IEEE Robotics and Automation Letters (RAL) paper presented at the
2020 IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS)
October 25-29, 2020, Las Vegas, NV, USA (Virtual)

RILaaS: Robot Inference and Learning as a Service

Ajay Kumar Tanwani, Raghav Anand, Joseph E. Gonzalez, Ken Goldberg

***Abstract*— Programming robots is complicated due to the lack of 'plug-and-play' modules for skill acquisition. Virtualizing deployment of deep learning models can facilitate large-scale use/re-use of off-the-shelf functional behaviors. Deploying deep learning models on robots entails real-time, accurate and reliable inference service under varying query load. This paper introduces a novel Robot-Inference-and-Learning-as-a-Service (RILaaS) platform for low-latency and secure inference serving of deep models on robots. Unique features of RILaaS include: 1) low-latency and reliable serving with gRPC under dynamic loads by distributing queries over multiple servers on Edge and Cloud, 2) SSH based authentication coupled with SSL/TLS based encryption for security and privacy of the data, and 3) front-end REST API for sharing, monitoring and visualizing performance metrics of the available models. We report experiments to evaluate the RILaaS platform under varying loads of batch size, number of robots, and various model placement hosts on Cloud, Edge, and Fog for providing benchmark applications of object recognition and grasp planning as a service. We address the complexity of load balancing with a Q-learning algorithm that optimizes simulated profiles of networked robots; outperforming several baselines including round robin, least connections, and least model time with 68.30% and 14.04% decrease in round-trip latency time across models compared to the worst and the next best baseline respectively. Details and updates are available at: https://sites.google.com/view/rilaas**

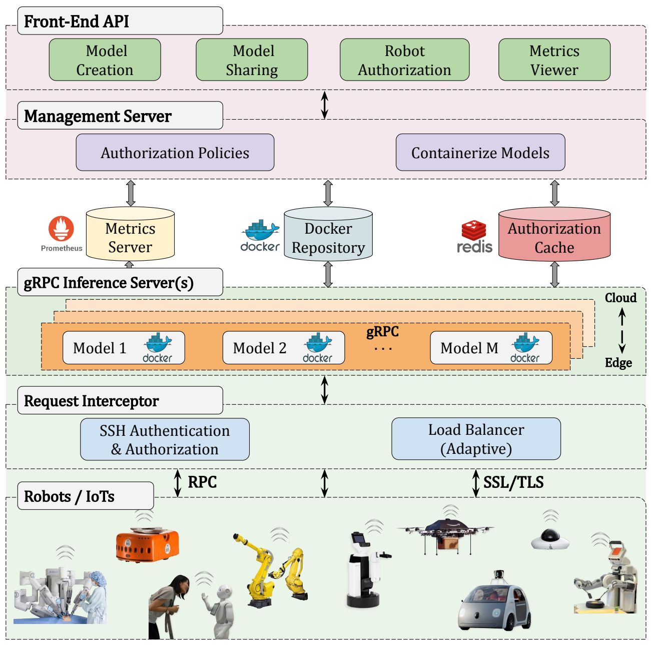


Fig. 1: RILaaS uses a hierarchy of resources in the Cloud-Edge continuum to distribute inference/prediction serving of deep learning models such as grasp planning and object recognition on a fleet of robots. Users can manage robots and models with a front-end API that interacts with the inference loop through a metrics server, authorization cache, and a Docker repository.

I. Introduction

Robot programming has evolved from low level coding to more intuitive methods. Common ways of programming robots include use of a teaching pendant to playback a set of via-points, offline programming with a simulator, programming by demonstration such as kinesthetic teaching, and/or programming by exploration for trial and error learning of the desired task. Despite the variety of interfaces, teaching a new task to a robot requires skilled personnel for data collection, labeling and/or learning a control policy from hundreds of hours of robot training [1]. Instead of retraining a skill for every new situation, we advocate the need of a **programming-by-abstraction** approach where high-level skills such as grasping and object recognition etc. can be acquired in a 'plug-and-play' manner to facilitate programming of new skills.

Recent advancements in deep learning have led to a rise of robotic applications that rely on computationally expensive models such as deep neural networks for perception, planning and control. Typical usage of a deep learning model involves: **training**, **adaptation** and/or **inference**. The training stage involves estimation of model parameters on large scale data, adaptation is the process of transferring/fine-tuning the model to a new environment, while inference requires predicting the model output for a given input. While training and adaptation are computationally and resource intensive, inference decouples model from applications and must be done in real-time to meet the performance requirements of the application. As an example, training a deep object recognition model on ImageNet-1k may last for days, adaptation may take hours, but the inference time is often less than 100 milliseconds.

Robots are increasingly linked to the network and thus not limited by the onboard resources for compute, storage and networking with Cloud and Fog Robotics [2], [3]. By offloading the computational and storage requirements over the network, the robots can share training, adaptation and inference of deep learning models and reduce the burden of collecting and labelling massive data for programming a separate model for each robot. Once trained, the models can be deployed to an inference serving system to meet the performance requirements of the application such as bandwidth, latency, accuracy and so on. To our surprise, there is very little research on how to use/re-use and deploy such

University of California, Berkeley. {ajay.tanwani, raghav98, jegonzal, goldberg}@berkeley.edu

models once they are trained. The focus of this paper is on scalable inference serving of deep models on networked robots.

In this paper, we introduce a novel Robot-Inference-and-Learning-as-a-Service (RILaaS) platform to meet the service level objectives in inference serving of deep models on robots. RILaaS abstracts away applications from the training phase with virtualized computing and storage of models and datasets, thereby, removing hardware and software dependencies on custom middleware. It allows users to easily upload, test, share, monitor and deploy trained models on robots for querying the service ubiquitously. The service optimizes for low latency and scalable inference across a fleet of robots by distributing queries over Cloud and Edge using an adaptive load balancing strategy (see Fig. 1). We observe that using reinforcement learning to optimize the load profiles of networked robots outperforms several baselines including round robin, least connections, and least model time. We show the application of RILaaS to deep object recognition and grasp planning, where a fleet of robots send RGB and/or depth images of the environment over a wireless network as input, and retrieve the object locations and/or grasp configurations as output.

