New Problems in Distributed Inference for DNN Models on Robotic IoT

(Anonymous Authors)

***Abstract*—The rapid advancements in machine learning (ML) techniques have led to significant achievements in various robotic tasks. Deploying these ML approaches on real-world robots requires fast and energy-efficient inference of their deep neural network (DNN) models. To our knowledge, distributed inference, which involves inference across multiple powerful GPU devices, has emerged as a promising optimization to improve inference performance in modern data centers. However, when deployed on real-world robots, existing parallel methods can not simultaneously meet the robots' latency and energy requirements and raise significant challenges.**

This paper reveals and evaluates the problems hindering the application of these parallel methods in robotic IoT, including the failure of data parallelism, the unacceptable communication overhead of tensor parallelism, and the significant transmission bottlenecks in pipeline parallelism. By raising awareness of these new problems, we aim to stimulate research toward finding a new parallel method to achieve fast and energy-efficient distributed inference in robotic IoT.

I. Introduction

The rapid progress in machine learning (ML) techniques has led to remarkable achievements in various fundamental robotic tasks, such as object detection[10], [19], [21], robotic control[14], [33], [40], and environmental perception[3], [15], [36]. However, deploying these ML applications on real-world robots requires fast and energy-efficient inference of their deep neural network (DNN) models, given the need for swift environmental responses and the limited battery capacity of robots. Placing the entire model on robots not only requires additional computing accelerators on robots (e.g., GPU[25], FPGA[26], SoC[7]), but also introduce additional energy consumption (e.g., 162% more for [21] in our experiments) due to the computationally intensive nature of DNN models, while placing the entire model in the cloud brings an extended response delay.

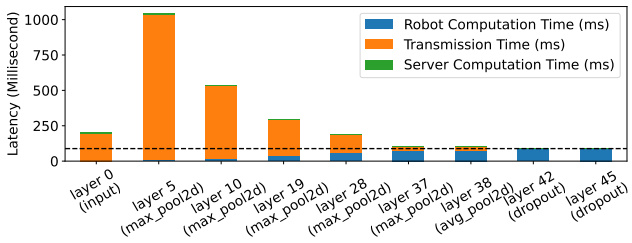
Distributed inference, which involves inference across multiple GPU devices, has emerged as a promising approach to meet the latency requirements of robotic applications and extend the battery lifetime of robots. This paradigm has been widely adopted in data centers[37], [41], [9], where numerous GPUs are utilized to speed large model inference, such as in the case of ChatGPT[35]. Adopting distributed inference across robots and other powerful GPU devices through the Internet of Things for these robots (robotic IoT) not only accelerates the inference process by leveraging the high computing capabilities of powerful GPUs but also alleviates the local computational burden, thereby reducing energy consumption, making it an ideal solution for robotic applications..

However, all existing parallel methods for distributed inference in the data center are ill-suited for robotic IoT. In data centers, there are mainly three kinds of parallel methods: Data parallelism (DP) replicates the model across devices, and lets each replica handle one mini-batch (i.e., a subset that slices out of an input data set); Tensor parallelism (TP) splits a single DNN layer over devices; Pipeline parallelism (PP) places different layers of a DNN model over devices (layer partitioning) and pipelines the inference to reduce devices' idling time (pipeline execution). In this paper, we demonstrate several issues that impede the application of existing parallel methods in robotic IoT.

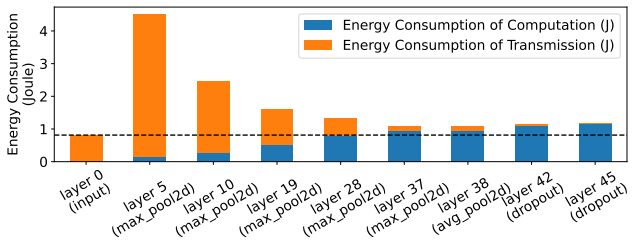
Problem 1 (DP). The small batch sizes inherent to robotic IoT applications (typically 1) hinder the mini-batch computation, rendering DP inapplicable for robotic IoT. In the data center, DP is feasible due to the large batch sizes employed (e.g., 16 images), allowing for the division of inputs into mini-batches that still contain several complete inputs (e.g., 2 images). However, in robotic IoT, real-time performance is crucial, necessitating immediate inference upon receiving inputs, which typically have smaller batch sizes (e.g., 1 image). Further splitting these inputs would result in mini-batches containing incomplete inputs (e.g., 1/4 of an image), which cannot be computed parallel to speed up inference.

Problem 2 (TP). TP requires frequent synchronization among devices, leading to unacceptable communication overhead in robotic IoT. By partitioning parameter tensors of a layer across GPUs, TP allows concurrent computation on different parts of this tensor but requires an all-reduce communication[41] to combine computation results from different devices, which entails significant communication overhead. Consequently, TP is used mainly for large layers that are too large to fit in one device in data centers and require dedicated high-speed interconnects (e.g., 400 Gbps for NVLink[13]) even within data centers. On the contrary, robots must prioritize seamless mobility and primarily depend on wireless connections, which inherently possess limited bandwidth, as described in Sec.II-A, making all-reduce synchronization an unacceptable overhead (e.g., commonly hundreds of milliseconds for each layer of VGG19[31] in our experiments).

