^{*,i}_{\text{offload}}\right\}_{i=0}^{N}. \tag{7}$$

Then, our supervised learning problem is to learn a policy π_{offload} that maps outputs of the fast "robot" model to an action $\hat{a}_{\text{offload}}$ indicating whether invoking the slow "cloud"

DNN is predicted to be necessary:

$$\hat{a}^i_{\text{offload}} = \pi_{\text{offload}}(\texttt{conf}^i_{\text{robot}}, y^i_{\text{robot}}). \quad (8)$$

In practice, we trained π_{offload} as a simple feed-forward neural network using the cross-entropy loss between $\hat{a}_{\text{offload}}$ and a^*_{offload}. The neural network was extremely compact and just < 20 KB in size.

6.3 Performance on the Edge TPU

In our hardware experiments, we used the Google Edge TPU development board, where the main TPU application specific integrated circuit (ASIC) runs the fast model of MobileNet v2.[4] Then, our small supervised learning offloader decides whether the EfficientNet-Large model should be run on a *complementary second* TPU attached via a USB port, as pictured in Fig. 8. Ideally, our offloader should only invoke the large model when it is highly uncertain, and otherwise enjoy the low-latency of the small model for fast decision-making.

Figure 9 illustrates the benefits of our learned offloader and are analogous to the gains observed in Fig. 6. Specifically, our learned offloader (blue) achieves virtually the same upper-bound classification accuracy as the large model (red) but with a much lower mean latency. Consistently using the large model ("All-large", red) is analogous to the all-cloud benchmark in our previous experiments. Further, our learned offloader has higher accuracy than a random policy (green) and simply using the small model at all times (black) with only a marginal increase in mean latency. Interestingly, our gains are more pronounced if we consider median latency, since our learned policy mostly utilizes the small model but selectively, and judiciously, queries the large model when it is uncertain.

Overall, our learning-based offloading approach shows strong performance for an increasingly important scenario of on-board vision model selection for compute-and-power constrained robots. Our results illustrate the generality of our approach to a significantly varied scenario from Sect. 5.2. Specifically, we used a different deep learning accelerator (the TPU), different vision DNNs (EfficientNets), and instead used a supervised-learning based offloader. As such, we believe our results to be a valuable first step for learning-based offloading policies in energy-constrained robotics. We believe a promising direction for future work is to integrate our TPU setup on a low-power robot and integrate offloading within the context of decision-making and planning.

[4] Source code for on-device distributed inference can be found at: https://bitbucket.org/sandeep_chinchali/edgetpu_dev_board_release/src/distributed_inference/.

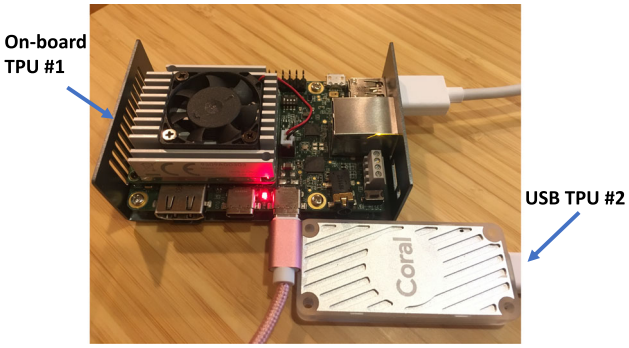


Fig. 8 Google Edge TPU hardware setup: the fast model and offloader run on the on-board hardware and can selectively run a slower, but more accurate, model on the USB TPU. This serves as a simple example for model selection on the same physical platform

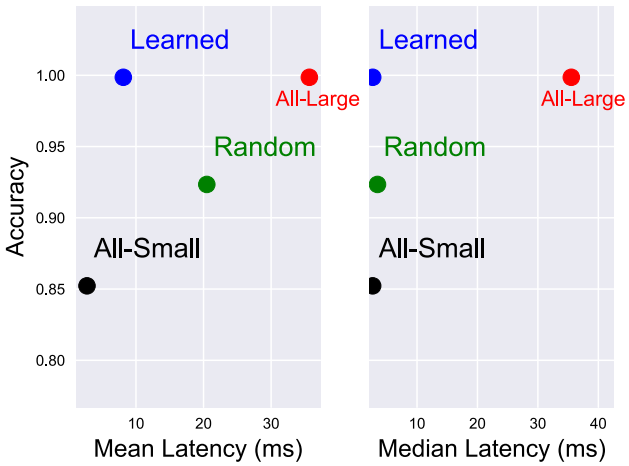


Fig. 9 Offloading on the Google Edge TPU with EfficientNets: our learning-based offloading framework also applies when we must select between a small MobileNet and large EfficientNet vision DNN. Specifically, our learning-based offloader (blue) achieves the accuracy of consistently invoking the large model with only a marginal gain in mean and median latency (x-axis). Further, it outperforms benchmarks of random sampling and simply executing the fast model at all times (Color figure online)