A. Contributions

This paper makes three contributions:

1) We present RILaaS: a novel user-based low-latency inference serving platform to facilitate large-scale use/re-use of deep models for robot programming.
2) We provide examples of deep object recognition and grasp planning as a service with RILaaS and benchmark their performance with varying number of robots, batch sizes and dynamic loads.
3) We optimize the round-trip latency times for scalable inference serving by distributing queries over Cloud and Edge servers with a reinforcement learning algorithm that outperforms several baselines under simulated dynamic loads by at least 14.04% reduction in round-trip latency time compared to the next best least-connections strategy.

II. Related Work

A. Cloud and Fog Robotics

Cloud Robotics provides on-demand availability of configurable resources to support robots' operations [2]. The centralized Cloud approach alone often limits the latency and throughput of data than deemed feasible for many robotics applications. Fog Robotics distributes the resource usage between the Cloud and the Edge in a federated manner to mitigate the latency, security/privacy, and network connectivity issues with the remote Cloud data centers [4], [3], [5]. Popular cloud robotics platforms include RoboEarth [6] – a world-wide web style database to store knowledge generated by humans and robots accessed via Rapyuta platform; KnowRob [7] – a knowledge processing system for grounding the knowledge on a robot; RoboBrain [8] – a large scale computational system that learns from publicly available resources over internet; rosbridge [9] – a communication package between the robot and the Robot Operating System (ROS) over Cloud; while Dex-Net as a Service (DNaaS) [10] are recent efforts to provide Cloud-based services for analytical grasp planning.

To the best of our knowledge, RILaaS is the first user-based data-driven general purpose inference serving platform for programming robots. We provide grasp planning and single-shot object recognition services as an example where the robots send RGB and/or depth images of the environment and retrieve the recognized objects and the grasp locations for robotic manipulation.

B. Inference Serving

Inference serving is emerging as an important part of a machine learning pipeline for deploying deep models. The growing demand of machine learning based services such as image recognition, speech synthesis, recommendation systems etc. is resulting in tighter latency requirements and more congested networks. Large tech companies have built their private model serving infrastructure to handle scaling, performance, and life cycle management in production, however, their adoption in a wider machine learning and robotics community is rather limited.

A simple way to deploy a trained model is to make a REST API using Flask. Although simple and quick, it often causes scale, performance, and model life cycle management issues in production. **Tensorflow-serving** uses SavedModels to package the trained models for scaling and sharing of the deployed models [11]. The serving, however, does not support arbitrary pre-processing and post-processing of the data which limits a range of applications. **Clipper** supports a wide variety of frameworks including Caffe, Tensorflow and Scikit-learn for inference serving in the Cloud. Additionally, it uses caching and adaptive batching to improve the inference latency and throughput [12]. **InferLine** combines a planner and a reactive controller to continuously monitor and optimize the latency objectives of the application [13]. **Rafiki** optimizes for model accuracy with a reinforcement learning algorithm subject to service level latency constraints [14]. **INFaaS** automatically navigates the decision space on behalf of users to meet user-specified objectives [15]. Recently, a number of companies have entered the model serving space with Amazon Apache MXNet, Nvidia TensorRT, Microsoft ONNX and Intel OpenVino to satisfy the growing application demands. All these services are typically optimized to serve specific kinds of models in the Cloud only. Moreover, creation or updating of the models at the back end is manual and cumbersome. In comparison to these services, RILaaS allows users to upload trained deep models, share with other users and/or make them publicly available for others to test models with custom data and easily deploy on new robots for querying the trained models. It distributes the queries over Cloud and Edge to satisfy more stringent service level objectives than possible with inference serving in the Cloud only.

C. Inference Optimization

Deploying deep learning models is not just about setting up the web server API, but ensuring that the service is scalable and the requests are optimized for service level objectives. The Cloud provides auto-scalable resources for compute and storage, whereas resources at the Edge of the network are limited. Edge and Fog Computing brings Cloud-inspired computing, storage, and networking closer to the robot where the data is produced [16], [17]. Quality of service provisioning depends upon a number of factors such as communication latency, energy constraints, durability, size of the data, model placement over Cloud and/or Edge, computation times for learning and inference of the deep models, etc [18]. Nassar and Yilmaz [19] and Baek et al. [20] allocate resources in the Fog network with a reinforcement learning based load balancing algorithm. Chinchali et al. use a deep reinforcement learning strategy to offload robot sensing tasks over the network [21].

RILaaS takes a distributed approach to inference serving where a load-balancer receives inference requests from nearby robots/clients at the Edge and learns to decide whether to process the requests on Cloud or Edge servers based on their resource consumption. We show its application to vision-based grasping and object recognition and investigate the inference scaling problem by simulating increasing number of requests in the network.

III. Problem Formulation and Challenges

Consider a multi-agent setting of M robots $\langle r_1 \dots r_M \rangle$ each having access to a set of trained models or policies $\langle \pi_1 \dots \pi_D \rangle$ that are deployed on a set of N servers. Each model may be deployed on one or more servers, and the location of each server is fixed either on Cloud, Edge or anywhere along the continuum. The m-th robot observes the state of the environment as $\{\boldsymbol{\xi}_t^{(m)}\}_{t=1}^{T_B}$ in a mini-batch of size T_B, sends the request asynchronously to the inference service and receives the response $\{\boldsymbol{y}_t^{(m)}\}_{t=1}^{T_B}$. The job of the inference service is to compute the responses $\{\boldsymbol{y}_t^{(m)} = \pi_d(\boldsymbol{\xi}_t^{(m)})\}_{t=1}^{T_B}$ for the requested d-th model such that the round-trip latency time $t^{(\mathrm{rtt})}$ is optimized in communication with the set of robots, while preserving the privacy and security of the data. Note that we do not consider the transfer problem of adapting the model output to new environments in this work, and only address the scalability issues in inference serving of deep models on a fleet of robots.