Consequently, existing distributed inference approaches[16], [4] in robotic IoT primarily adopt the PP paradigm and focus on layer partitioning of PP, aiming to achieve fast and energy-efficient inference. This is because the PP paradigm in



(a) Inference Latency



(b) Energy consumption

Fig. 1: Our experiments on VGG19[31] reveal the comprehensive performance of various layer partitioning methods. The X-axis of the graph represents different layer partitioning scheduling schemes, where 'layer i' signifies that all layers up to and including the i-th layer are computed on the robot, while the subsequent layers are processed on the GPU devices. Note that different hardware conditions, network conditions and DNN model structure will lead to different performance, making this field an attractive area for a wide range of research.

data centers consists of layer partitioning and pipeline execution, where the pipeline execution of PP enhances inference throughput rather than reducing the completion time of a single inference[5], which is the most critical requirement in robotic IoT. Based on the fact that the amounts of output data in some intermediate layers of a DNN model are significantly smaller than that of its raw input data[8], DNN layer partitioning methods constitute various trade-offs between computation and transmission, taking into account application-specific inference speed requirements and energy consumption demands, as shown in Fig.1.

Problem 3 (PP). Existing methods based on PP face significant challenges due to transmission bottlenecks in robotic IoT, which are inherent to the PP's scheduling mechanism. PP is unable to overlap the transmission and computation phases within the same inference to alleviate the transmission overhead, as it can only overlap these phases across multiple inferences via pipeline execution, which increases inference throughput but not the completion time of a single inference[5]. Even with optimal layer partitioning from [16], [4], such transmission overhead inherent to PP's scheduling mechanism still becomes a substantial bottleneck due to the limited bandwidth of robotic IoT (e.g., up to 63% of inference time and 35% of energy consumption in our experiments).

In this paper, we take the first step to reveal and evaluate the problems hindering existing parallel methods for distributed inference applying to robotic IoT. These findings aim to raise research efforts to find a new parallel method to speed up distributed inference on robotic IoT so that the DNN models deployed on real-world robots can achieve fast and energy-efficient inference, and it will nurture diverse ML applications deployed on mobile robots in the field.

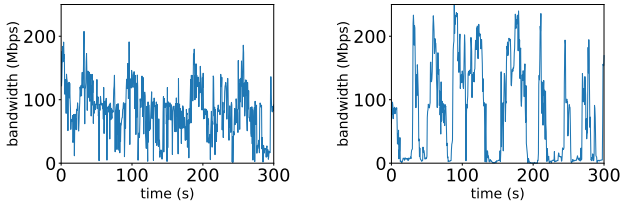
The rest of the paper is organized as follows: Sec.II introduces the characteristics of robotic IoT; Sec.III describes in detail about the problems on distributed inference on robotic IoT; Sec.IV provides with evaluation results; Sec.V concludes.

II. Characteristics of Robotic IoT

A. Wireless transmission of Robotic IoT

In real-world robotic IoT scenarios, devices often navigate and move around for tasks like search and exploration. While wireless networks provide high mobility, they also have limited bandwidth. For instance, Wi-Fi 6, the most advanced Wi-Fi technology, offers a maximum theoretical bandwidth of 1.2 Gbps for a single stream [18]. However, not only the limited hardware resources on the robot can not fully play the potential of Wi-Fi 6[39], but also the actual available bandwidth of wireless networks is often reduced in practice due to factors such as movement of the devices[20], [27], occlusion from by physical barriers[6], [30], and preemption of the wireless channel by other devices[2], [29].

To demonstrate the instability of wireless transmission in real-world situations, we conducted a robot surveillance experiment using four-wheel robots navigating around several given points at 5-40cm/s speed in our lab (indoors) and campus garden (outdoors), with hardware and wireless network settings as described in Sec. IV. We believe our setup represents robotic IoT devices' state-of-the-art computation and communication capabilities. We saturated the wireless network connection with iperf [1] and recorded the average bandwidth capacity between these robots every 0.1s for 5 minutes.



(a) Indoors (b) Outdoors

Fig. 2: The instability of wireless transmission in robotic IoT networks.

The results in Fig.2 show average bandwidth capacities of 93 Mbps and 73 Mbps for indoor and outdoor scenarios, respectively. The outdoor environment exhibited higher instability, with bandwidth frequently dropping to extremely low values around 0 Mbps, due to the lack of walls to

reflect wireless signals and the presence of obstacles like trees between communicating robots, resulting in fewer received signals compared to indoor environments.

In summary, robotic IoT systems' wireless transmission is constrained by limited bandwidth, both due to the theoretical upper limit of wireless transmission technologies and the practical instability of wireless networks.