7 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 learning-based methods are an effective choice to optimize offloading policies even for variants of the offloading problem as stated. For example, we have demonstrated the generality of our approach beyond cloud robotics by using supervised learning to select between compute-efficient vision models on the same device.

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 (Altman, 1999), which has seen recent research activity within deep reinforcement learning (Achiam et al., 2017; Chow et al., 2018).

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 (Bajcsy, 1988), but also incorporates information about network conditions or input stream staleness.

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. The NASA University Leadership initiative (Grant #80NSSC20M0163) also provided funds to assist the authors with their research, but this article solely reflects the opinions and conclusions of its authors and not any NASA entity.

References

Abadi, M., Barham, P., Chen, J., Chen, Z., Davis, A., Dean, J., Devin, M., Ghemawat, S., Irving, G., & Isard, M., et al. (2016). Tensorflow: A system for large-scale machine learning. In *Proceedings of the OSDI*. Savannah.

Achiam, J., Held, D., Tamar, A., & Abbeel, P. (2017). Constrained policy optimization. In *International conference on machine learning* (pp. 22–31).

Altman, E. (1999). *Constrained Markov decision processes*. CRC Press.

Amos, B., Ludwiczuk, B., & Satyanarayanan, M.. (2016). *Openface: A general-purpose face recognition library with mobile applications*. Technical report, CMU-CS-16-118, CMU School of Computer Science.

Bajcsy, R. (1988). Active perception. In *Proceedings of the IEEE*.

Bellman, R. (1957). *A Markovian decision process*. DTIC Document: Technical report.

Blundell, C., Cornebise, J., Kavukcuoglu, K., & Wierstra, D. (2015). Weight uncertainty in neural networks. *International conference on machine learning (ICML)*.

Bozcuoğlu, A. K., Kazhoyan, G., Furuta, Y., Stelter, S., Beetz, M., Okada, K., et al. (2018). The exchange of knowledge using cloud robotics. *IEEE Robotics and Automation Letters*, *3*(2), 1072–1079.

Brockman, G., Cheung, V., Pettersson, L., Schneider, J., Schulman, J., Tang, J., & Zaremba, W. (2016). *Openai gym*. arXiv:1606.01540.

Camacho, E. F., & Alba, C. B. (2013). *Model predictive control*. Springer.

Chinchali, S., Pergament, E., Nakanoya, M., Cidon, E., Zhang, E., Bharadia, D., et al. (2020). *International symposium on experimental robotics (ISER), Malta, Valetta*.

Chinchali, S., Sharma, A., Harrison, J., Elhafsi, A., Kang, D., Pergament, E., et al. (2019). Network offloading policies for cloud robotics: A learning-based approach. In *Robotics: Science and systems, Freiburg im Breisgau, Germany*.

Chinchali, S. P., Cidon, E., Pergament, E., Chu, T., & Katti, S. (2018). Neural networks meet physical networks: Distributed inference between edge devices and the cloud. In *ACM workshop on hot topics in networks (HotNets)*.

Chow, Y., Nachum, O., Duenez-Guzman, E., & Ghavamzadeh, M. (2018). A lyapunov-based approach to safe reinforcement learning. In *Neural information processing systems (NIPS)*.

Edge TUP. (2019). Retrieved September 1, 2019, from https://cloud.google.com/edge-TUP/.

Fetch Robotics. (2019). *Introducing the fetch cloud robotics platform*. Retrieved May 14, 2019, from https://fetchrobotics.com/products-technology/cloud-robotics-platform-for-warehouse-automation/.

Forouzan, B. A., & Fegan, S. C. (2002). *TCP/IP protocol suite*. McGraw-Hill.