To this end, we introduce a novel user-based inference serving platform for deploying deep learning models on robots, and apply reinforcement learning for optimizing the round-trip latency times under dynamic loads. Next, we describe the specific challenges in developing the general purpose inference serving platform and discuss the RILaaS methodology to address the outlined issues.

Model Support: Prominent machine learning frameworks such as PyTorch, Tensorflow, Spark, Caffe are widely used for training and adaptation of deep models. Deploying these multiple frameworks on a robot or a set of inference servers is complex because of conflicting dependencies between each framework. RILaaS accepts any arbitrary model for deployment by using Docker containers to allow each framework to exist independently of the other. Each container can be customized to the requirements of a particular framework. The containers accept inputs of `Map<name, numeric array>` and return outputs of the same form, where the map function adapts the model inputs and outputs to the RILaaS format.

Rapidly Deployable: RILaaS abstracts away applications from models to facilitate ease of deployment on custom hardware with varying specifications. It only requires the public SSH key of the robot for authenticating and subscribing to the required models, after which the robot can readily access model outputs over a network call.

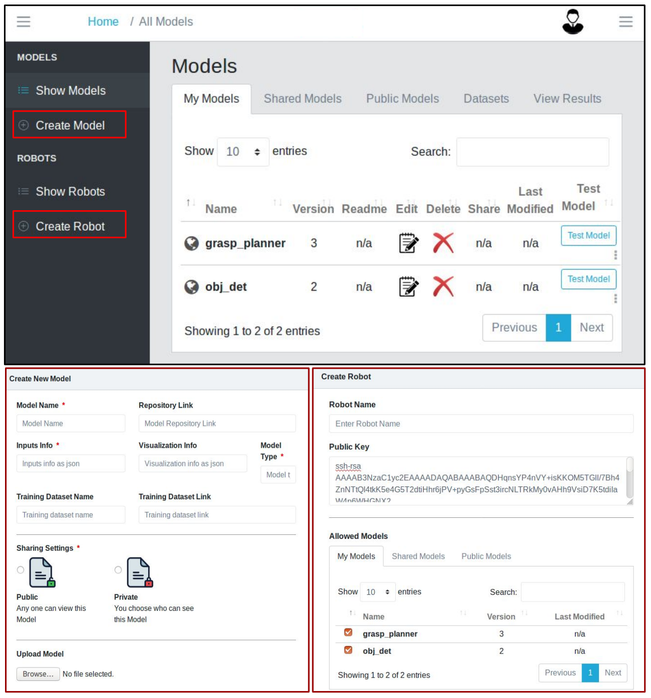
Security and Privacy: Inference serving by transmitting sensitive data over untrusted wireless networks (such as images of private locations) is vulnerable to data infiltration and cyber attacks. Additionally, targeted Denial of Service (DoS) attacks can be a bottleneck to meet the bandwidth requirements of time-sensitive applications [22]. Hosting models on the Edge of the network can keep data private and the network secure, but it comes at the cost of developing and maintaining a heterogeneous Edge infrastructure. RILaaS uses a Fog robotics approach to place models on the Cloud and the Edge servers depending upon the security requirements specified by the user. This allows access to the auto-scalable compute and storage capacity of the Cloud for low-sensitivity models while using secure but less powerful Edge infrastructure for private data. Moreover, RILaaS's front end allows easy management of access controls on a per-robot per-model basis.

Scalable Workloads: Robots may have to trade-off between doing fast inference on a remote server using hardware accelerators such as a GPU while incurring additional network overhead or doing slow inference locally. Latency times need be optimized to deal with dynamic application dependent workloads. RILaaS optimizes the inference serving latency for each individual model by using reinforcement learning to distribute queries over the Cloud and the Edge servers according to their resource consumption.

Performance Monitoring: Monitoring the inference service is useful to evaluate the empirical accuracy and latency characteristics in comparison to the service level objectives of the application. RILaaS allows users to specify and log metrics for each model and each robot over a front-end.

IV. RILaaS Architecture

RILaaS is divided into four modules: **1) Front-end**, **2) Management Server**, **3) Inference Server** and **4) Request Interceptor**. The front-end provides a simple interface to upload trained models and deploy them on robots. The management server is responsible for storing the authorization policies and deploying the containerized models on requested servers. The inference server computes the response of the incoming queries using specified models. The request interceptor authorizes the use of specified models,



Home / All Models

MODELS

Show Models

Create Model

ROBOTS

Show Robots

Create Robot

Models

My Models | Shared Models | Public Models | Datasets | View Results

Show 10 entries Search:

Name	Version	Readme	Edit	Delete	Share	Last Modified	Test Model
grasp_planner	3	n/a			n/a	n/a	Test Model
obj_det	2	n/a			n/a	n/a	Test Model

Showing 1 to 2 of 2 entries Previous 1 Next

Create New Model

Model Name * : Model Name

Repository Link: Model Repository Link

Inputs Info * : Inputs info as json

Visualization Info: Visualization info as json

Model Type * : Model t

Training Dataset Name: Training dataset name

Training Dataset Link: Training dataset link

Sharing Settings *

Public — Any one can view this Model

Private — You choose who can see this Model

Upload Model

Browse... No file selected.

Create Robot

Robot Name: Enter Robot Name

Public Key:

ssh-rsa
AAAAB3NzaC1yc2EAAAADAQABAAABAQDHqnsYP4nVY+isKKOM5TGlI/7Bh4
ZnNTtQl4tkK5e4G5T2dtiHhr6jPV+pyGsFpSst3ircNLTRkMy0vAHh9VsiD7K5tdila
W4n6WHGNX2

Allowed Models

My Models | Shared Models | Public Models

Show 10 entries Search:

	Name	Version	Last Modified
✓	grasp_planner	3	n/a
✓	obj_det	2	n/a

Showing 1 to 2 of 2 entries Previous 1 Next

Fig. 2: Front-end API snapshots (not shown to scale): *(top)* Users can upload, share and visualize models and datasets, *(botom-left)* interface to upload new models and set access control policies, *(bottom-right)* interface to deploy available models on robots.

while the load-balancer hosted on the request interceptor learns to distribute queries over multiple inference servers. Additionally, the monitoring server collects metrics about the model and the robot performance. The user first uploads or chooses a publicly shared model over the front-end where it is containerized and deployed on the inference server. Robots are added by specifying their public SSH key and subscribing to the desired models. Robots can then query the deployed models over the network using a minimalist client library. The monitoring server runs in the background to log the desired metrics for visualization via the front-end. The overall architecture is summarized in Fig. 1.