B. Comparison with Datacenter Networks

Data center networks, used for large model inference (e.g., ChatGPT[35]), are wired and typically exhibit higher bandwidth capacity and lower fluctuation than robotic IoT networks. GPU devices in data centers are connected using high-speed networking technologies like InfiniBand[32] or PCIe[13], offering bandwidths from 40 Gbps to 500 Gbps. Bandwidth fluctuation in these networks is primarily caused by congestion on intermediate switches and can be mitigated through traffic scheduling on switches [24].

C. Existing distributed inference strategies in the data center

Data parallelism. Data parallelism [37] is a widely used technique in distributed inference that partitions input data across multiple devices, such as GPUs, to perform parallel inference. Each device maintains a complete model replica and independently processes a subset of the input data (mini-batch), aggregating results to generate the final output. Data parallelism enhances throughput by distributing workload across devices, leveraging their combined computational power.

However, data parallelism's scalability is constrained by the total batch size [23], which is particularly problematic in robotic IoT applications where smaller batch sizes are inherent due to the need for swift environmental responses. In robotic applications, immediate inference upon receiving inputs is crucial for obtaining real-time target points quickly. For example, in our experiments [21], the robot constantly obtains the latest images from the camera for inference, with a batch size of only 1. These small batches cannot be further split into mini-batches, a fundamental requirement for effective data parallelism.

Tensor parallelism. Tensor parallelism [41] is a distributed inference technique that divides a model's layer parameters across multiple devices, each storing and computing a portion of the weights. This approach requires an all-reduce communication step after each layer to combine results from different devices, introducing significant overhead, especially for large DNN layers. To mitigate this, TP is typically deployed across GPUs within the same server in data centers, using fast intra-server GPU-to-GPU links like NVLink [13], which is beneficial when the model is too large for a single device.

However, in robotic IoT, the limited bandwidth (Sec.II-A) makes TP's communication cost prohibitively high. Experiments on VGG19, officially provided by PyTorch[28], in our indoor scenario, showed that all-reduce communication for each layer takes hundreds of milliseconds, with the maximum layer in VGG19 taking an average of 1.05 seconds, more than ten times slower than computing the entire model locally, making TP impractical for robotic IoT.

Pipeline parallelism. Pipeline parallelism [9] is a distributed inference technique that partitions DNN model layers across multiple devices(layer partitioning), forming an inference pipeline for concurrent processing of multiple tasks. While PP can increase throughput and resource utilization, it primarily focuses on enhancing overall throughput rather than reducing single-inference latency[5], which is crucial in robotic IoT. As a result, existing distributed inference approaches [16], [4] in robotic IoT mainly concentrate on the layer partitioning aspect of PP, aiming to achieve fast and energy-efficient inference by optimizing the allocation of DNN layers across devices while considering factors such as device capabilities, network bandwidth, and energy consumption, as discussed further in Sec.III-A.

III. Problems in Existing Distributed Inference for DNN Models on Robotic IoT

A. Existing distributed inference on robotic IoT

Existing distributed inference approaches [16], [4] in robotic IoT primarily adopt the PP paradigm and focus on layer partitioning to achieve fast and energy-efficient inference. These approaches can be categorized into two main groups: accelerating inference for diverse DNN structures and optimizing robot energy consumption during inference.

For accelerating inference, earlier methods [8], [22], [11] focused on simple chain-like DNN models by exploiting smaller output data sizes of intermediate layers compared to raw input data [8], creating trade-offs between computation and transmission to minimize overall inference time (Fig. 1). However, the increasing complexity of DNN structures, now evolved into directed acyclic graphs (DAGs), poses new challenges, potentially leading to NP-hardness in performance optimization [16], which addressed by graph theory techniques [16], [38]. Varying hardware and network conditions further complicate the problem.

For optimizing energy consumption, existing methods [34], [17], [4] build upon the aforementioned techniques and consider reducing the system energy consumption of the entire layer partitioning execution process under deadline constraints. While [34] only considers transmission energy consumption, [17], [4] aim to reduce the whole system's energy consumption during DNN layer execution and data transfer.

In summary, these two categories primarily adopt the PP paradigm but suffer from the transmission bottleneck inherent to PP's scheduling mechanism (Sec.III-B and Sec.III-C). Achieving fast and energy-efficient inference on robotic IoT remains an open issue.

B. Dilemma on Inference Time

Layer partitioning methods consist of three phases: computing earlier DNN layers on robots, transmitting intermediate results, and completing inference on the GPU device. The GPU device's computation time is negligible compared to the other two phases (Fig.1) due to the high computing capabilities

of GPU devices. This paper focuses on the computation phase of robots and the data transfer phase via robotic IoT.

The data transmission phase can only begin after the computation phase on robots is completed, preventing overlap for a single inference task. PP can only overlap computation and data transmission phases from different inference tasks, not from the same task [5]. Existing layer partitioning methods select appropriate layers to be computed on robots, reducing the transmission volume of intermediate results based on the smaller output data size of some intermediate layers [5].

