2024 IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS)
October 14-18, 2024. Abu Dhabi, UAE

PEERNet: An End-to-End Profiling Tool for Real-Time Networked Robotic Systems

Aditya Narayanan, Pranav Kasibhatla, Minkyu Choi, Po-han Li, Ruihan Zhao, and Sandeep Chinchali

***Abstract*— Networked robotic systems balance compute, power, and latency constraints in applications such as self-driving vehicles, drone swarms, and teleoperated surgery. A core problem in this domain is deciding when to offload a computationally expensive task to the cloud, a remote server, at the cost of communication latency. Task offloading algorithms often rely on precise knowledge of system-specific performance metrics, such as sensor data rates, network bandwidth, and machine learning model latency. While these metrics can be modeled during system design, uncertainties in connection quality, server load, and hardware conditions introduce real-time performance variations, hindering overall performance. We introduce PEERNet, an end-to-end and real-time profiling tool for cloud robotics. PEERNet enables performance monitoring on heterogeneous hardware through targeted yet adaptive profiling of system components such as sensors, networks, deep-learning pipelines, and devices. We showcase PEERNet's capabilities through networked robotics tasks, such as image-based teleoperation of a Franka Emika Panda arm and querying vision language models using an Nvidia Jetson Orin. PEERNet reveals non-intuitive behavior in robotic systems, such as asymmetric network transmission and bimodal language model output. Our evaluation underscores the effectiveness and importance of benchmarking in networked robotics, demonstrating PEERNet's adaptability. Our code is open-source and available at** `github.com/UTAustin-SwarmLab/PEERNet`.

I. Introduction

Networked robotics [1], [2] is a proven framework for addressing compute, power, and latency constraints in robotic systems, with applications in self-driving vehicles [3], drone swarms [4], and teleoperated surgery [5]. A key challenge in networked robotics is to decide when to offload computationally expensive tasks to a remote server at the cost of network latency. Previous works have studied optimal strategies for data sharing and computation offloading in various networked robotics settings [6], [7], [8], [9], [10]. However, these algorithms face an issue in deployment: they lack real-time data on system performance. Without real-time monitoring of dynamic real-world systems, networked robotics is difficult to optimize.

For example, consider an autonomous vehicle that offloads perception and localization tasks to a remote server, the so-called cloud, over a wireless network connection. Some practical system design questions include:

1) How can we select a deep learning model that ensures decision accuracy while minimizing latency?
2) How is the overall latency affected by the type of connectivity, *e.g.*, Wi-Fi, LTE vs. 5G?
3) What models of LiDAR and cameras provide data rates fast enough to match the network bandwidth?

The University of Texas at Austin.
Corresponding author: `adityan@utexas.edu`

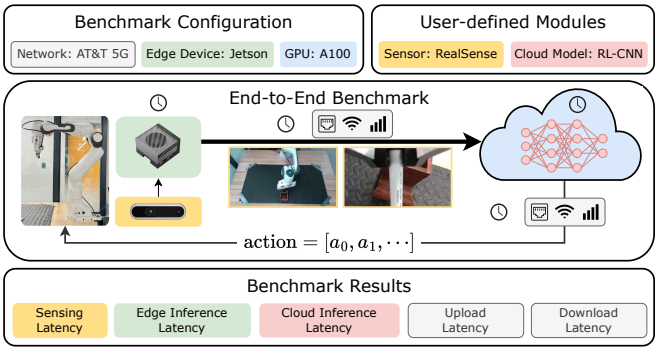

Fig. 1: **PEERNet is an end-to-end profiling tool for networked robotics.** Our modular tool allows users to inject custom external modules and specify custom configurations of sensors, networks, deep-learning pipelines and devices. Detailed profiles of latency, complete with asymmetric network timing, are generated.

Answering such questions requires a granular understanding of the performance of individual system components, from sensor data rates to Machine Learning (ML) inference latency to network transmission delay. This practical engineering use case shows that real-time system-specific performance profiling is essential for the efficient deployment of modern networked robotics.

Conventional methods for profiling networked robotic systems fall into two categories: isolated benchmarking and end-to-end system benchmarking. Existing end-to-end system benchmarks [11], [12] assess components within fully assembled systems, but frequently overlook networking, especially in asymmetric networks where upload and download latencies differ. These benchmarks generally fail to present a holistic software platform for modular profiling.

For these reasons, we introduce the **P**rofiler for **E**nd-to-**E**nd **R**eal-time **Net**worked robotic systems (PEERNet), a novel end-to-end profiling tool designed for real-time networked robotic systems. By modeling arbitrary networked robotic systems as combinations of sensors, networks, and computation, our method is capable of detailed profiling of arbitrary user-defined systems. This is the first tool that integrates with industry-standard products such as Nvidia single board computers and Robot Operating System (ROS), allowing users to systematically benchmark all aspects of a cloud or edge robotics system. As shown in Figure 1, our system emphasizes end-to-end profiling of the entire flow of data from sensing to upload, remote inference, and download, allowing us to profile at a level of detail not possible with

isolated benchmarks.

In summary, our contributions are:

1) **End-to-End Networked Robotics Benchmarking:** We present PEERNet, a Python framework for end-to-end benchmarking of networked robotics systems, profiling data flow and system performance.
2) **Enhanced Modularity and Accessibility:** PEERNet boasts a high degree of modularity and accessibility, facilitated by highly modular implementation and a comprehensive Command-Line Interface (CLI) which allows users to define custom networked robotics setups to profile, even allowing users to pass external programs into the benchmarking framework.
3) **One-way Delay Estimation:** Catered towards networked robotics, PEERNet times one-way network delay using Network Time Protocol (NTP) synchronization. Compared to round-trip time measurements, PEERNet provides specificity in the data collected, especially those of networked robotics systems operating in asymmetric networks. For example, our experiments show that the latency variations between upload and download are notable because edge devices upload considerably more data than they download.
4) **Deployment Examples on Real Systems:** We explore three distinct robotics scenarios encompassing offloaded inference and teleoperated robotics. Leveraging the capabilities of PEERNet, we conduct comprehensive benchmarking to assess performance across various implementations of networked robotics. Our findings underscore the end-to-end benchmarking capabilities of our framework, emphasizing its prowess in profiling, extensibility, and modularity.

