Policies for Network Offloading with Deadlines

Ahmed Ahmed [1] **Daniel Guillen** [1]

Abstract

Robotic systems are turning toward using computationally-intensive resources like Deep Neural Networks (DNN) for different perception, decision making, and localization tasks. Additionally, more projects have also started to explore cloud computation for "offloading" DNN queries, which is interesting because they can provide improved accuracy and increased compute resources. We aim to extend the problem of how resource-constrained systems can optimally choose to offload a task to a cloud-server model over the local model by incorporating a "deadline" that requires a response within a certain amount of time. We also encode an instance of the problem where an image-processing system chooses which model to query, while minimizing latency, accounting for the deadline, and maximizing prediction accuracy of the given task.

1. Introduction

A large number of today's robotic or resource constrained systems are turning toward using computationally-intensive resources like Deep Neural Networks (DNN) for a variety of perception, decision making, and localization tasks. Additionally, more projects have also started to explore the idea of using edge computation or cloud computation (as opposed to machine-local computation) for DNN queries, where these tasks can be "offloaded" to another server and eventually the output is returned to the system for usage. Our project aims to explore and build upon the problem of how resource constrained systems can optimally choose to offload a given ML task to a cloud server-specific DNN model over choosing to evaluate the given ML task on the local, robot-specific DNN model, taking into account (1) the difference in accuracy between the cloud model and the local model, (2) the difference in long-term and short-term cost (bandwidth and latency related costs) between querying a local vs. cloud model, and (3) the uncertainty over the current task. We will also explore a new direction by incorporating (4) a "deadline" requirement that requires a definite response from the system in a certain amount of time. The deadline constraint creates a dilemma where the system must take into account that high-latency queries (like a cloud model query) are more risky the closer the system is to its deadline (as the system may not receive a response before the deadline is reached) whereas if the system has a lot more time before the deadline, it is much safer to query high-latency models. In order to experimentally evaluate the problem, we also encode a specific instance of this problem where a system has to process images as they arrive and choose which model to query, while trying to minimize latency cost, meeting the deadline requirement, and maximize accuracy of the model prediction over the perception task (face classification, in our case). All the environment, training, and evaluation code we implemented can be found here (https://github.com/eliasd/CS234_Final_Project).

1.1. Relevant Literature

Our project largely builds off of the work that our mentor, Sandeep Chinchali, has done on the "Robot Offloading Problem" in his 2019 paper "Network Offloading Policies for Cloud Robotics: a Learning-based Approach" [1]. While his work largely formally defines the problem of offloading DNN tasks between local and cloud models and provides a lot of the context on the topic, we want our project to modify some of the key problem components that the original paper presented. In conversation with our mentor, we identified one key element that we wanted to contribute to the main topic, namely the idea of a "deadline" where a system has a limited amount of time before some response is required and the system has to account for this deadline when it chooses between querying a local or external DNN model.

When considering other approaches to the mobile network offloading problem, the need and novelty of reinforcement learning becomes clear. Related works in this area have suggested a fixed interval during which 'key-frames' are to be uploaded to the cloud to limit network usage [11]. However, this approach ignores the fact that selecting a good

*Equal contribution [1]Department of Computer Science, Stanford University. Correspondence to: Ahmed Ahmed <ahmedah@stanford.edu>, Daniel Guillen <eliasd@stanford.edu>.

Proceedings of the 35^{th} International Conference on Machine Learning, Stockholm, Sweden, PMLR 80, 2018.

interval is in and of itself a difficult task, and that a single fixed interval is not likely to be able to capture all the important frames. Other approaches have attempted heuristic policies [12][13], where the authors used hand-engineered features to best determine when a frame is "important" for accurate prediction. This is also problematic as it relies on domain-specific solutions, which will not generalize well from say, bounding box prediction to image classification. This approach is also most likely sub-optimal, as the state-space for any non-trivial robotic off-loading problems is high-dimensional, and any heuristic function would face difficulties meaningfully weighting all information to decide when to offload to a local or cloud model. The need for a domain-agnostic approach that can generalize well across complex decision making problem formulations means that reinforcement learning is both an appropriate and promising method for network offloading.