However, transmission becomes a bottleneck in robotic IoT due to limited bandwidth, and these layer partitioning methods are insufficient. Even with optimal layer partitioning [16], [4], the transmission overhead inherent to PP's scheduling mechanism remains a substantial bottleneck (Sec. IV). Moreover, adjusting the scheduling scheme to reduce traffic volume is not ideal. Allocating more layers to robots reduces transmission volume and time but increases energy consumption and underutilizes GPU devices, which may not reduce overall inference time.

concrete examples and analysis. The focus on smaller models up to 77M parameters raises multiple questions including to what extent these approaches work, what kind of latency, and throughput requirements they are able to satisfy and more importantly, when is distributed inference required given the hardware used in the testbed, Nvidia Jetson Xavier (8GB and 16GB) looks more than capable to handle sufficiently big models.

There is no discussion on what kind of applications suffer from these shortcomings and which don't. More importantly, whichever applications do suffer from the lack of distributed inference, what model sizes are they built with, the latency constraints they generally operate in, the impact to performance and accuracy when not all devices are available, and more.

C. Dilemma on Energy Consumption

The comparison of transmission and standby energy consumption in Tab.I reveals that wireless network cards consume only 0.21W during our experiments. This finding may lead people to believe that transmission in robotic IoT consumes little energy or can even be ignored. However, it is crucial to recognize that transmission energy consumption should be the the energy consumed by the device during the transmission period, not only the energy used for transmission itself (wireless network card energy consumption). Our findings reveal that such transmission energy consumption accounts for nearly one-third of the total energy consumed during inference (see Tab.I). That is because the device cannot be put into low-power sleep mode even when on standby, as it has to wait for inference results from the GPU devices and promptly continue working when it gets the inference results, and chips like CPU, GPU, and memory consume non-negligible power even when not computing, due to the static power consumption rooted in transistors' leakage current[12]. Consequently, both the energy consumed during the execution of DNN layers and the transmission energy consumption resulting from prolonged transmission times substantially impact the overall power consumption of the inference process in robotic IoT.

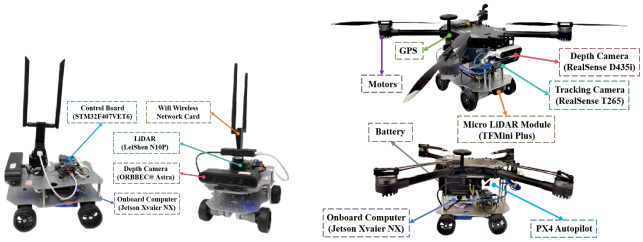
D. possible solution

Unclear novelty compared to past works: the paper applies existing works and presents results/remaining communication challenges.

IV. Evaluation

Testbed. The evaluation was conducted on a custom four-wheeled robot (Fig 3a), and a custom air-ground robot(Fig 3b). They are equipped with a Jetson Xavier NX[25] 8G onboard computer that is capable of AI model inference with local computation resources. The system runs Ubuntu 20.04 with ROS Noetic and a dual-band USB network card (MediaTek MT76x2U) for wireless connectivity. The Jetson Xavier NX interfaces with a Leishen N10P LiDAR, ORBBEC Astra depth camera, and an STM32F407VET6 controller via USB serial ports. Both LiDAR and depth cameras facilitate environmental perception, enabling autonomous navigation, obstacle avoidance, and SLAM mapping.

The GPU device accepting offloaded computation tasks from the robot is a PC equipped with an Intel(R) i5 12400f CPU @ 4.40GHz and an NVIDIA GeForce GTX 2080 Ti 11GB GPU, connected to our robot via Wi-Fi 6 over 80MHz channel at 5GHz frequency in our experiments.



(a) Four-wheeled robot (b) Air-ground robot

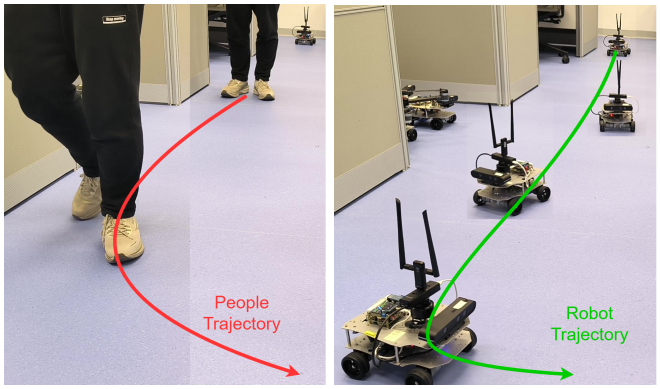
Fig. 3: The detailed composition of the robot platforms

	inference	transmission	standby
Power (W)	13.35	4.25	4.04

TABLE I: Power (Watt) of our robot in different states.

Table I presents the overall on-board energy consumption (excluding motor energy consumption for robot movement) of the robot in various states: inference (model inference with full GPU utilization, including CPU and GPU energy consumption), transmission (communication with the GPU device, including wireless network card energy consumption), and standby (robot has no tasks to execute).