II. Related Works

This section discusses progress towards benchmarking systems for networked robotics.

A. Standardized Tasks and Datasets

One approach to benchmarking in modern robotics is to develop standardized tasks and datasets to provide meaningful comparisons between robotic systems. RTRBench [13] provides standardized "kernels" for common robotic perception and planning tasks. In the context of networked robotics, IoTBench [14] outlines the requirements and difficulties of producing a comprehensive benchmark for networked systems. Live robotics profiling remains an open challenge, as existing approaches ignore frameworks for data collection.

B. Isolated Component Benchmarking

Other approaches focus on profiling individual components in robotic systems, *i.e.*, sensors or ML models. Baidu's DeepBench [15] provides a flexible framework for profiling operations in ML inference, and MLPerf [16] builds one of the largest databases of crowd-sourced ML inference benchmarks. Aside from the more prevalent benchmarking tools and datasets for ML tasks, recent works also provide datasets and data collection methods for hardware components, such as LiDAR [17]. While these isolated benchmarks can be comprehensive due to crowd-sourcing, they do not truly reflect the holistic performance of robotic systems, as they do not reflect data flow relationships seen in real hardware.

C. Network Delay Estimation

Estimating network delay is a crucial component in profiling networked robotic systems. Network delay estimation is well studied, both in the round-trip case [18] and the one-way delay case [19], [20]. In this work, we integrate one-way delay estimation using clock synchronization [21] into a networked robotics profiling framework.

D. System-level Benchmarking Tools

In recent years, many holistic system-level benchmarking tools have been developed. RobotPerf [11] is a holistic benchmarking framework on top of ROS2 [22], and PyRobot [23] provides a high-level interface to facilitate research and benchmarking with ROS systems. LEAF [12], a software package tailored to benchmark federated learning, closely approaches networked robotic benchmarking by considering multi-robot systems and the effects of networking. Despite a system-level holistic view of robotic systems, these tools fail to provide a general solution for networked robotics benchmarking by ignoring network latency and being constrained to particular hardware or software stacks.

FogRos [24], [25] builds a lightweight framework for easy deployment of ROS applications to cloud architectures. Similarly, PEERNet allows interfacing with standard software stacks such as ROS and ROS2, but aims to add profiling capabilities to cloud robotic systems by considering diverse network types and arbitrary components.

E. Teleoperation Latency Studies

In robotic teleoperation, prior works have quantified network latency and its effects on teleoperation task performance [26], [27]. However, these methods focus on particular cases of networked robotic systems rather than building tools for general and modular networked robotics.

In conclusion, existing methods and tools for robotic system benchmarking do not fully address the problem of networked robotics profiling. Although standardized tasks compare individual systems, they lack insight into the system's implementations. Isolated component benchmarks are easy to use and comprehensive but fail to reflect the combined performance of completed systems. Network profiling, including one-way delay estimation, is well documented, but prior work has not integrated it into the benchmarking of networked robotics.

III. A Motivating Example

To motivate the need for PEERNet, we present a practical networked robotics example in which end-to-end and real-time benchmarking is essential. The example includes common functional components in networked robotic systems, *e.g.*, sensors, networks, and devices with computing capabilities, which can all be profiled by PEERNet.

Consider an automated factory where quality control is accomplished using computer vision models trained to identify defects in parts on a production line. The pipeline for such a system is as follows: A camera (sensor) samples images from videos from an assembly line. Due to computing constraints, these images are either processed locally on the robot arm's (devices) low-level controller or offloaded over the Internet (network) to cloud servers such as Amazon Web Services machines (devices). In the cloud, images are classified using computationally expensive vision models, and labels are transmitted back to the factory. Finally, the robot arm can discard defective parts.

To deploy offloading algorithms minimizing end-to-end latency of such systems, the following must be known:

- Data rates of sensors: the frequency at which images of the assembly line can be sampled.
- Local inference latency: the time taken to classify an image as defective or not on the local device.
- Network upload latency: the time taken to transmit images to the cloud.
- Cloud inference latency: the time taken to classify an image as defective or not on the cloud.
- Network download latency: the time taken to transmit labels back to the robot arms.
- Robot arm latency: the time taken for the robot arms to discard a sample.

Complete knowledge of these metrics in real-time enables optimization of this motivating example and more general networked robotic systems. PEERNet presents a modular and easy-to-use software stack to collect these metrics.

IV. Implementation of PEERNet

In Section III, we provide an example of a networked robotic system that requires real-time profiling to optimize. We now present the implementation of the main modules of PEERNet, closely following that motivating example.

A. Functional Modules

PEERNet consists of four major functional modules that closely mirror the components of the networked robotic example presented in Section III. Our software stack provides sample implementations of each module but more importantly, a framework for users to customize.