A. User End: Front-End and Management Server

RILaaS provides a user-facing REST API that interacts with the Django management server to create, view and update models, datasets, robots and metrics (see Fig. 2 for front-end snapshots). The front-end is a user-based platform that provisions for:

Model Creation: Users upload the model folder containing the pre-trained model weights and specify the input, output types and optional pre-processing and post-processing modules. The management server containerizes the model automatically and uploads the image in a docker repository hosted on AWS. We package each model in a separate Docker container to resolve system conflicts between models and prevent over-utilization of system resources.

Model Sharing: Users can make their models private, public or share with other users on the platform to facilitate re-usability of models across applications.

Robot Creation: Users deploy the uploaded models on robot(s) by adding their public SSH key for authentication. Note that all publicly available models are automatically made available to any robot registered with the service.

Dataset Creation: The front-end allows users to upload test datasets for querying the uploaded models and visualizing the model outputs. The test datasets can similarly be made public for other users to test the models. This allows users to ensure the functioning of their deployed models before querying them from the robot.

Metrics Viewer: A flexible query interface through Prometheus allows users to view metrics about their model/robot such as requests sent/received and the round-trip communication latency times. Additional end-points for metrics can be added via a dedicated endpoint that is asynchronously monitored by the management server.

B. Robot End: Request Interceptor and Inference Server

Request Interceptor receives the incoming requests from the networked robots and distributes them to the inference servers. The request interceptor may be deployed on the robot itself or centrally at the Edge of the network for a fleet of robots. Note that multiple request interceptors can also be deployed for the same application. The request interceptor is responsible for SSH based authentication of the robots and authorizing access control for the models. Authentication and authorization policies prevent misuse of compute resources by intruders. Authentication is done using JSON Web Tokens (JWT) signed with private SSH key of the robot, while authorization policies are stored in a database in the management server. Naively fetching model access policies from remote databases for every request can slow down inference, thereby, these access policies are stored on a local Redis cache to minimize network calls to a remote database for each robot query. The cache is updated using an event-triggered system that maintains the most recent version of access control policies from the management server. The request interceptor subsequently directs the authorized queries to the inference servers using a user-specified load balancing strategy to optimize the round-trip latency times.

Inference Servers deploy the containerized models on provisioned servers to process the incoming requests. The servers may be placed on Cloud, Edge and/or anywhere along the continuum depending upon the application requirements. Modular resource placement allows the robots to access resources from the Edge and seamlessly switch to the Cloud for scalability if Edge resources are not sufficient to meet the service level objectives. Moreover, non-critical models can also be rate limited on a per-robot basis in order to prevent DoS attacks from occurring at the Edge and ensure high availability of important models.

C. Inference Query Life Cycle

RILaaS abstracts away the hardware and software dependencies required for inference of deep robot models. Once a model has been deployed on the RILaaS platform, a robot or a fleet of robots can readily access the deep models by

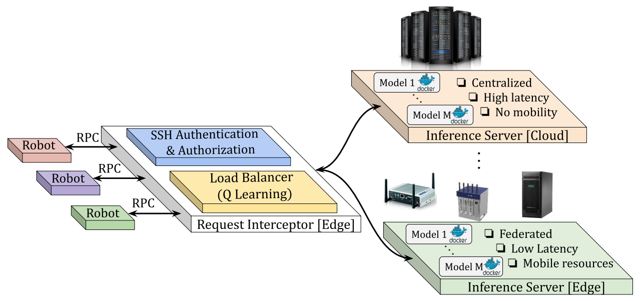


Fig. 3: Inference optimization with adaptive load-balancing: A Q-Learning algorithm adapts the distribution of the incoming requests between the Cloud and the Edge resources to optimize the round-trip latency time.

a simple network call after installing the minimalist RILaaS client python package. As shown in the code snippet below, the `RobotClient` object contains the necessary parameters for authentication and authorization of the robot and the required deep model. The robot specifies the target address of the request interceptor, the model name and the model version for inference, the private SSH key of the robot for inference and the SSL certificate location. The SSL certificate encrypts the communication between the robot and the servers. The robot communicates with the servers using gRPC, an open source Remote Procedure Call library built on HTTP/2. Once it is created, the `RobotClient` object is used to make predictions with a simple function call.

```
from client import RobotClient
rc = RobotClient(
            TARGET_IP,
            MODEL_NAME,
            MODEL_VERSION,
            PRIVATE_SSH_KEY_PATH,
            SSL_CERTIFICATE_LOCATION
      )
outputs = rc.predict(inputs)
```

V. Inference Optimization with Adaptive Load-Balancing

The inference requests from a robot or a fleet of robots can be optimized for large-scale serving of deep models. A-priori estimation of querying rate of the model and the round-trip inference time of the model provide a useful criteria for inference optimization. Ensemble modeling is also useful to deploy multiple models of the same task and optimize the inference times. Appropriate model selection among ensembles provides a trade-off between accuracy and latency to satisfy the service level objectives [12], [14]. Optimizing the placement and use of resources can also increase the overall system efficiency. For example, simple application profiling may be used for resource placement in a constrained network where there are many CPUs and few GPUs. Finding an appropriate balance for performance and cost, however, is challenging when the application demands and the availability of resources keeps changing over time, making continuous re-evaluation necessary [23].

Load balancing across multiple servers is useful for optimizing resource utilization, reducing latency and ensuring fault-tolerant configurations [24]. Traditional load balancing strategies supported in RILaaS include,

Round Robin: Requests are distributed in a cyclic order regardless of the load placed on each server.