We evaluated two real-world environments: indoors (robots move in our laboratory with desks and separators interfering with wireless signals) and outdoors (robots move in our campus garden with trees and bushes interfering with wireless



(a) Targeted people (b) Robot moving trajectory

Fig. 4: A real-time people-tracking robotic application on our robot based on a well-known human pose estimation ML model, KAPAO[21].

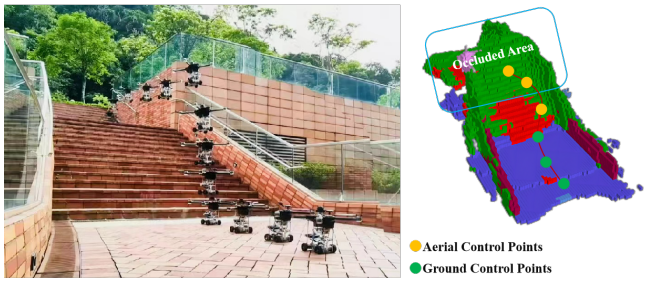


Fig. 5: By predicting occlusions in advance, AGRNav[33] gains an accurate perception of the environment and avoids collisions, resulting in efficient and energy-saving paths.

signals, resulting in lower bandwidth). The corresponding bandwidths between the robot and the GPU device in indoors and outdoors scenarios are shown in Fig.2.

Real-world Robotic Applications. We evaluated two typical real-world AI applications on robots: KAPAO, a real-time people-tracking application on our four-wheeled robot (Fig 6), and AGRNav, an autonomous navigation application on our air-ground robot (Fig 7). These applications feature different model input and output size patterns: Kapao takes RGB images as input and outputs key points of small data volume. In contrast, AGRNav takes point clouds as input and outputs predicted point clouds and semantics of similar data volume as input, implying that AGRNav needs to transmit more data during offloading.

A. *Tensor parallelism*

B. *Pipeline parallelism*

Baselines. We selected two state-of-the-art DNN layer partitioning methods as baselines: DSCCS [16], which focuses on accelerating inference, and SPSO-GA [4], which aims to optimize energy consumption. We set the deadline constraints of SPSO-GA to 1 Hz, corresponding to the minimum frequency required for controlling the robot's movement.

Baselines	SPSO-GA		DSCCS	
Environment	indoors	outdoors	indoors	outdoors
Transmission time/s	0.282	0.249	0.229	0.186
Inference time/s	0.450	0.494	0.368	0.387
Percentage(%)	62.72	50.35	62.31	48.09

TABLE II: KAPAO: the average statistics of transmission time, inference time and percentage that transmission time accounts for of inference time at inference of each frame.

Baselines	SPSO-GA		DSCCS	
Environment	indoors	outdoors	indoors	outdoors
Transmission time/s	0.169	0.225	0.084	0.113
Inference time/s	0.552	0.606	0.413	0.512
Percentage(%)	30.59	37.13	20.34	22.10

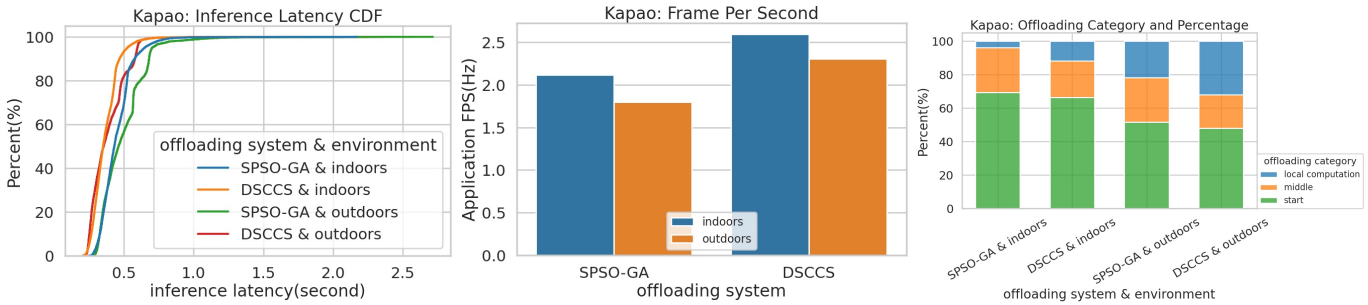
TABLE III: AGRNav. Similar to Tab.II

we mainly focus on inference time and energy consumption.

1) Inference Time: **KAPAO.** Comparing results in Tab.**??** and Tab.II with offloading enabled, both SPSO-GA and DSCCS reduced Kapao's inference time by 34.78% and 46.66% indoors and 28.41% and 44.93% outdoors, with DSCCS achieving 18.22% (indoors) and 23.08% (outdoors) lower inference time than SPSO-GA. This results in 19.23% (indoors) and 21.74% (outdoors) higher application FPS for DSCCS compared to SPSO-GA. While both systems significantly reduced inference time via offloading, transmission time accounts for 48.086% to 62.716% of the whole inference time, indicating that even with SOTA layer partitioning, the transmission bottleneck inherent to PP's scheduling mechanism cannot be mitigated.