2. Approach

2.1. Markov Decision Process Formulation

We define the generic offloading problem to be a finite-horizon Markov Decision Process (MDP):

$$\mathcal{M}_{offload} = (\mathcal{S}, \mathcal{A}, \mathcal{R}, \mathcal{H}, \mathcal{D})$$

Where $\mathcal{S}$ is our state space, $\mathcal{A}$ is our action space, $\mathcal{R}$ is our reward function and $\mathcal{H}$ is the horizon, and $\mathcal{D}$ is the given deadline — that is, the amount of time left before the system or agent must make a decision.

The action space is defined by which prediction to select or query. For a given input timestep, an agent can either choose to use any of the past predictions or query any of the available models. In this case, an agent would be able to query the local model or a cloud model (with each model query having an associated computation and network cost) or save on computation cost and network latency costs by using any previous predictions from the local, edge, or cloud models (but at the potential cost of inaccuracy).

The state space is defined by:

$$\begin{aligned} s_t \;=\; [\; & \phi(x_t),\;\; \hat{f}_{\text{local}},\;\; \hat{f}_{\text{cloud}}, \\ & \Delta t_{\text{local}},\;\; \Delta t_{\text{cloud}}, \\ & \Delta N_{\text{budget}}, \mathcal{H} - t, D\;] \end{aligned}$$

where x_t is an input at the given timestep, $\phi(x_t)$ is the feature vector for our input, $\hat{f}_{\text{local}}$ and $\hat{f}_{\text{cloud}}$ are the past local and cloud predictions, respectively, Δt_{local} and Δt_{cloud} are the count of the timesteps that have passed since the last local and cloud query, respectively, ΔN_{budget} is our remaining "budget" for cloud queries, D is the amount of time (at this timestep) that has elapsed toward the deadline (ranging from 0 to the deadline $\mathcal{D}$), and $\mathcal{H} - t$ is the time remaining. The inclusion of the "budget" term ΔN_{budget} for the cloud model queries is intended to abstractly represent the network bandwidth (i.e. the amount of problem-specific data that can be offloaded over a network) that is available over the episode length — this value can be deterministically or stochastically set at the initial timestep according to some analysis of the available network.

The dynamics are as follows. If we choose to use a past prediction of any form (i.e. $\hat{f}_{\text{local}}$ or $\hat{f}_{\text{cloud}}$), then these past predictions don't change but we increment the timesteps that have passed since they were queried (Δt_{local} and Δt_{cloud}) by 1. If we choose to query the local model, then we update the respective prediction ($\hat{f}_{\text{local}}$) and reset the time since it was last queried (Δt_{local}) to zero. If we use the cloud model, then we will update our cloud prediction ($\hat{f}_{\text{cloud}}$), reset the time since we last queried the cloud model (Δt_{cloud}) to zero, and decrement our budget of cloud queries ΔN_{budget} by one. The dynamics of the input x_t are independent of the offloading actions and are problem specific, depending on the domain.

For our deadline dynamics, the time elapsed toward the deadline, D, is sampled at each time-step from a probability distribution — this is intended to model the stochasticity in system computations and network response which can vary between different timesteps. Any latency/computation distribution model can be used but for our problem instance we chose a normal distribution, as we felt that would most realistically model the latency that occurs because of external or internal factors.

We formulated our reward function to take into account our competing objectives of maximizing overall prediction accuracy, minimizing costs associated with network latency and compute time, and minimizing deadline-related errors (e.g. choosing high-latency queries when the system is close to the deadline). Therefore we define our reward/cost function to be a linear combination of the error of our prediction, ($\mathcal{L}$), the costs of the given action, and a deadline cost term:

$$R(s, a) = -\alpha \cdot \mathcal{L}(\hat{f}_t, y_t) - \beta \cdot \text{Cost}_q(a_t) - \gamma \cdot D \cdot \text{Cost}_d(a_t)$$

Where α, β, and γ are parameters for the weights assigned for the relative importance between accuracy, query cost, and the deadline, respectively (the higher the deadline time is, the closer we are to the deadline, the less time we have to query the cloud model so the higher the penalty).