Least Connections: The next request is assigned to the server with the least number of active connections.

Least Model Time: Requests are assigned based on running estimate of average round-trip latency for each model. To prevent choosing a single server for extended periods of time, we randomize the server selection with a small probability to explore all available resources.

We use *nginx* [25] for load-balancing with round robin or least connections. The nginx load balancing strategies naively assume homogeneity of servers, i.e., each request takes a similar amount of time to process on available resources. Moreover, the heuristics used in these strategies are not suitable for handling dynamic loads where the number of requests vary over time. In this work, we seek to optimize the inference times under dynamic loads by distributing queries over a set of non-homogeneous servers between the Edge and the Cloud (see Fig. 3 for an overview).

We formulate the adaptive load-balancing as a reinforcement learning problem to minimize the expected round-trip latency for each request in a given time horizon on a per-model basis. We assign an 'agent' to each model to distribute the incoming queries, i.e., the number of agents scale linearly with the number of models used. Each agent keeps an estimate of each server in a Markov decision process tuple $\langle \mathcal{S}, \mathcal{A}, \mathcal{R} \rangle$ where $\boldsymbol{s}_t \in \mathcal{S}$ is the state representation of the server at time t, $\boldsymbol{a}_t \in \mathcal{A}$ is the action of sending request to one of the N servers which results in transition to a new state $\boldsymbol{s}'_t \in \mathcal{S}'$ along with the reward $r(\boldsymbol{s}_t, \boldsymbol{a}_t) \in \mathcal{R}$ as an estimate of the round-trip latency, i.e.,

$$\boldsymbol{s}_t = \begin{bmatrix} p_{t,1} \ , \ q_{t,1} \\ p_{t,2} \ , \ q_{t,2} \\ \vdots \\ p_{t,N} \ , \ q_{t,N} \end{bmatrix}, \quad \boldsymbol{a}_t = \begin{bmatrix} 1 \\ 2 \\ \vdots \\ N \end{bmatrix}, \quad r_t = -\left(1 + \mathcal{L}(\boldsymbol{s}_t, \boldsymbol{a}_t)\right)^2, \tag{1}$$

where $p_{t,i}$ is the number of requests of a model on server i at time t, $q_{t,i}$ represents the total number of active requests of all models on server i at time t, and $\mathcal{L}(\boldsymbol{s}_t, \boldsymbol{a}_t) \in \mathbb{R}$ is the round-trip latency of inference query cycle, i.e., time required to send the request and receive the response from the service. Note that the reward function penalizes the increase of latency times in a quadratic manner. The agent learns to choose the server by taking action $\boldsymbol{a}_t$ such that the expected latency in a given time horizon is minimized from inference request load profiles of networked robots. The expected latency is estimated by the Q-function $Q(\boldsymbol{s}_t, \boldsymbol{a}_t) \in \mathbb{R}$,

$$Q(\boldsymbol{s}_t, \boldsymbol{a}_t) = \mathbb{E}\left[\sum_{t=0}^{T} \gamma^t r(\boldsymbol{s}_t, \boldsymbol{a}_t)\right], \ \boldsymbol{a}_t = \underset{\boldsymbol{a}_t = 1 \ldots N}{\arg\max} \ Q(\boldsymbol{s}_t, \boldsymbol{a}_t), \tag{2}$$

where $\gamma \in \mathbb{R}$ is the discount factor of future rewards. The Q-function is recursively updated using the Bellman equation [26]. With a small probability, a server is randomly

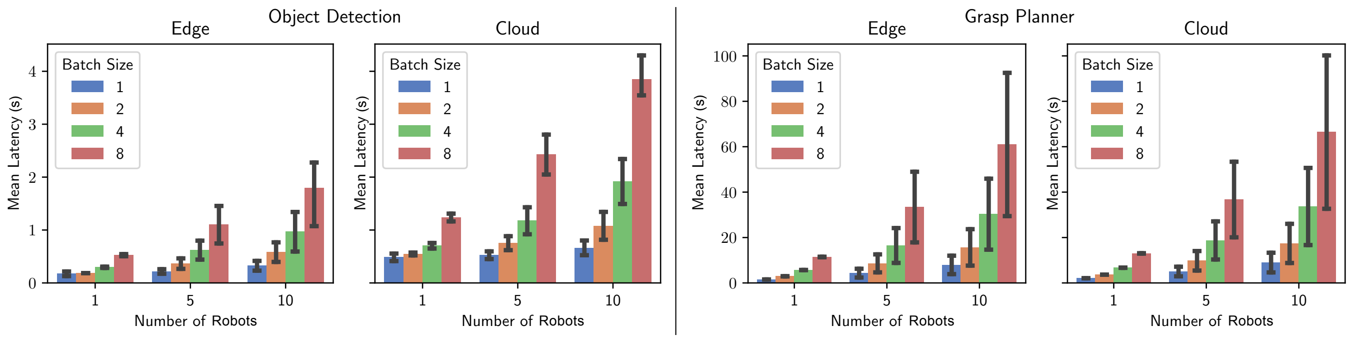


Fig. 4: Comparison of the average round-trip latency times in seconds of the object recognition model on *(left)* and the grasp planning model on *(right)* with the use of Edge or Cloud resources (same latency scale is used for both resources). We make two observations: 1) the round-trip communication time scales sub-linearly with increasing batch size and number of robots across both models, 2) the difference between the Edge and the Cloud latency times is dominant when the computation time is less than the communication time as for the object recognition model in comparison to the grasp planning model.