1) The **Sensors** module abstracts input data streams to edge devices, defining sensors as any hardware or software from which a program can sample data. It includes traditional sensors such as cameras and Inertial Measurement Units (IMUs) and non-traditional input streams such as datasets or human input.
2) The **Networks** module abstracts all communication between devices. Our base software stack contains three network implementations frequently used in networked robotic systems: a Transmission Control Protocol (TCP) implementation built through ZeroMQ (a lightweight messaging library), a User Datagram Protocol (UDP) implementation built through ZeroMQ, and a wrapper for networking in ROS [28]. The network module also contains a set of utilities for transmitting and receiving metadata that enables one-way network profiling with time-synced devices.
3) The **Inference** module abstracts computation on local and cloud devices. For example, implementations of the inference module encapsulate object detection, image classification, and Large Language Model (LLM) queries. However, the framework we provide is general and encapsulates general computing tasks.
4) The **Logging** module is responsible for organizing and maintaining the data collected during end-to-end benchmarks. The core component of the logging module is the logger, a tree-like data structure with serializable nodes for tracking iterative and nested components of a benchmark. Nodes in a logger represent events during the life cycle of a benchmark. We implement a data-typing system for nodes, with nodes typed based on the metric they collect, such as latency or throughput. The logger provides an interface for users to define custom node types to represent custom metrics, such as power consumption or temperature measurements. As detailed in Section IV-B.1, the serializable and recursive structure of this data structure enables concise profiling of one-way transmission time among time-synced devices.

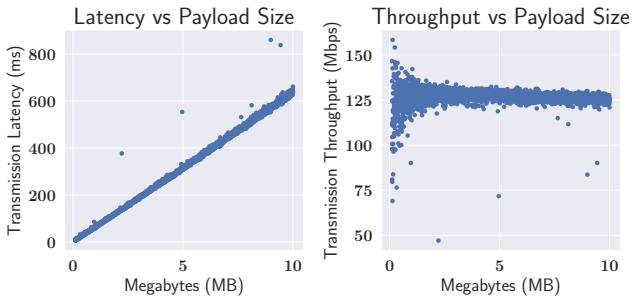

Fig. 2: **PEERNet's one-way delay measurements incur minimal error, of** 2.01**ms.** Latency estimates exhibit a constant offset error, measured here to be -2.01ms, and throughput estimates show decreasing variance for higher payload sizes, in the scale of images.

B. Implementation Details

We now present how PEERNet composes the four functional modules described in Section IV-A to make an end-to-end benchmarking tool. We cover the network timing scheme used to profile one-way delay in Figure 2 and the implementation of a CLI to benchmark offloaded computation.

1) One-Way Network Profiling: In many networked robotic systems, devices exhibit asymmetric data transfer; edge devices upload much more data than they download, and servers see more incoming traffic [29]. In our smart factory example, images (on the scale of megabytes each) are uploaded to the cloud, but only labels (on the scale of kilobytes) are downloaded. Thus, PEERNet implements a one-way delay profiling scheme. Our implementation involves syncing device clocks on the sender and receiver using an NTP implementation such as Chrony, as seen in

Algorithm 1 PEERNet's Offloaded Inference Scheme

Definitions:
$\mathcal{S}$: A sensor
$\mathcal{D}$: A device that can sample data from $\mathcal{S}$
$\mathcal{M}$: An ML model
$\mathcal{C}$: A cloud device that can perform inference using $\mathcal{M}$
$\mathcal{N}$: A bi-directional network between $\mathcal{D}$ and $\mathcal{C}$

Offloaded Inference Scheme:
while not done **do**
    $\mathcal{D}$ samples data x from $\mathcal{S}$
    $\mathcal{D}$ uploads x to $\mathcal{C}$ over $\mathcal{N}$
    $\mathcal{C}$ performs inference, obtaining $y = \mathcal{M}(x)$
    $\mathcal{C}$ transmits y to $\mathcal{D}$ over $\mathcal{N}$
end while

[21]. With synced device clocks, we exploit the serializable structure of the nodes in our logger to augment all messages sent across networks with logging metadata. Devices on the receiving end can deserialize attached metadata and interact with logger objects. This allows PEERNet to start the timing on a sending device before a message is sent and to end the timing on a receiving device after a message is received.

Measuring one-way delay using synchronized clocks incurs some error due to imperfect clock synchronization. However, we quantify the total error in one-way delay estimates and study its propagation to estimates of other network-related metrics such as throughput. Figure 2 shows the results of an introspective study, where we observe the latency and throughput of various payload sizes. As the ideal relationship between latency and payload size is linear with the intercept 0, we calculate the intercept of a line of best fit to estimate the total one-way delay estimation error, which is -2.01 ms. The negative intercept indicates that the sending device's clock is faster than the receiver's. The ratio between throughput and payload size is ideally constant. We show that for sufficiently large payloads such as images, it holds. For small payloads, the throughput estimates are affected by small offsets in the latency estimates.

2) CLI for Offloaded Inference: Many networked robotics systems, including our smart factory example, fall into the category of offloaded computation or offloaded inference [6], [7], [8], [9], [10]. In these systems, edge devices selectively offload computation to cloud servers to conserve computing resources and minimize latency. PEERNet provides an implementation that combines its functional modules into an offloaded edge robotic system (Algorithm 1) and exposes a CLI, enabling users to define custom sensors, networks, and computing programs. The CLI is highly modular, allowing for custom selection of sensors, networks, and models, even allowing users to pass custom implementations of sensors and ML models from a single terminal command.

The pipeline for our offloaded inference implementation (Algorithm 1) is as follows. First, users specify on the command line from an edge device $\mathcal{D}$, a cloud device $\mathcal{C}$, a sensor $\mathcal{S}$, a network type $\mathcal{N}$, and a cloud model $\mathcal{M}$. The edge device samples a datapoint x from the sensor, and uploads x to $\mathcal{C}$ over $\mathcal{N}$. Cloud device $\mathcal{C}$ performs inference with model $\mathcal{M}$, and returns the output y to edge device $\mathcal{D}$ over network $\mathcal{N}$. During the process, PEERNet's logging module is used to profile every operation indicated in algorithm 1 such as sampling from a sensor and one-way network transmission. Appendix A contains a full list of metrics collected and examples of CLI usage.