Gal, Y., Islam, R., & Ghahramani, Z. (2017). *Deep bayesian active learning with image data*.

Goldberg, K., & Kehoe, B.. (2013). *Cloud robotics and automation: A survey of related work*. EECS Department, University of California, Berkeley, Tech. Rep. UCB/EECS-2013-5.

Harrison, J., Sharma, A., & Pavone, M. (2018). Meta-learning priors for efficient online Bayesian regression. In *Workshop on the algorithmic foundations of robotics (WAFR)*.

Higuera, J. C. G., Xu, A., Shkurti, F., & Dudek, G. (2012). Socially-driven collective path planning for robot missions. In *IEEE conference on computer and robot vision*.

Hochreiter, S., & Schmidhuber, J. (1997). Long short-term memory. *Neural Computation*, *9*(8), 1735–1780.

Intel. (2016). *Data is the new oil in the future of automated driving*. Retrieved January 30, 2019, from https://newsroom.intel.com/editorials/krzanich-the-future-of-automated-driving/#gs.LoDUaZ4b.

Jain, A., Das, D., Gupta, J. K., & Saxena, A. (2015). Planit: A crowdsourcing approach for learning to plan paths from large scale preference feedback. In *IEEE international conference on robotics and automation (ICRA)*.

Kalva, H. (2006). The h. 264 video coding standard. *IEEE Multimedia*, *13*(4), 86–90.

Kang, D., Emmons, J., Abuzaid, F., Bailis, P., & Zaharia, M. (2017a). Noscope: Optimizing neural network queries over video at scale. In *Proceedings of the VLDB Endow*.

Kang, Y., Hauswald, J., Gao, C., Rovinski, A., Mudge, T., Mars, J., et al. (2017b). Neurosurgeon: Collaborative intelligence between the cloud and mobile edge. *ACM SIGPLAN Notices*, *52*(4), 615–629.

Kehoe, B., Matsukawa, A., Candido, S., Kuffner, J., & Goldberg, K. (2013). Cloud-based robot grasping with the google object recognition engine. In *2013 IEEE international conference on robotics and automation (ICRA)* (pp. 4263–4270). IEEE.

Kehoe, B., Patil, S., Abbeel, P., & Goldberg, K. (2015). A survey of research on cloud robotics and automation. *IEEE Transactions on Automation Science and Engineering*, *12*(2), 398–409.