With this MDP formulation in mind, we establish the objective of the offloading problem to find the optimal policy π^* that minimizes the total expected cost (maximizes the expected reward) over the episode:

$$\underset{\pi^*}{\text{argmax}}\; \underset{\pi^*}{\mathbb{E}} \left[\sum_{t=0}^{\mathcal{H}} R(s_t, \pi^*(s_t)) \right]$$

2.2. Policy Gradient Methods

Because of the inherent stochasticity of our problem formulation, a deterministic policy would not be sufficient for maximizing reward. Therefore, we elected to use policy gradient methods as they can be used to learn stochastic policies. One of the baseline approaches that we used to compare our main algorithm to was the Vanilla Policy Gradient (VPG) method. However, instead of using the discounted sum of returns from a roll-out G_t, we utilized an Advantage Actor-Critic method instead, where the gradient is:

$$\nabla_\theta J(\theta) = \mathbb{E}_{\pi\theta}[\nabla_\theta \log \pi_\theta(s,a) A^{\pi_\theta}(s,a)]$$

The three other baseline algorithms that we used to compare our main algorithm to included (1) a random policy where an action is randomly selected at each step, (2) a local-model-only policy where the local model is queried at each step, and (3) a cloud-model-only policy where the cloud model is queried at each step.

We use an Advantage Actor-Critic approach with Proximal Policy Optimization (PPO) as our main algorithm to learn our policy. This algorithm also attempts to maximize the size of improvement without policy collapse, but unlike methods like Trust Region Policy Optimization it does not require the use of second-order methods, or a hard KL-divergence constraint which has lead to better sample complexity and computationally efficiency empirically [3]. We used a variant of PPO known as PPO-clip, which updates policies as follows:

$$\theta_{k+1} = \underset{\theta}{\operatorname{argmax}}\, \mathbb{E}_{s,a\sim\pi_{\theta_k}}[L(s,a,\theta_k,\theta)]$$

Where the objective function $L(s,a,\theta_k,\theta)$ is defined as follows:

$$\min\left(\frac{\pi_\theta(a \mid s)}{\pi_{\theta_k}(a \mid s)} \cdot A^{\pi_\theta}(s,a),\ g(\epsilon, A^{\pi_\theta}(s,a))\right)$$

where:

$$g(\epsilon, A^{\pi_\theta}(s,a)) = \begin{cases} (1+\epsilon)A & \text{if } A \geq 0 \\ (1-\epsilon)A & \text{if } A < 0 \end{cases}$$

Which effectively limits how much advantages can lead to our new policy deviating from our original policy.

3. Experimental Performance

3.1. Problem Instance: Face Detection

In this section, we formulate an instance of the offloading problem that is intended to be a representative problem setting for resource-constrained systems. In this case, the problem instance is that of a system that is being exposed to a video stream of human faces as sensory input and is expected to accurately classify the person as they appear in each frame. To model a video stream, we simulate a time-coherent stream of image frames where each x_t is a given frame from the stream. The simulated stream is stochastically generated at the beginning of each episode and contains an image x_t for each timestep and is divided into different sub-intervals where only different images of the same person are shown before switching to another sub-interval with another set of images of another person. This is intended to model the time-coherence of a video stream where a single individual will appear over a sequence of frames in slightly different configurations (depending on their movement) before leaving and having another person enter the video frame. This simulated stream model is derived from our mentor's original paper (see "Synthetic Input Stream Experiments" from [1]).

We choose the feature encoding ϕ that is used in the state space to be the 128-dimension FaceNet embeddings of the given input x_t, which can be found in the FaceNet dataset. For the local model, we use the SVM classifications of the FaceNet embeddings [10] and for the cloud model, we use the groundtruth as the effective predictions of the cloud model (which is intended to have a very high accuracy).