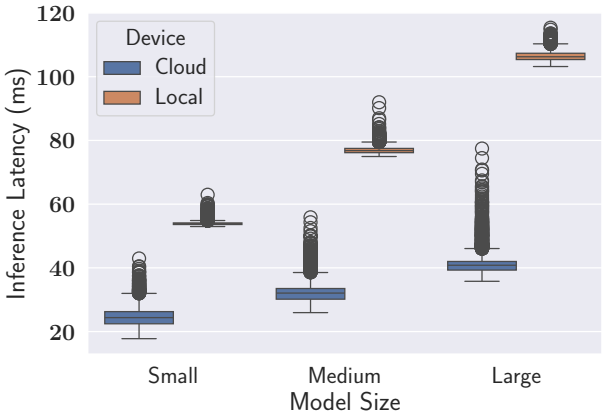

Fig. 3: **PEERNet precisely quantifies inference costs at the edge and in the cloud.** For the EfficientNetV2 family of models, local computation on an edge device is roughly 2.5 times as slow as offloaded computation to a cloud server, but cloud servers display a high variance in inference latency.

In summary, PEERNet is implemented around four core functional modules: sensors, networks, inference, and logging. It provides utilities for profiling one-way delay across networks using NTP for clock syncing and implements a CLI tailored to benchmark offloaded inference systems.

V. Experimental Validation

We validate PEERNet by implementing and profiling three networked robotic systems. We demonstrate that PEERNet is modular, robust, and capable of precisely profiling various systems. All of our experiments are conducted on physical hardware and live wireless networks. Our first experiment, offloaded image classification, demonstrates our profiler's generalizability to common networked robotics systems. Our second experiment demonstrates PEERNet's profiling of complex behavior in LLM inference. Finally, our third experiment, a teleoperation example on a Franka Emika Panda robot arm, demonstrates PEERNet's ability to profile arbitrary systems in a modular fashion.

A. Offloaded Image Classification with EfficientNetV2

Consider a self-driving car performing image classification in real-time on a video stream. For a resource-constrained edge device, offloading image classification to cloud servers is a viable solution. Suppose further that our self-driving car has access to a bank of image classification models, each with varying model complexities. To develop methods to select which image classifier from the bank to run, data on

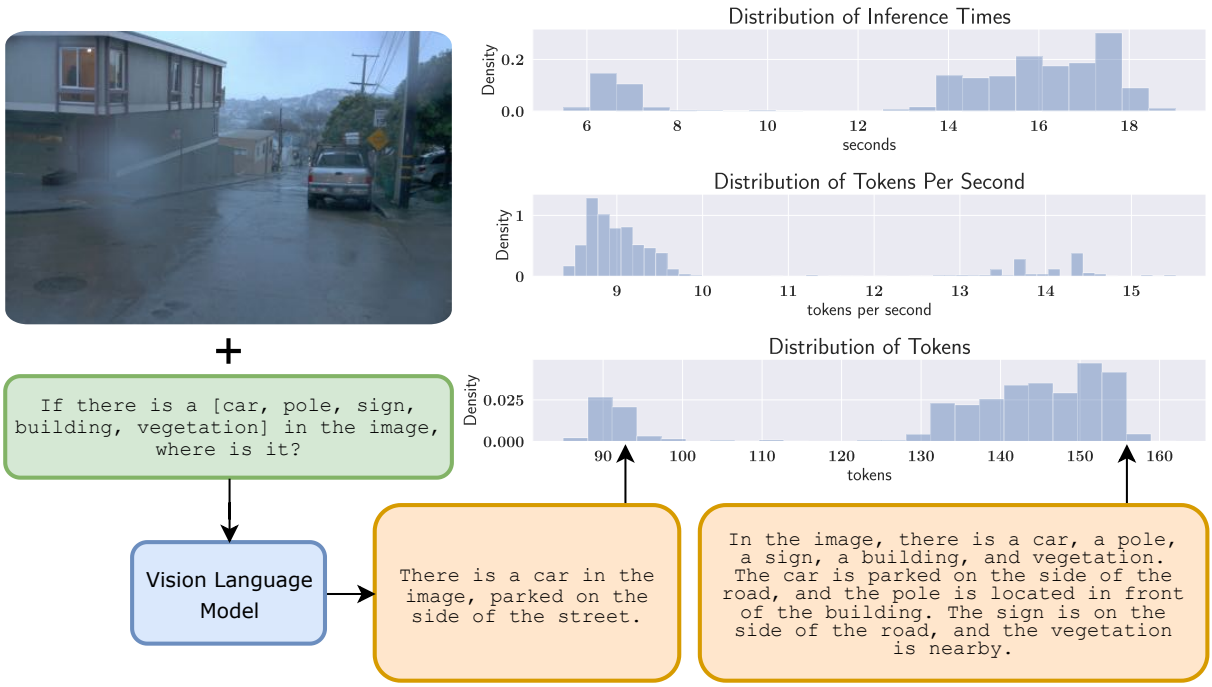

Fig. 4: **Profiling with PEERNet reveals non-intuitive behavior of a vision language model.** Responses to a single prompt vary in length, with two centers in the distribution of output length. Consequently, the inference latency is bimodal.

the inference latency of each model running locally and on cloud servers is essential.

We implement such an offloaded image classification system with the following parameters. The edge device is an Nvidia Jetson AGX Orin 64Gb, a popular single-board computer for deep learning on the edge. The bank of models is the EfficientNetV2 family of image classifiers [30], trained on the ImageNet dataset [31], which includes three model variants: small, medium, and large. A remote server with an Nvidia A5000 GPU plays the role of a cloud server. The remote server is configured as an NTP server, and the Nvidia Jetson is configured as an NTP client. As this is a case of offloaded computation, the experiment is fully specified through PEERNet's CLI for offloaded inference. We specify a network type, provide domain names and IP addresses for each device, and specify a model to be run, either locally or in the cloud. Appendix B contains the precise terminal commands issued to reproduce our offloaded image classification experiment.