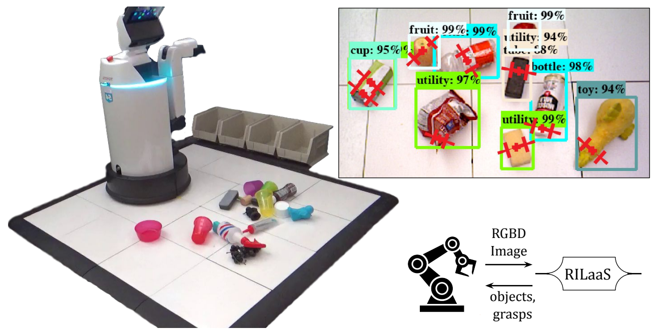


Fig. 5: Vision-based decluttering application where the robots send the RGBD image of the environment to the inference service and retrieves the object categories and bounding boxes, along with their grasp locations to put the objects in their corresponding bins.

chosen to encourage exploration of the state and action space. The agent continuously optimizes the action selection to drive down the latency times for each model based on the observed load profiles from the networked robots. Note that we assume the location and the number of servers to be fixed and each model is deployed on all servers without loss of generality. In case the number of servers change as in starting additional Cloud instances on performance drop, the adaptive load-balancing policy needs to be retrained.

VI. Experiments and Results

We now present experiments for evaluating the RILaaS platform to serve deep models of object recognition and grasp planning on a large scale. We empirically investigate the effect of varying batch size, number of robots and resource placement, followed by the adaptive load-balancing experiments to optimize simulated dynamic load profiles with a fleet of robots. We use the Amazon EC2 (East) `p2.1xlarge` instance with 1 Tesla K80 GPU in Northern Virginia (us-east-1) for Cloud compute and use Amazon S3 buckets for Cloud storage. The Edge infrastructure comprises of a workstation with 1 NVidia V100 GPU located at a nearby data center.

A. Application Workloads

We consider real-world application scenarios where RILaaS is used to provide object recognition and grasp planning as a service for vision-based robot decluttering, building upon our previous work in [3], [27].

Object Recognition: We use the MobileNet-Single Shot MultiBox Detector (SSD) model with focal loss and feature pyramids as the base model for object recognition. The input RGB image is fed to a pre-trained VGG16 network, followed by feature resolution maps and a feature pyramid network, before being fed to the output class prediction and box prediction networks. The model is trained on 12 commonly used household and machine shop object categories using a combination of synthetic and real images of the environment.

Grasp Planning: Grasping diversely shaped and sized novel objects has a wide range of applications in industrial and consumer markets. Robots in homes, factories or warehouses require robust grasp plans in order to interact with objects in their environment. We use an adaptation of the Dex-Net grasp planning model to plan grasps from the depth images of the environment. The model samples antipodal grasp pairs from a depth image and feeds them to a convolutional neural network to predict the probability of successful grasp as determined by the wrench resistance metric. The sampled grasps are successively filtered with a cross-entropy method to return the most likely grasp. Note that the pre-processing step of sampling many different grasps requires CPU usage, whereas predicting the grasp success requires GPU resources for efficient grasp planning.

Vision-Based Decluttering: We sequentially pipeline the object recognition and grasp planning models together for vision-based surface decluttering [3]. The robot sends RGBD images of the environment, where the RGB image is used for object recognition and the cropped depth image from the output bounding box of the object recognition model is used by the grasp planning model to output the top ranked grasp for the robot to pick and place the object into its corresponding bin (see Fig. 5).

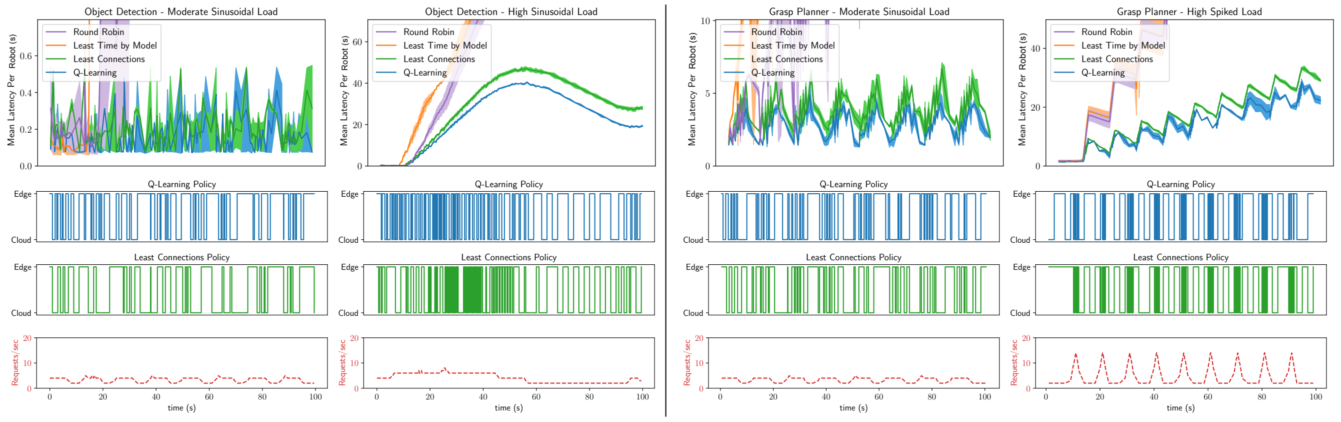


Fig. 6: Inference optimization of varying test load profiles for object recognition on *(left)* and grasp planning on *(right)*. For each model, top row shows the round-trip latency of load-balancing strategies, second and third row shows the Q-learning and least connections policy output in allocating Edge or Cloud resources, fourth row shows the requests rate profile. Q-learning scales better with increasing loads than other load-balancing strategies by optimally figuring out how to use Edge resources more frequently to reduce the average round-trip latency times.

TABLE I: Computation time for inference $t^{(\mathrm{inf})}$ vs round trip communication time $t^{(\mathrm{rtt})}$ (in milliseconds) for inference over Edge and EC2-East Cloud. Results are averaged across 6 trials. Communication time dominates the computation time and increases as the distance to the server increases.