We used a zero-one loss function as the error term for a model's prediction where $\mathcal{L}(\hat{f}_t, y_t) = 1$ if the prediction was inaccurate and $\mathcal{L}(\hat{f}_t, y_t) = 0$ if the prediction was accurate. For the model query costs, we set the cost for using past local or cloud model predictions to zero (as using past predictions requires no significant compute-cost or latency related cost), the cost for querying the local model to 1.0, and the cost for querying the cloud model to 5.0 as querying the cloud model is the most time-consuming query due to compute time and network latency. Following from our mentor's paper, we used $\alpha = 1.0$ as the accuracy weighting and $\beta = 10.0$ as the query cost weighting.

3.2. Deep Reinforcement Learning

Table 1

Mean episodic return	Tahn	Relu
32 hidden units	-430.706	-413.147
64 hidden units, two-layers	-399.697	-385.769

We approach the offloading problem using deep reinforcement learning for a variety of reasons. Firstly, our MDP problem formulation involves a large state space (that can vary depending on the input feature vector $\phi(x_t)$ used) and includes varying levels of complexity surrounding the problem dynamics (i.e. the dynamics around the input x_t along with the dynamics around the rest of the problem). With a deep RL approach, we can generalize very well across

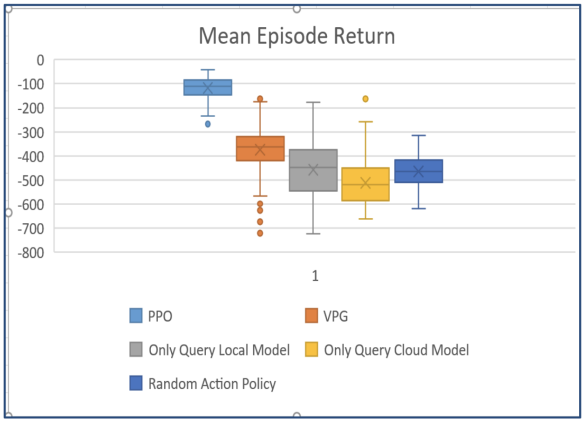

Figure 1. PPO policy had the best mean episode return.

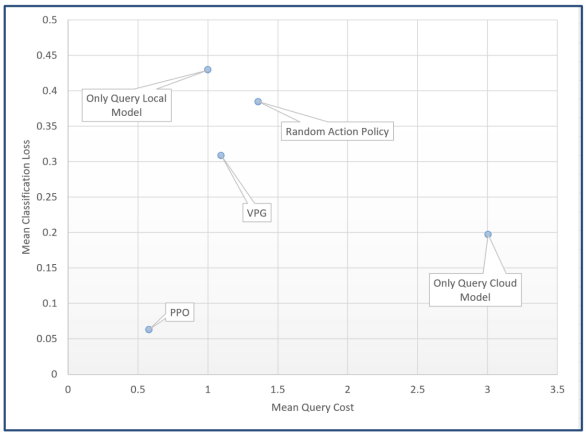

Figure 2. Trade off between mean classification loss and mean query cost.

the state space and avoid having to model and estimate the problem dynamics (due to the model-free nature of deep RL). Moreover, because we incorporate a feature vector into our state space, we already provide a good basis for a deep RL agent to learn all that it needs to about the state space complexity.

As mentioned in our approach, we used an actor-critic method for both our VPG and PPO algorithms with a neural network for both the policy and the estimation of advantages. After discussing with our mentor, and consulting a research talk on neural architecture search [6], we decided that an experimental approach would be the best way to determine our specific architecture. We decided to test both the tahn and relu activation functions. We decided not to use the sigmoid activation function because it's outputs are not zero-centered, which empirically leads to unstable gradient updates during backpropagation [7]. Given the size of the FaceNet dataset [10], we decided to use two variations on our choice of hidden units: one hidden layer with 32 neurons, or two hidden layers with 64 neurons each. The output layer for all networks is a softmax activation so we can generate a stochastic policy. Additionally, we include the action constraint of not allowing the cloud model query when the cloud budget has run out by mapping the cloud model query action to the local model query action when ∇N_{budget}, an action constraint derived from our mentor's original paper [1].