Figure 3 shows the performance of each variant of EfficientNetV2 when run locally and on a cloud server. PEERNet's profiling enables us to precisely quantify the latency of each variant of EfficientNetV2 locally and on the cloud server. We see that each variant running locally is roughly 2.5 times slower than the same model running in the cloud. A more subtle observation from the profiling is the high variance in inference latency in the cloud compared to the low variance in inference latency locally. This observation is consistent with the fact that cloud servers have many users, while edge devices rarely run many parallel processes. Thus, the variance in inference latency in the cloud is large compared to the variance in latency on the edge. In summary, this experiment demonstrates the powerful abstraction of offloaded computation as a framework to benchmark in and validates PEERNet's CLI as a modular and easy-to-use method to obtain precise profiling data on offloaded computation systems.

B. LLM Inference at the Edge

Recently, the robotics community has begun to adopt LLMs in robotic pipelines. These models provide reasoning capabilities, performing tasks such as planning [32], [33]. Furthermore, multi-modal LLMs such as LLaVA [34] are particularly suited to robotics applications due to their ability to reason about images and video. Due to the size of modern LLMs, many applications adopt a cloud inference model, where LLMs are served over wireless networks. However, with advancing edge hardware and powerful GPUs at the edge, the feasibility of LLM inference at the edge is an open question.

In this experiment, we use PEERNet to profile the performance of LLaVA [34], a multi-modal LLM capable of image reasoning on edge inference. We implement and benchmark the edge-inference system using PEERNet's utilities, and demonstrate the ability to obtain complex information about model performance. In our experimental setup, the edge device is an Nvidia Jetson AGX Orin 64Gb. The multi-modal LLM we use is LLaVA-v1.5-7b [35]. We focus our profiling on understanding the output length of LLM responses and the relationship between output length and inference latency.

Figure 4 presents the collected data on the performance of LLM inference at the edge, along with a test prompt and image, from the Waymo Open Dataset [36]. Benchmarking with PEERNet reveals that for a particular image and prompt, LLaVA-v1.5's output length is strongly bimodal. Further-

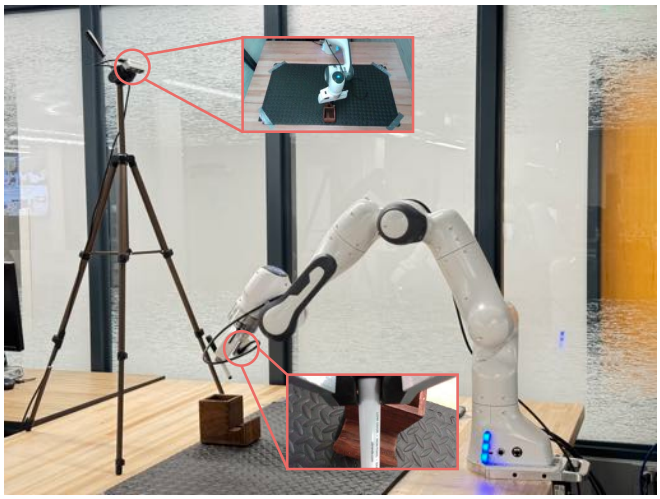

Fig. 5: **Physical setup for Teleoperation.** The objective of teleoperation is for the robot arm to manipulate the pen into the wooden cup, using visual information from two cameras, one on the arm, and another on a tripod.

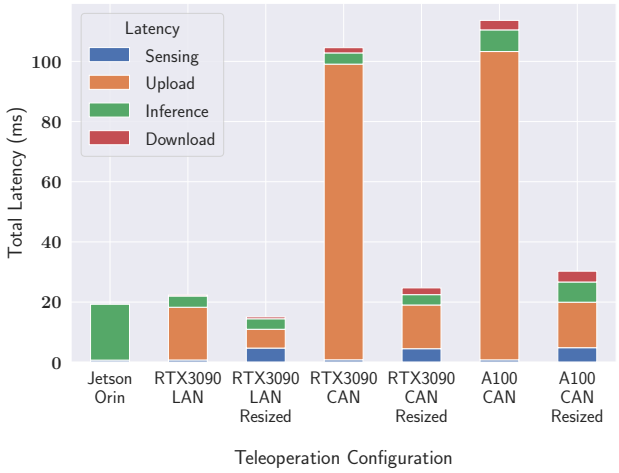

Fig. 6: **Profiling with PEERNet reveals latency tradeoffs and costs in end-to-end teleoperation pipelines.** PEERNet identifies the combination of an RTX3090 GPU, local image resizing, and a LAN to be the most performative teleoperation configuration, with lower end-to-end latency than local inference.

more, we note that our edge device can produce between 8 and 15 tokens per second, corresponding to between 5 and 20 seconds per inference with a particular prompt.

C. Teleoperation of a Robotic Arm

Teleoperation, in which a physical robot is controlled remotely by another machine or human, is a prevalent use case of networked robotics, with implementations in self-driving [37], [38], augmented reality [39], and telesurgery [40], [41]. We design, implement, and benchmark a teleoperation system for a Franka Emika Panda robot arm to demonstrate how PEERNet can be used to profile complex teleoperation systems.

We implement and profile a teleoperation pipeline of vision-based robot manipulation. In this task, the manipulator must start from its initial configuration 50cm above the table and manipulate a pen into a square cup (7.5cm x 7.5cm), using image observations from two camera angles. We outfit a Franka Emika Panda robotic arm with two RealSense cameras, one mounted on the end effector and one mounted on a tripod (Figure 5). We employ a CNN-based reinforcement learning policy to ensure consistency in task performance [42].