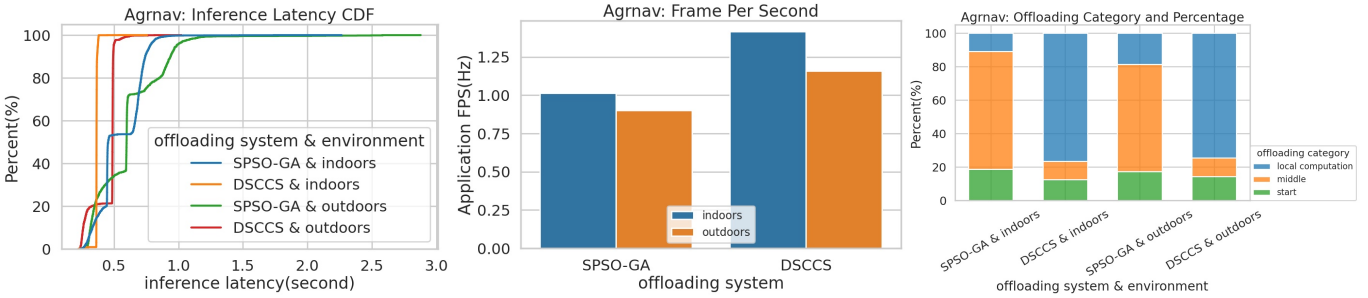
The difference between DSCCS and SPSO-GA can be attributed to their optimization goals: DSCCS minimizes inference latency, while SPSO-GA minimizes power consumption under deadline constraints. The two offloading systems both automatically adapt to available bandwidth, transitioning to all offload (placing all DNN layers on the GPU device, referring to the 'start' in Fig. 6c) when bandwidth is sufficient, and to local computation (placing all DNN layers on robots, referring to the 'local computation' in Fig. 6c) when bandwidth is too low. These accounts for high ratio of offloading category of start (offloading whole inference input) in indoors of both systems, which is reduced in outdoors and DSCCS has a higher ratio of local computation than SPSO-GA in both environments as depicted in Fig. 6c. Consequently, SPSO-GA consistently exhibits higher inference latency and lower application FPS compared to DSCCS in both environments (Fig.6a and Fig. 6b).

AGRNav. The performance gain of the two offloading systems varied for AGRNav, as shown in Tab.III and Fig.7. DSCCS still reduced inference time by 27.54% and 10.18% in indoors and outdoors and increased FPS by 27.45% and 12.75%. However, SPSO-GA achieved similar inference time and FPS as local computation indoors and even increased inference time by 6.32% and reduced FPS by 12.75% outdoors.



(a) CDF of end-to-end inference latency (b) Average processing frame per second (FPS). (c) Categories and percentage of various scheduling

Fig. 6: The performance of KAPAO under difference baselines and environments. In Fig.6c, 'start' represents placing all DNN layers on the GPU device, 'local computation' represents placing all DNN layers on robots, and 'middle' represents distributing DNN layers across both robots and GPU devices.



(a) CDF of end-to-end inference latency (b) Average processing frame per second (FPS). (c) Categories and percentage of various scheduling

Fig. 7: The performance of AGRNav under difference baselines and environments.

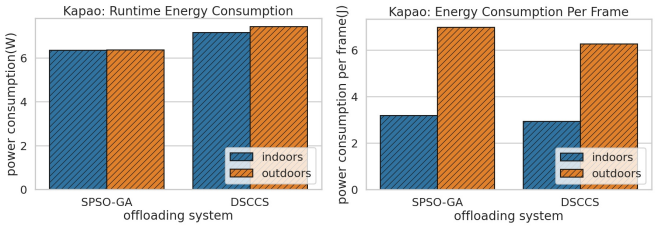
Model(number of parameters)	Local computation time/s	Environment	Transmission time/s SPSO-GA	Transmission time/s DSCCS	Inference time/s SPSO-GA	Inference time/s DSCCS	Percentage(%) SPSO-GA	Percentage(%) DSCCS
kapao(77M)	0.71(±0.02)	indoors	0.26(±0.14)	0.09(±0.13)	0.45(±0.16)	0.63(±0.13)	55.47	14.10
		outdoors	0.18(±0.51)	0.16(±0.48)	0.61(±0.52)	0.49(±0.48)	26.19	29.25
agrnav(0.84M)	1.01(±0.03)	indoors	0.25(±0.09)	0.25(±0.00)	0.95(±0.21)	0.93(±0.04)	17.03	0.02
		outdoors	0.33(±1.45)	0.64(±2.56)	1.02(±0.68)	0.92(±0.48)	6.90	2.41

TABLE IV: The average statistics of transmission time, inference time and percentage that transmission time accounts for of inference time at inference of each frame of Kapao and AGRNav. ($\pm n$) means the standard deviation of n. TODO: some case bw abnormal.

Comparing Fig.6c and Fig.7c, both SPSO-GA and DSCCS prefer to place more layers on the GPU device for AGRNav, which has more computation (see inference time in Tab. **??**).