After creating our MDP as a OpenAI gym environment [7], we used spinning up to easily run multiple experiments with the variations on our neural architecture as mentioned before, averaged over 5 seeds for 10 epochs which we listed in Table 1. We chose the number of seeds & epochs to strike a balance between ensuring we mitigate the variance of deep reinforcement learning which often leads to fairly different results depending on the seed [9] and making sure we judiciously used our time and compute resources. After evaluating on VPG, we saw that the architecture with the highest mean episodic return at the end of tests was two hidden layers with 64 units each using the tahn activation.

3.3. Results & Analysis

Figure 1 highlights the mean episode return (that is, the mean episode cost) that each policy had while Figure 2 breaks down the mean query cost and mean classification loss that each policy had, where the origin represents the optimal (and un-achievable) point with zero query classification loss. Figure 1 shows how well PPO performed compared to the baselines, achieving a mean episode return of -117.24, much higher than any of the other policy performances. Even the VPG policy was only able to achieve a mean episode return of -373.66 (the second best mean episode return), about 3.2 times higher in magnitude than the PPO return. We attribute this significant increase in performance to the clipping regularization measures that PPO incorporates into its objective function that prevent large changes in the policy that can lead to performance collapses during training. The random policy, local-model-only policy, and cloud-model-only policy baselines achieved very similar performance returns with each other with the cloud-model-only policy having the worst performance out of all the policies — they achieved an episode return of -464.60, -457.77, and -511.45, respectively.

Looking at action distributions, we see that the PPO policy was very conservative with its cloud model queries only querying the cloud model about 7% of the time while querying the local model 25% of the time. The learned PPO policy also effectively learns to depend on its past predictions, using the past local predictions 34% of the time and the past cloud predictions 34% of the time. Both of these action distributions indicates how the PPO policy is able to handle the general accuracy vs. query cost tradeoff along

with the additional deadline cost — the PPO policy is able to recognize when it has a low confidence on a given input and then optimally decide between the local model and cloud model for a new accurate prediction, only querying the cloud model when absolutely necessary i.e. when confidence in local model is especially small. From then on, it depends on its past predictions (which is why those actions have such high percentages) for the given sub-intervals until its confidence goes down again (due to the introduction of a new person) at which point it then goes back to querying the models for a new prediction. In this way, it is able to both (1) minimize the deadline cost, only querying the cloud model when it has a small associated deadline cost i.e. when there is enough time before the deadline to take on a network latency cost and (2) improve the overall system accuracy while minimizing the query cost associated with both the local and cloud model queries.

Compared to the PPO action distribution, the VPG action distribution is a little bit different, querying the cloud model and local model, 20% and 33% of the time, respectively, while using the past cloud model prediction and past local model prediction 39% and 8% of the time, respectively. Compared to the PPO policy, the VPG policy is less selective with its cloud model queries which drives up its total query cost but it also performs worse classification wise (Figure 2), potentially due to an over-dependence on its past cloud predictions and an under-dependence on its past local predictions, a feature that the PPO policy was able to learn.

Looking at Figure 2, we can break down the performance of each policy across both the mean classification loss and mean query cost axis. The PPO policy performs the best across each axis, maintaining both the minimum classification loss and query cost. Unsurprisingly, the cloud-model-only policy has the largest query cost as it chooses the most expensive query cost option available (the cloud model query) which drives down its mean episode return (which is why it has the lowest episode return) but it also has the second-best prediction accuracy, even performing better than the VPG policy (classification loss of about 0.2 for cloud-model-only policy compared to about 0.3 for VPG policy). Interestingly, the PPO policy is able to have a better accuracy than the cloud-model-only policy due to two key reasons: (1) the bandwidth budget constraint on the cloud model query action limits the true amount of accurate predictions and (2) the PPO policy conservatively uses its cloud model queries, reserving the queries for when accuracy is needed the most, driving down the overall classification loss. The local-model-only policy had the worst mean classification loss (which can be attributed to the inherent limits of the SVM classification of the local model) but it did have the second-lowest mean query cost behind the PPO policy.