Kuffner, J. (2010). Cloud-enabled robots. In *IEEE-RAS international conference on humanoid robots*. IEEE.
Lin, T.-Y., Maire, M., Belongie, S., Hays, J., Perona, P., Ramanan, D., Dollár, P., & Zitnick, C. L. (2014). Microsoft coco: Common objects in context. In *European conference on computer vision* (pp. 740–755). Springer.
Mao, H., Netravali, R., & Alizadeh, M. (2017). Neural adaptive video streaming with pensieve. In *Proceedings of the conference of the ACM special interest group on data communication* (pp. 197–210). ACM.
Marsh, S. (2018). *Amazon alexa crashes after christmas day overload*. Retrieved January 20, 2019, from https://www.theguardian.com/technology/2018/dec/26/amazon-alexa-echo-crashes-christmas-day-overload.
Marshall, A. (2021). *Starsky robotics unleashes its truly driverless truck in florida*. Wired Magazine. https://www.wired.com/story/starsky-robotics-truck-self-driving-florida-test.
Mitchell, B. (2018). *Learn exactly how fast a wi-fi network can move*. Retrieved January 31, 2019, https://www.lifewire.com/how-fast-is-a-wifi-network-816543.
Mnih, V., Badia, A. P., Mirza, M., Graves, A., Lillicrap, T., Harley, T., Silver, D., & Kavukcuoglu, K. (2016). Asynchronous methods for deep reinforcement learning. In *International conference on machine learning* (pp. 1928–1937).
Mnih, V., Kavukcuoglu, K., Silver, D., Graves, A., Antonoglou, I., Wierstra, D., & Riedmiller, M.. (2013). *Playing atari with deep reinforcement learning*. *arXiv preprint* arXiv:1312.5602
Mohanarajah, G., Hunziker, D., D'Andrea, R., & Waibel, M. (2015a). Rapyuta: A cloud robotics platform. *IEEE Transactions on Automation Science and Engineering*, *12*(2), 481–493.
Mohanarajah, G., Usenko, V., Singh, M., D'Andrea, R., & Waibel, M. (2015b). Cloud-based collaborative 3d mapping in real-time with low-cost robots. *IEEE Transactions on Automation Science and Engineering*, *6*, 66.
Padhye, J., Firoiu, V., & Towsley, D. (1999). *A stochastic model of tcp reno congestion avoidence and control*.
Pakha, C., Chowdhery, A., & Jiang, J. (2018). *Reinventing video streaming for distributed vision analytics*. In *10th USENIX workshop on hot topics in cloud computing (HotCloud 18)*. USENIX Association. https://www.usenix.org/conference/hotcloud18/presentation/pakha.
Penmetcha, M., & Min, B.-C. (2021). A deep reinforcement learning-based dynamic computational offloading method for cloud robotics. In *IEEE Access*.
Quigley, M., Conley, K., Gerkey, B., Faust, J., Foote, T., Leibs, J., Wheeler, R., & Ng, A. Y. (2009). Ros: An open-source robot operating system. In *ICRA workshop on open source software, Kobe, Japan* (Vol. 3, p. 5).
Rahman, A., Jin, J., Cricenti, A., Rahman, A., & Yuan, D. (2016). A cloud robotics framework of optimal task offloading for smart city applications. In *IEEE global communications conference (GLOBECOM)*.
Riazuelo, L., Civera, J., & Montiel, J. M. M. (2014). C2tam: A cloud framework for cooperative tracking and mapping. *Robotics and Autonomous Systems*, *62*(4), 401–413.
Riiser, H., Vigmostad, P., Griwodz, C., & Halvorsen, P. (2013). Commute path bandwidth traces from 3g networks: Analysis and applications. In *Proceedings of the 4th ACM multimedia systems conference (MMSys'13)* (pp. 114–118). ACM. ISBN 978-1-4503-1894-5.
ROS Answers. (2017). *Ros ate my network bandwidth!* Retrieved January 31, 2019, from https://answers.ros.org/question/256080/ros-ate-my-network-bandwidth/.
Salmerón-Garcı, J., Íñigo-Blasco, P., Dı, F., Cagigas-Muniz, D., et al. (2015). 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.
Sandler, M., & Howard, A. (2018). *Mobilenetv2: The next generation of on-device computer vision networks*. https://ai.googleblog.com/2018/04/mobilenetv2-next-generation-of-on.html.
Schroff, F., Kalenichenko, D., & Philbin, J. (2015). Facenet: A unified embedding for face recognition and clustering. In *Proceedings of the IEEE conference on computer vision and pattern recognition* (pp. 815–823).
Sugiura, K., & Zettsu, K. (2015). Rospeex: A cloud robotics platform for human-robot spoken dialogues. In *IEEE international conference on intelligent robots and systems (IROS)*.
Sutton, R. S., & Barto, A. G. (1998). Reinforcement learning: An introduction. *IEEE Transactions on Neural Networks*, *9*(5), 1054.
Szepesvári, C. (2010). Algorithms for reinforcement learning. *Synthesis Lectures on Artificial Intelligence and Machine Learning*, *4*(1), 1–103.
Tan, M., & Le, Q. V. (2019). *Efficientnet: Rethinking model scaling for convolutional neural networks*. arXiv preprint arXiv:1905.11946
Tanwani, A. K., Anand, R., Gonzalez, J. E., & Goldberg, K. (2020). Rilaas: Robot inference and learning as a service. *IEEE Robotics and Automation Letters*, *5*(3), 4423–4430.
Tanwani, A. K., Mor, N., Kubiatowicz, J., Gonzalez, J. E., & Goldberg, K. (2019). *A fog robotics approach to deep robot learning: Application to object recognition and grasp planning in surface decluttering*. arXiv preprint arXiv:1903.09589
Tian, N., Chen, J., Ma, M., Zhang, R., Huang, B., Goldberg, K., & Sojoudi, S. (2018). *A fog robotic system for dynamic visual servoing*. *arXiv preprint* arXiv:1809.06716
Verizon. (2013). *4g lte speeds vs. your home network*. Retrieved January 31, 2019, https://www.verizonwireless.com/articles/4g-lte-speeds-vs-your-home-network/.
Verma, L., Fakharzadeh, M., & Choi, S. (2013). *Wifi on steroids: 802.11 ac and 802.11 ad*. *IEEE Wireless Communications, 20*(6), 30–35.
Wan, J., Tang, S., Yan, H., Li, D., Wang, S., & Vasilakos, A. V. (2016). Cloud robotics: Current status and open issues. *IEEE Access*, *4*, 2797–2807.
Wu, H., Lou, L., Chen, C.-C., Hirche, S., & Kuhnlenz, K. (2013). Cloud-based networked visual servo control. In *IEEE transactions on industrial electronics*.
Xiangyu, Z., Xinyu, Z., Mengxiao, L., & Jian, S. (2017). Shufflenet: An extremely efficient convolutional neural network for mobile devices. In *Computer vision and pattern recognition*.
Youtube. (2021). *Youtube: Recommended upload encoding settings*. https://support.google.com/youtube/answer/1722171?hl=en.