2) Energy Consumption: Comparing the energy consumption in Tab.**??**, Fig.8a, and Fig.9a, all baselines significantly reduced power consumption by offloading computation to the GPU device. In Kapao, DSCCS consumed 14.29% and 21.88% more energy per second than SPSO-GA indoors and outdoors (Fig.8a) due to more layers placed on robots shown in Fig.6c. However, SPSO-GA consumed 8.33% and 19.05% more energy overall to process a frame than DSCCS (Fig.8b) because it only considers wireless network card energy consumption rather than the energy consumption during transmission.

In AGRNav, DSCCS consumed 37.35% and 25.00% more

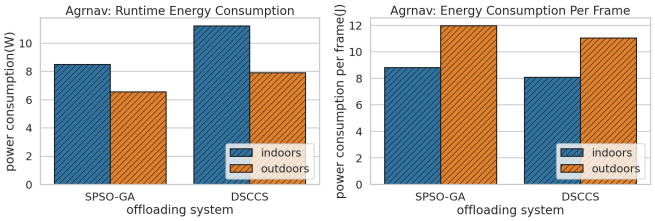


(a) Runtime Power Consumption (b) Energy Consumption Per Inference

Fig. 8: The Energy consumption of KAPAO at different baselines and environments.

Model(number of parameters)	Local computation time/s	Environment	Power consumption(W) DSCCS	Power consumption(W) SPSO-GA	Power consumption per inference(W) DSCCS	Power consumption per inference(W) SPSO-GA
kapao(77M)	15.04(±0.62)	indoors	5.86(±2.28)	13.57(±3.06)	2.65(±0.92)	8.52(±1.79)
		outdoors	11.06(±4.44)	11.27(±4.52)	6.73(±5.70)	5.52(±5.44)
agrnav(0.84M)	10.82(±1.45)	indoors	8.15(±2.78)	10.71(±1.44)	7.74(±1.72)	9.92(±0.46)
		outdoors	9.85(±2.39)	10.56(±1.72)	10.05(±6.67)	9.76(±5.07)

TABLE V: The energy consumption against time and energy consumption per inference of KAPAO and AGRNav at different baselines and environments. ($\pm n$) means the standard deviation of n.



(a) Runtime Power Consumption (b) Energy Consumption Per Inference

Fig. 9: AGRNav. Similar to Fig.8

energy per second than SPSO-GA indoors and outdoors (Fig.9a), while SPSO-GA consumed 9.32% and 9.09% more energy to process a frame than DSCCS (Fig.9b). Note that outdoors has lower power consumption than indoors in AGRNav, possibly due to more time waiting for data transmission, especially the returned output from the GPU device (Fig.7c and Tab.III).

3) Validation on a wider range of models: The conclusions reached previously remain unchanged.

V. Conclusion

In this paper, we explored the problems that hinder the application of existing parallel methods for distributed inference on robotic IoT, including the failure of data parallelism due to small batch sizes, the unacceptable communication overhead of tensor parallelism caused by all-reduce communication, and the significant transmission bottlenecks inherent to pipeline parallelism's scheduling mechanism. By raising awareness of these issues, we aim to stimulate research efforts towards developing novel parallel methods that address these problems. We envision that fast and energy-efficient inference will foster the deployment of diverse robotic tasks on real-world robots in the field.

References

[1] iPerf - Download iPerf3 and original iPerf pre-compiled binaries.

[2] Adame, T., Carrascosa-Zamacois, M., and Bellalta, B. Time-sensitive networking in ieee 802.11 be: On the way to low-latency wifi 7. *Sensors 21*, 15 (2021), 4954.

[3] Cao, A.-Q., and de Charette, R. Monoscene: Monocular 3d semantic scene completion. In *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition* (2022), pp. 3991–4001.

[4] Chen, X., Zhang, J., Lin, B., Chen, Z., Wolter, K., and Min, G. Energy-efficient offloading for dnn-based smart iot systems in cloud-edge environments. *IEEE Transactions on Parallel and Distributed Systems 33*, 3 (2021), 683–697.

[5] Crankshaw, D., Sela, G.-E., Mo, X., Zumar, C., Stoica, I., Gonzalez, J., and Tumanov, A. Inferline: latency-aware provisioning and scaling for prediction serving pipelines. In *Proceedings of the 11th ACM Symposium on Cloud Computing* (2020), pp. 477–491.

[6] Ding, M., Wang, P., López-Pérez, D., Mao, G., and Lin, Z. Performance impact of los and nlos transmissions in dense cellular networks. *IEEE Transactions on Wireless Communications 15*, 3 (2015), 2365–2380.

[7] Honkote, V., Kurian, D., Muthukumar, S., Ghosh, D., Yada, S., Jain, K., Jackson, B., Klotchkov, I., Nimmagadda, M. R., Dattawadkar, S., et al. 2.4 a distributed autonomous and collaborative multi-robot system featuring a low-power robot soc in 22nm cmos for integrated battery-powered minibots. In *2019 IEEE International Solid-State Circuits Conference-(ISSCC)* (2019), IEEE, pp. 48–50.