4. Conclusion

In this paper, we have introduced an extension to the offloading problem by introducing a new formulation that includes the addition of a deadline, a timing constraint intended to represent the amount of time that a resource-constrained system has before a response (or classification prediction, in our problem instance) is required by the system. We also formulated and implemented a simulated problem instance of the offloading problem where a system has to attempt to accurately classify the individual face presented to it in various images (across a time-coherent synthesized sequence of image frames with different individual faces) while trying to minimize the overall query cost and overall deadline cost.

We approach the problem instance with two different deep RL policy gradient methods, Proximal Policy Optimization (PPO) and Vanilla Policy Gradient (VPG) both with Advantage Actor-Critic, finding that PPO not only performs the best overall, with the highest mean episode return, but by a large margin over VPG and the other baselines tested. We find that the learned PPO policy works so well due to its selective usage of the query-expensive and deadline-expensive cloud model queries and intelligent dependence on past predictions — the PPO policy learned when to query for a new prediction from the local model, only using the cloud model when confidence in the accuracy of the local model is very low, and then relying on its previous predictions for the next sequence of images until a new person is introduced and a new prediction is needed. Our results indicate that deep RL policy gradient methods can be used very effectively under this problem environment, finding effective offloading policies that maximize episode return (driving down total cost).

4.1. Future Directions

While our project is tackles a substantial modification from the original RL network offloading paper [1], we simplified our code in that we did not utilize 'stages', which more finely defined cases from the four actions we defined. Adding them in would make our simulation closer to reality.

Another substantial modification we could make to this problem is adding more actions. We considered adding an 'edge' model that would be an intermediate choice between the local and cloud model in terms of cost and accuracy. We decided against this because of the time for training policies on Spinning Up, but this would be a fairly simply extension to implement relative to our current work.

Ultimately, the most ambitious and exciting extension would be to train and implement our algorithm on a physical drone that actively queries a real-time cloud server. This would be the best proof that our representation of deadlines is an accurate model of delay in real-world networking.

4.2. Acknowledgements

We would like to acknowledge our project mentor, Sandeep Chinchali for his contributions to our project. Sandeep give us guidance on how we could extend his paper, and was often there for us when we had questions about how we should re-implement his paper. Sandeep also guided us through his implementations for his project (available here) which helped significantly speed up our project. The custom gym environment we created for our project can be found here (https://github.com/eliasd/CS234_Final_Project).

5. References

[1] "Network Offloading Policies for Cloud Robotics: a Learning-based Approach" https://arxiv.org/abs/1902.05703

[2] "Proximal Policy Optimization Algorithms" https://arxiv.org/abs/1707.06347

[3] Spinning-Up Documentation https://spinningup.openai.com/en/latest/

[4] The Pursuit of (Robotic) Happiness: How TRPO and PPO Stabilize Policy Gradient Methods https://towardsdatascience.com/the-pursuit-of-robotic-happiness-how-trpo-and-ppo-stabilize-policy-gradient-methods-545784094e3b

[5] Trust Region Policy Optimization http://proceedings.mlr.press/v37/schulman15.pdf

[6] Advanced Machine Learning Day 3: Neural Architecture Search https://www.microsoft.com/en-us/research/video/advanced-machine-learning-day-3-neural-architecture-search/

[7] CS 231N Website: Commonly used activation functions http://cs231n.github.io/neural-networks-1/#actfun

[8] OpenAI Gym http://gym.openai.com/docs/

[9] How Many Random Seeds? Statistical Power Analysis in Deep Reinforcement Learning Experiments https://arxiv.org/abs/1806.08295

[10] 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.

[11] 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.

[12] Javier Salmeron-Garcı, Pablo Inigo-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.

[13] 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.