We consider several different configurations to accomplish our teleoperation task, using different GPUs, networks, and image compression. We profile each configuration with PEERNet to compare end-to-end performance and demonstrate how PEERNet builds an understanding of sources of latency and their effects on end-to-end latency. All teleoperation configurations follow a similar pipeline. An edge device (Nvidia Jetson Orin 64GB) samples images from the two RealSense cameras. Based on the configuration, the edge device may downsample images to lower the resolution and size, which we refer to as resizing. The edge device uploads images to a chosen cloud device over a chosen network type, where inference is performed with the CNN-based controller. Actions are downloaded back to the edge, where the robot arm executes them. This process is repeated until the episode terminates after a fixed number of steps. For each configuration, we profile 30 episodes, each with 30 steps until termination. We discard the first episode and the first step from each episode to account for GPU warm-up.

We consider three types of local and cloud devices:

1) **Jetson**: For a *local* configuration, the Jetson edge device performs inference locally using the GPU onboard.
2) **RTX3090**: A desktop machine with an Nvidia RTX3090 GPU.
3) **A100**: A remote server with an Nvidia A100 GPU.

We consider three types of networks:

1) **Local Area Network (LAN)**: Devices are connected via Ethernet to a network switch.
2) **Campus Area Network (CAN)**: Devices are connected via WiFi to a campus-wide network.
3) **AT&T 5G**: Devices are connected to the internet via an AT&T 5G mobile hotspot, and connected to a CAN through a Virtual Private Network (VPN).

Figure 6 shows component and end-to-end latency for LAN and CAN configurations. PEERNet reveals several tradeoffs at play. First, a tradeoff between inference latency and network latency–remote devices show lower inference latency than the edge device, but uploading images comes with additional latency due to networks. Second, a tradeoff between network latency and sampling latency–when resizing images at the edge, network latency is reduced due to smaller payload size, but, sampling time increases due to the latency of image resizing. We also note an asymmetry in network delay across all teleoperation configurations. As images are uploaded but actions are downloaded, upload latency contributes significantly to end-to-end latency, while download latency is negligible. Based on PEERNet's profiling, we see that the configuration that minimizes end-to-end latency is to

Configuration			Latency (mean ± std in ms)				
Resize Images	GPU	Network	Sensing	Upload	Inference	Download	Total
✗	Jetson Orin	✗	0.71 ± 0.26	✗	18.55 ± 3.34	✗	19.26 ± 3.34
✗	RTX 3090	LAN	0.76 ± 0.25	17.53 ± 1.72	3.73 ± 1.12	0.37 ± 0.12	22.38 ± 2.07
✓	RTX 3090	LAN	4.71 ± 1.08	6.25 ± 0.99	3.48 ± 1.11	0.62 ± 0.11	15.07 ± 1.84
✗	RTX 3090	CAN	0.81 ± 0.28	98.28 ± 11.33	3.73 ± 1.53	1.73 ± 1.81	104.55 ± 11.58
✓	RTX 3090	CAN	4.50 ± 1.03	14.53 ± 10.13	3.43 ± 1.52	2.24 ± 6.05	24.71 ± 11.94
✗	A100	CAN	0.79 ± 0.27	102.48 ± 26.40	7.19 ± 3.38	3.16 ± 8.95	113.62 ± 28.08
✓	A100	CAN	4.83 ± 1.08	15.13 ± 7.63	6.71 ± 3.05	3.59 ± 4.83	30.26 ± 9.59
✓	RTX 3090	AT&T 5G	3.03 ± 0.58	432.71 ± 34.14	3.05 ± 1.35	228.84 ± 149.83	667.63 ± 153.67
✓	A100	AT&T 5G	3.14 ± 0.48	339.20 ± 234.52	4.56 ± 15.68	128.04 ± 108.65	474.94 ± 258.94

TABLE I: **PEERNet's profiling of teleoperation configurations for a robotic manipulation task.** PEERNet quantifies the tradeoffs between network latency and inference cost, disambiguates upload and download latency, and identifies the most performative setups.

perform inference on an RTX3090 GPU connected via LAN, and to resize images before transmission. Table I shows the details of all the tested configurations, including those with an AT&T mobile 5G network. We note that the AT&T 5G configuration incurs extreme network latency, with end-to-end latency up to 4 times worse than those on a CAN.

In summary, we design, implement, and profile three teleoperation configurations. We show that PEERNet collects specific data on important steps in a teleoperation pipeline and that profiling using PEERNet can be used to identify optimal choices for configuring cloud robotic setups.

VI. Discussion and Conclusion

We present PEERNet, an end-to-end and real-time benchmarking suite for networked robotics. Our package is easy to use, interfaces with industry-standard hardware and software, and pays special attention to profiling one-way delays in wireless networks. We demonstrate the modularity and scope of PEERNet's profiling abilities through the exploration of two robotic offloading experiments and a teleoperation experiment on a Franka Emika Panda arm. We believe PEERNet is a significant step towards the efficient deployment of optimized networked robotics.

VII. Acknowledgement

This work was supported in part by the Lockheed Martin Corporation, in part by the National Science Foundation under grant 2148186, and in part by federal agencies and industry partners as specified in the Resilient and Intelligent NextG Systems (RINGS) Program.

References

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

[2] Y. Mao, C. You, J. Zhang, K. Huang, and K. B. Letaief, "A survey on mobile edge computing: The communication perspective," *IEEE communications surveys & tutorials*, vol. 19, no. 4, pp. 2322–2358, 2017.

[3] S. Kumar, S. Gollakota, and D. Katabi, "A cloud-assisted design for autonomous driving," in *Proceedings of the first edition of the MCC workshop on Mobile cloud computing*, 2012.