Location	$t^{(\mathrm{inf})}$	$t^{(\mathrm{rtt})}$
Object Detection		
EC2-East	42.79 ± 0.41	483.82 ± 70.87
Edge	$\mathbf{36.03 \pm 3.18}$	$\mathbf{172.77 \pm 43.55}$
Grasp Planner		
EC2-East	1501.61 ± 12.76	2051.48 ± 22.684
Edge	$\mathbf{1386.95 \pm 22.92}$	$\mathbf{1515.59 \pm 26.16}$

B. Scalability of RILaaS

We deploy the trained models on the RILaaS platform to receive images from the robot, perform inference, and send back the output results to the robot. We measure the round-trip time $t^{(\mathrm{rtt})}$, i.e., time required for communication to/from the server and the inference time $t^{(\mathrm{inf})}$, i.e., time required to compute the model response for a given input. We experiment with two hosts for the inference service: EC2 Cloud (East), and Edge with GPU support.

Resource Placement with Cloud vs Edge: Results in Table I show that the communication time is a major component of the overall round-trip latency time. Deploying the inference service on the Edge significantly reduces the round-trip inference time and the timing variability in comparison to hosting the service on Cloud, while incurring a communication overhead of around 100 milliseconds only. The difference in resource placement is less pronounced for the grasp planning model where CPU computation time in sampling grasp pairs is a dominant factor. Moreover, the authentication time only takes 1 millisecond on average with Redis cache in comparison to 630 milliseconds with a relational database on AWS.

Effect of Batch Size and Number of Robots: We next vary the batch size and number of robots making concurrent requests to the service. Fig. 4 suggests that the average round-trip latency grows sub-linearly with the batch size and the number of robots querying the service. Moreover, deploying models on Edge yields lower round-trip latency times across both models, but the difference is more pronounced for the object recognition model with lower inference time than the grasp planning model.

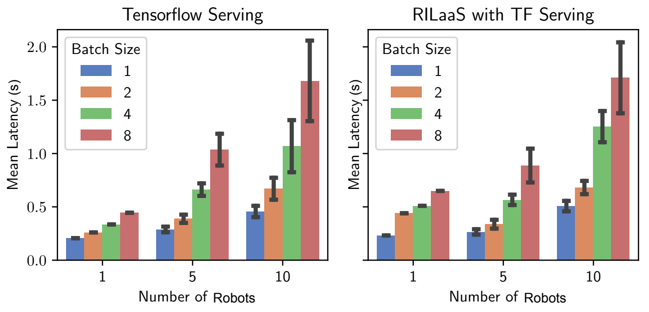


Fig. 7: A comparison between mean latency of tensorflow Serving and RILaaS for the object detection model at the Edge. RILaaS performs on par with tensorflow serving and the gap closes further with more robots.

Comparison with Tensorflow Serving: Fig. 7 suggests that RILaaS gives comparable results to tensorflow-serving for the object recognition model deployed at the Edge. Note that the tensorflow-serving does not provide out-of-the-box pre-processing/post-processing, authentication, authorization and metrics viewing for models that it supports. Consequently, the grasp planning model cannot be hosted on tensorflow-serving as it iterates over preprocessing and inference. RILaaS supports a tensorflow-serving backend while providing the aforementioned features to make it feasible for deploying a wide variety of models.

C. Inference Optimization under Dynamic Loads

We next simulate time-varying requests of different profiles to evaluate the performance of inference optimization with adaptive load-balancing. We query the object recognition and the grasp planning model alternatively at specified rates to simulate the decluttering setup, and compare the Q-learning based adaptive load-balancing with round robin, least connections and least model time strategies. The request profiles include: 1) **uniform loads** of $1, 2, 4, 8$ requests per second, 2) **step-wise increasing loads** of $1, 2, 3, 4$ requests per second, 3) **spiked loads** where nominal load of 2 requests per second is augmented with 13 requests per second for up to 2 seconds, 4) **Poisson distributed loads** where requests follow the Poisson process with arrival rate of $1, 2, 4, 8$ requests per second, 5) **sinusoidal loads** with varying amplitudes and frequencies of $0.05, 0.01, 0.08$ Hz. The first 4 types of load profiles are used for both training and testing, while the sinusoidal load profiles are only used for testing of the optimal inference serving policy.

Fig. 6 shows the plots of the object recognition and grasp planning model for various request profiles. It can be seen that the Q-learning strategy outperforms the commonly used load-balancing strategies. Least-connections performance is better among the fixed load-balancing strategies and its performance is similar to Q-learning for lighter workloads. The inference serving policy reveals that the Q-learning is able to decrease the average latency times by more frequently using the Edge resource as compared to the Cloud. Overall, the adaptive load-balancing strategy with Q-learning for object recognition gives 15.76% and 70.7% decrease in round-trip latency time compared to the next best least connections and worst performing round-robin baseline. Similarly, the grasp planning model shows 12.32% and 65.91% decrease in the round-trip latency time with Q-learning in comparison to least connections and round-robin strategies.

VII. Conclusions and Future Directions

Virtualizing robot storage, compute and programming is a key enabler for large-scale learning and inference of deep models for robotic applications. In this paper, we have introduced RILaaS as a novel user-based inference serving platform for deploying deep learning models on robots that satisfies heterogeneous model support, rapid deployment, security and privacy, and low latency requirements of the applications. We used reinforcement learning for scalable inference serving that adapts better with dynamic loads than commonly used load balancing strategies. We provide deep object recognition and grasp planning as a service and showed its application to vision-based decluttering of objects from the floor and depositing them in target bins. To the best of our knowledge, RILaaS is the first of its kind user based inference serving platform of deep models for robotic applications.

References

[1] S. Levine, P. Pastor, A. Krizhevsky, J. Ibarz, and D. Quillen, "Learning hand-eye coordination for robotic grasping with deep learning and large-scale data collection," *IJRR*, vol. 37, no. 4-5, pp. 421–436, 2018.

[2] B. Kehoe, S. Patil, P. Abbeel, and K. Goldberg, "A survey of research on cloud robotics and automation," *Automation Science and Engineering, IEEE Transactions on*, vol. 12, no. 2, pp. 398–409, 2015.

[3] A. K. Tanwani, N. Mor, J. Kubiatowicz, J. E. Gonzalez, and K. Goldberg, "A fog robotics approach to deep robot learning: Application to object recognition and grasp planning in surface decluttering," in *IEEE Intl Conf. on Robotics and Automation (ICRA)*, 2019, pp. 4559–4566.