Sandeep Chinchali is an assistant professor (starting Fall 2021) at the University of Texas at Austin ECE department and a core member of UT Austin's Robotics consortium. He completed his Computer Science Ph.D. at Stanford University co-advised by Prof. Sachin Katti and Prof. Marco Pavone. His research interests are in cloud robotics and compute-efficient deep learning.

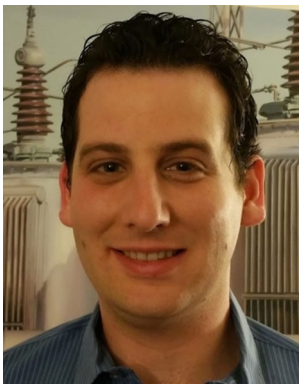
Apoorva Sharma is a graduate student in the Stanford University Aeronautics and Astronautics department. Prior to studying at Stanford, he received a B.S. in Engineering at Harvey Mudd College in 2016. Apoorva's research interests are in the intersection of machine learning, control theory, and planning.

James Harrison is a graduate student in the department of Mechanical Engineering at Stanford University. He received a B.Eng. in Mechanical Engineering from McGill University in 2015, and an M.S. in Mechanical Engineering from Stanford University in 2017. James' research interests include control theory, robotics, and machine learning.

Amine Elhafsi is a Ph.D. student in the Department of Aeronautics and Astronautics at Stanford University. He obtained his M.S. degree from Stanford in 2019. Prior to joining Stanford, Amine graduated summa cum laude with a B.S. in aerospace engineering from UCLA. Amine's research interests include motion planning, optimal control, machine learning, and robotics.

Daniel Kang is a Ph.D. student in Computer Science at Stanford University. Daniel did his bachelors and M.Eng. at MIT, with his thesis in computational biology. His research focuses on deploying (unreliable) machine learning models efficiently and with guarantees.

Evgenya Pergament is a Ph.D. student in the EE department at Stanford University, advised by Prof. Sachin Katti. Her research interests are in edge computing, machine learning, and computer systems. Prior to studying at Stanford, she received a M.S. and B.S. in Computer Science from Technion—Israel Institute of Technology.



Eyal Cidon obtained his Ph.D. from Stanford University, advised by Sachin Katti. His research interests are in cloud and edge computing, machine learning, and computer systems. Eyal graduated with a B.Sc. degree in Computer Engineering from Technion—Israel Institute of Technology.

Sachin Katti is currently an Associate Professor of Electrical Engineering and Computer Science at Stanford University. He received his Ph.D. in EECS from MIT in 2009. His research focuses on designing and building next generation high capacity wireless networks using techniques from information and coding theory. His dissertation won the 2008 ACM Doctoral Dissertation Award—Honorable Mention and the George Sprowls Award for Best Doctoral Dissertation in EECS at MIT.

Dr. Marco Pavone is an Associate Professor of Aeronautics and Astronautics at Stanford University, where he is the Director of the Autonomous Systems Laboratory and Co-Director of the Center for Automotive Research at Stanford. Before joining Stanford, he was a Research Technologist within the Robotics Section at the NASA Jet Propulsion Laboratory. He received a Ph.D. degree in Aeronautics and Astronautics from the Massachusetts Institute of Technology in 2010. His main research interests are in the development of methodologies for the analysis, design, and control of autonomous systems, with an emphasis on self-driving cars, autonomous aerospace vehicles, and future mobility systems.