[4] G. Asaamoning, P. Mendes, D. do Rosário, and E. Cerqueira, "Drone swarms as networked control systems by integration of networking and computing," *Sensors*, vol. 21, 2021.

[5] P. Berkelman and J. Ma, "A compact modular teleoperated robotic system for laparoscopic surgery," *The International journal of robotics research*, vol. 28, no. 9, pp. 1198–1215, 2009.

[6] S. Chinchali, 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," *Autonomous Robots*, vol. 45, no. 7, pp. 997–1012, 2021.

[7] B. Ghosh, M. Khan, A. Ashok, S. Chinchali, and P. S. Duggirala, "Dynamic selection of perception models for robotic control," 2022.

[8] P. han Li, O. S. Toprak, A. Narayanan, U. Topcu, and S. Chinchali, "Online foundation model selection in robotics," 2024.

[9] H. Lin, S. Zeadally, Z. Chen, H. Labiod, and L. Wang, "A survey on computation offloading modeling for edge computing," *Journal of Network and Computer Applications*, vol. 169, p. 102781, 2020.

[10] S. S. Narasimhan, S. Bhat, and S. P. Chinchali, "Safe networked robotics with probabilistic verification," *IEEE Robotics and Automation Letters*, vol. 9, no. 3, pp. 2917–2924, 2024.

[11] V. Mayoral-Vilches, J. Jabbour, Y.-S. Hsiao, Z. Wan, M. Crespo-Álvarez, M. Stewart, J. M. Reina-Muñoz, P. Nagras, G. Vikhe, M. Bakhshalipour, M. Pinzger, S. Rass, S. Panigrahi, G. Corradi, N. Roy, P. B. Gibbons, S. M. Neuman, B. Plancher, and V. J. Reddi, "Robotperf: An open-source, vendor-agnostic, benchmarking suite for evaluating robotics computing system performance," 2024.

[12] S. Caldas, S. M. K. Duddu, P. Wu, T. Li, J. Konečný, H. B. McMahan, V. Smith, and A. Talwalkar, "Leaf: A benchmark for federated settings," 2019.

[13] M. Bakhshalipour, M. Likhachev, and P. B. Gibbons, "Rtrbench: A benchmark suite for real-time robotics," in *IEEE International Symposium on Performance Analysis of Systems and Software (ISPASS)*, 2022.

[14] C. A. Boano, S. Duquennoy, A. Förster, O. Gnawali, R. Jacob, H.-S. Kim, O. Landsiedel, R. Marfievici, L. Mottola, G. P. Picco, X. Vilajosana, T. Watteyne, and M. Zimmerling, "Iotbench: Towards a benchmark for low-power wireless networking," in *IEEE Workshop on Benchmarking Cyber-Physical Networks and Systems (CPSBench)*, 2018.

[15] "Deepbench." `https://github.com/baidu-research/DeepBench`, 2020.

[16] V. J. Reddi, C. Cheng, D. Kanter, P. Mattson, G. Schmuelling, C.-J. Wu, B. Anderson, M. Breughe, M. Charlebois, W. Chou, *et al.*, "Mlperf inference benchmark," in *ACM/IEEE 47th Annual International Symposium on Computer Architecture (ISCA)*, 2020.

[17] F. Schalling, S. Ljungberg, and N. Mohan, "Benchmarking lidar sensors for development and evaluation of automotive perception," in *4th International Conference and Workshops on Recent Advances and Innovations in Engineering (ICRAIE)*, 2019.

[18] A. Acharya and J. Saltz, *A study of internet round-trip delay*. Digital Repository at the University of Maryland, 1998.

[19] L. De Vito, S. Rapuano, and L. Tomaciello, "One-way delay measurement: State of the art," *IEEE Transactions on Instrumentation and Measurement*, vol. 57, no. 12, pp. 2742–2750, 2008.

[20] J.-H. Choi and C. Yoo, "One-way delay estimation and its application," *Computer Communications*, vol. 28, no. 7, pp. 819–828, 2005.

[21] V. Smotlacha *et al.*, "One-way delay measurement using ntp synchronization," in *Proc. of TERENA Networking Conf*, 2003.

[22] S. Macenski, T. Foote, B. Gerkey, C. Lalancette, and W. Woodall, "Robot operating system 2: Design, architecture, and uses in the wild," *Science Robotics*, vol. 7, no. 66, 2022.

[23] A. Murali, T. Chen, K. V. Alwala, D. Gandhi, L. Pinto, S. Gupta, and A. Gupta, "Pyrobot: An open-source robotics framework for research and benchmarking," 2019.

[24] K. E. Chen, Y. Liang, N. Jha, J. Ichnowski, M. Danielczuk, J. Gonzalez, J. Kubiatowicz, and K. Goldberg, "Fogros: An adaptive framework