[4] S. L. K. C. Gudi, S. Ojha, B. Johnston, J. Clark, and M.-A. Williams, "Fog robotics for efficient, fluent and robust human-robot interaction," 2018.

[5] D. Song, A. K. Tanwani, and K. Goldberg, *Robotics goes MOOC*. Springer: Springer Nature MOOCs, 2019, ch. Networked-, Cloud- and Fog-Robotics.

[6] M. Waibel, M. Beetz, J. Civera, R. D'Andrea, J. Elfring, D. Gálvez-López, K. Häussermann, R. Janssen, J. Montiel, A. Perzylo, B. Schieß le, M. Tenorth, O. Zweigle, and R. De Molengraft, "RoboEarth," *IEEE Robotics & Automation Magazine*, vol. 18, no. 2, pp. 69–82, 2011.

[7] M. Beetz, D. Beßler, A. Haidu, M. Pomarlan, A. K. Bozcuoğlu, and G. Bartels, "Know rob 2.0 — a 2nd generation knowledge processing framework for cognition-enabled robotic agents," in *2018 IEEE Int. Conf. on Robotics and Automation (ICRA)*, 2018, pp. 512–519.

[8] A. Saxena, A. Jain, O. Sener, A. Jami, D. K. Misra, and H. S. Koppula, "Robobrain: Large-scale knowledge engine for robots," *CoRR*, vol. abs/1412.0691, 2014.

[9] M. Quigley, K. Conley, B. P Gerkey, J. Faust, T. Foote, J. Leibs, R. Wheeler, and A. Y Ng, "Ros: an open-source robot operating system," in *ICRA Wksp on Open Source Software*, vol. 3, Jan. 2009.

[10] L. Pusong, B. DeRose, J. Mahler, J. A. Ojea, A. K. Tanwani, and K. Goldberg, "Dex-net as a service (dnaas): A cloud-based robust robot grasp planning system," in *14th International Conference on Automation Science and Engineering (CASE)*, 2018, pp. 1–8.

[11] C. Olston, F. Li, J. Harmsen, J. Soyke, K. Gorovoy, L. Lao, N. Fiedel, S. Ramesh, and V. Rajashekhar, "Tensorflow-serving: Flexible, high-performance ml serving," in *Workshop on ML Systems at NIPS*, 2017.

[12] D. Crankshaw, X. Wang, G. Zhou, M. J. Franklin, J. E. Gonzalez, and I. Stoica, "Clipper: A low-latency online prediction serving system," *CoRR*, vol. abs/1612.03079, 2016.

[13] D. Crankshaw, G. Sela, C. Zumar, X. Mo, J. E. Gonzalez, I. Stoica, and A. Tumanov, "Inferline: ML inference pipeline composition framework," *CoRR*, vol. abs/1812.01776, 2018.

[14] W. Wang, S. Wang, J. Gao, M. Zhang, G. Chen, T. K. Ng, and B. C. Ooi, "Rafiki: Machine learning as an analytics service system," *CoRR*, vol. abs/1804.06087, 2018.

[15] F. Romero, Q. Li, N. J. Yadwadkar, and C. Kozyrakis, "Infaas: Managed & model-less inference serving," *CoRR*, vol. abs/1905.13348, 2019.

[16] F. Bonomi, R. Milito, J. Zhu, and S. Addepalli, "Fog computing and its role in the internet of things," in *Proceedings of the first edition of the MCC workshop on Mobile cloud computing*. ACM, 2012, pp. 13–16.

[17] W. Shi, J. Cao, Q. Zhang, Y. Li, and L. Xu, "Edge computing: Vision and challenges," *IEEE Internet of Things Journal*, vol. 3, no. 5, pp. 637–646, 2016.

[18] M. Mukherjee, L. Shu, and D. Wang, "Survey of fog computing: Fundamental, network applications, and research challenges," *IEEE Communications Surveys Tutorials*, pp. 1–1, 2018.

[19] A. T. Nassar and Y. Yilmaz, "Reinforcement-learning-based resource allocation in fog radio access networks for various iot environments," *CoRR*, vol. abs/1806.04582, 2018.

[20] J. Baek, G. Kaddoum, S. Garg, K. Kaur, and V. Gravel, "Managing fog networks using reinforcement learning based load balancing algorithm," *CoRR*, vol. abs/1901.10023, 2019.

[21] S. Chanchali, A. Sharma, J. Harrison, A. Elhafsi, D. Kang, E. Pergament, E. Cidon, S. Katti, and M. Pavone, "Network offloading policies for cloud robotics: a learning-based approach," *CoRR*, vol. abs/1902.05703, 2019.

[22] P. Pop, M. L. Raagaard, M. Gutierrez, and W. Steiner, "Enabling fog computing for industrial automation through time-sensitive networking (tsn)," *IEEE Communications Standards Magazine*, vol. 2, no. 2, pp. 55–61, 2018.

[23] X. Xu, S. Fu, Q. Cai, W. Tian, W. Liu, W. Dou, X. Sun, and A. Liu, "Dynamic resource allocation for load balancing in fog environment," *Wireless Communications and Mobile Computing*, pp. 1–15, 2018.

[24] A. Schaerf, Y. Shoham, and M. Tennenholtz, "Adaptive load balancing: A study in multi-agent learning," *CoRR*, vol. cs.AI/9505102, 1995.

[25] W. Reese, "Nginx: The high-performance web server and reverse proxy," *Linux J.*, vol. 2008, no. 173, 2008.

[26] R. S. Sutton and A. G. Barto, *Reinforcement Learning: An Introduction*. Cambridge, MA, USA: A Bradford Book, 2018.

[27] B. Staub, A. K. Tanwani, J. Mahler, M. Breyer, M. Laskey, Y. Takaoka, M. Bajracharya, R. Siegwart, and K. Goldberg, "Dex-Net MM: Deep Grasping for Surface Decluttering with a Low-Precision Mobile Manipulator," in *IEEE International Conference on Automation Science and Engineering (CASE)*, 2019, pp. 1–7.