for automating fog robotics deployment," in *IEEE 17th International Conference on Automation Science and Engineering (CASE)*, 2021.
[25] J. Ichnowski, K. Chen, K. Dharmarajan, S. Adebola, M. Danielczuk, V. Mayoral-Vilches, N. Jha, H. Zhan, E. Llontop, D. Xu, *et al.*, "Fogros2: An adaptive platform for cloud and fog robotics using ros 2," in *IEEE International Conference on Robotics and Automation (ICRA)*, 2023.
[26] M. J. Lum, J. Rosen, H. King, D. C. Friedman, T. S. Lendvay, A. S. Wright, M. N. Sinanan, and B. Hannaford, "Teleoperation in surgical robotics – network latency effects on surgical performance," in *Annual International Conference of the IEEE Engineering in Medicine and Biology Society*, 2009.
[27] A. Noguera Cundar, R. Fotouhi, Z. Ochitwa, and H. Obaid, "Quantifying the effects of network latency for a teleoperated robot," *Sensors*, vol. 23, no. 20, 2023.
[28] M. Quigley, K. Conley, B. Gerkey, J. Faust, T. Foote, J. Leibs, R. Wheeler, A. Y. Ng, *et al.*, "Ros: an open-source robot operating system," in *ICRA workshop on open source software*, 2009.
[29] H. Balakrishnan, V. N. Padmanabhan, and R. H. Katz, "The effects of asymmetry on tcp performance," in *Proceedings of the 3rd annual ACM/IEEE international conference on Mobile computing and networking*, pp. 77–89, 1997.
[30] M. Tan and Q. Le, "Efficientnetv2: Smaller models and faster training," in *International conference on machine learning*, 2021.
[31] J. Deng, W. Dong, R. Socher, L.-J. Li, K. Li, and L. Fei-Fei, "Imagenet: A large-scale hierarchical image database," in *IEEE conference on computer vision and pattern recognition*, 2009.
[32] S. Sharan, R. Zhao, Z. Wang, S. P. Chinchali, *et al.*, "Plan diffuser: Grounding llm planners with diffusion models for robotic manipulation," in *Bridging the Gap between Cognitive Science and Robot Learning in the Real World: Progresses and New Directions*, 2024.
[33] S. Vemprala, R. Bonatti, A. Bucker, and A. Kapoor, "Chatgpt for robotics: Design principles and model abilities," *Microsoft Auton. Syst. Robot. Res*, vol. 2, p. 20, 2023.
[34] H. Liu, C. Li, Q. Wu, and Y. J. Lee, "Visual instruction tuning," *Advances in neural information processing systems*, 2024.
[35] H. Liu, C. Li, Y. Li, and Y. J. Lee, "Improved baselines with visual instruction tuning," 2023.
[36] P. Sun, H. Kretzschmar, X. Dotiwalla, A. Chouard, V. Patnaik, P. Tsui, J. Guo, Y. Zhou, Y. Chai, B. Caine, V. Vasudevan, W. Han, J. Ngiam, H. Zhao, A. Timofeev, S. Ettinger, M. Krivokon, A. Gao, A. Joshi, Y. Zhang, J. Shlens, Z. Chen, and D. Anguelov, "Scalability in perception for autonomous driving: Waymo open dataset," in *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition (CVPR)*, 2020.
[37] F. Tener and J. Lanir, "Driving from a distance: challenges and guidelines for autonomous vehicle teleoperation interfaces," in *Proceedings of the 2022 CHI Conference on Human Factors in Computing Systems*, pp. 1–13, 2022.
[38] T. Zhang, "Toward automated vehicle teleoperation: Vision, opportunities, and challenges," *IEEE Internet of Things Journal*, vol. 7, no. 12, pp. 11347–11354, 2020.
[39] M. E. Walker, H. Hedayati, and D. Szafir, "Robot teleoperation with augmented reality virtual surrogates," in *14th ACM/IEEE International Conference on Human-Robot Interaction (HRI)*, 2019.
[40] S. Sedaghat and A. H. Jahangir, "Rt-telsurg: Real time telesurgery using sdn, fog, and cloud as infrastructures," *IEEE Access*, vol. 9, pp. 52238–52251, 2021.
[41] P. Barba, J. Stramiello, E. K. Funk, F. Richter, M. C. Yip, and R. K. Orosco, "Remote telesurgery in humans: a systematic review," *Surgical endoscopy*, vol. 36, no. 5, pp. 2771–2777, 2022.
[42] R. Zhao, U. Topcu, S. Chinchali, and M. Phielipp, "Learning sparse control tasks from pixels by latent nearest-neighbor-guided explorations," 2023.

APPENDIX

A. Tabulated Metrics Collected by Offloading CLI

Table II shows a list of metrics collected by the CLI for offloaded inference (Algorithm 1) by default.

Metric	Description
Sensing Time	Time taken for a sensor $\mathcal{S}$ to return sample x
Upload Size	Size of serialized x to be uploaded to $\mathcal{C}$
Upload Latency	Time taken for $\mathcal{D}$ to transmit x to $\mathcal{S}$
Upload Throughput	Observed throughput on uploading x to $\mathcal{S}$
Inference Latency	Time taken for $\mathcal{S}$ to perform inference with $\mathcal{M}$
Download Size	Size of serialized y to be transmitted to $\mathcal{D}$
Download Latency	Time taken for $\mathcal{S}$ to transmit y to $\mathcal{D}$
Download Throughput	Observed throughput on transmitting y to $\mathcal{D}$

TABLE II: CLI Options and Descriptions

B. Terminal Commands for Offloaded Image Classification

Below is an example of a terminal command for the offloading inference CLI. These commands reproduce the experiment with the small variant of EfficientNetV2, running between a server and a client. Table III gives descriptions of the commands.

```
PEERNet --server --name cluster
        --network zmq-tcp
        --network-config net_config.yaml
        --model-name efficientnet_v2_s
        --device cuda:6
        --iterations 10005

PEERNet --client --name jetson-orin
        --network zmq-tcp
        --network-config net_config.yaml
        --dataset-loc sample-image.JPEG
        --iterations 10005
        --result-loc results
        --generate-plots
```

Option	Description
`--server/--client`	Indicate device role.
`--name`	Device name in network configuration.
`--network`	[zmq-tcp | zmq-udp | ros] Specify the network type.
`--network-config`	Path to the network configuration.
`--sensor-type`	Specify sensor.
`--dataset-loc`	Location of the dataset.
`--sensor-object`	Location of the sensor object.
`--iterations`	Number of trials.
`--result-loc`	Output directory.
`--model-name`	ML model name.
`--device`	cuda / cpu.
`--generate-plots`	Create output files.

TABLE III: CLI Options and Descriptions