[8] Hu, C., Bao, W., Wang, D., and Liu, F. Dynamic adaptive dnn surgery for inference acceleration on the edge. In *IEEE INFOCOM 2019-IEEE Conference on Computer Communications* (2019), IEEE, pp. 1423–1431.

[9] Hu, Y., Imes, C., Zhao, X., Kundu, S., Beerel, P. A., Crago, S. P., and Walters, J. P. Pipeedge: Pipeline parallelism for large-scale model inference on heterogeneous edge devices. In *2022 25th Euromicro Conference on Digital System Design (DSD)* (2022), IEEE, pp. 298–307.

[10] Joseph, K. J., Khan, S., Khan, F. S., and Balasubramanian, V. N. Towards open world object detection. In *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition (CVPR)* (June 2021), pp. 5830–5840.

[11] Kang, Y., Hauswald, J., Gao, C., Rovinski, A., Mudge, T., Mars, J., and Tang, L. Neurosurgeon: Collaborative intelligence between the cloud and mobile edge. *ACM SIGARCH Computer Architecture News 45*, 1 (2017), 615–629.

[12] Kim, N. S., Austin, T., Baauw, D., Mudge, T., Flautner, K., Hu, J. S., Irwin, M. J., Kandemir, M., and Narayanan, V. Leakage current: Moore's law meets static power. *computer 36*, 12 (2003), 68–75.

[13] Li, A., Song, S. L., Chen, J., Li, J., Liu, X., Tallent, N. R., and Barker, K. J. Evaluating modern gpu interconnect: Pcie, nvlink, nv-sli, nvswitch and gpudirect. *IEEE Transactions on Parallel and Distributed Systems 31*, 1 (2019), 94–110.

[14] Li, Q., Gama, F., Ribeiro, A., and Prorok, A. Graph neural networks for decentralized multi-robot path planning. In *2020 IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS)* (2020), IEEE, pp. 11785–11792.

[15] Li, Y., Yu, Z., Choy, C., Xiao, C., Alvarez, J. M., Fidler, S., Feng, C., and Anandkumar, A. Voxformer: Sparse voxel transformer for camera-based 3d semantic scene completion. In *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition* (2023), pp. 9087–9098.

[16] Liang, H., Sang, Q., Hu, C., Cheng, D., Zhou, X., Wang, D., Bao, W., and Wang, Y. Dnn surgery: Accelerating dnn inference on the edge through layer partitioning. *IEEE transactions on Cloud Computing* (2023).

[17] Lin, B., Huang, Y., Zhang, J., Hu, J., Chen, X., and Li, J. Cost-driven off-loading for dnn-based applications over cloud, edge, and end devices. *IEEE Transactions on Industrial Informatics 16*, 8 (2019), 5456–5466.

[18] Liu, R., and Choi, N. A first look at wi-fi 6 in action: Throughput, latency, energy efficiency, and security. *Proceedings of the ACM on Measurement and Analysis of Computing Systems 7*, 1 (2023), 1–25.

[19] Liu, S., Li, X., Lu, H., and He, Y. Multi-object tracking meets moving uav. In *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition (CVPR)* (June 2022), pp. 8876–8885.

Model(number of parameters)	Local computation time/s	Environment	Transmission time/s		Inference time/s		Percentage(%)	
			SPSO-GA	DSCCS	SPSO-GA	DSCCS	SPSO-GA	DSCCS
MobileNet_V3_Small(2M)	0.033(±0.019)	indoors	0.035(±0.019)	0.016(±0.005)	0.044(±0.020)	0.031(±0.008)	79.79	53.24
		outdoors	0.035(±0.044)	0.017(±0.005)	0.047(±0.037)	0.033(±0.018)	50.04	51.49
RegNet_X_3_2GF(15M)	0.060(±0.022)	indoors	0.049(±0.026)	0.033(±0.011)	0.065(±0.028)	0.049(±0.016)	76.25	64.17
		outdoors	0.049(±0.055)	0.032(±0.032)	0.069(±0.050)	0.051(±0.030)	53.23	44.50
ResNet101(44M)	0.060(±0.023)	indoors	0.054(±0.451)	0.033(±0.010)	0.072(±0.453)	0.050(±0.016)	75.64	57.37
		outdoors	0.052(±0.064)	0.033(±0.036)	0.077(±0.059)	0.054(±0.034)	51.54	42.48
ConvNeXt_Base(88M)	0.047(±0.004)	indoors	0.033(±0.018)	0.020(±0.006)	0.044(±0.019)	0.032(±0.009)	75.39	49.37
		outdoors	0.032(±0.038)	0.020(±0.022)	0.045(±0.033)	0.034(±0.019)	52.82	35.63
ConvNeXt_Large(197M)	0.051(±0.005)	indoors	0.033(±0.017)	0.023(±0.008)	0.046(±0.019)	0.035(±0.013)	72.96	62.68
		outdoors	0.032(±0.038)	0.023(±0.024)	0.054(±0.040)	0.041(±0.028)	48.94	43.96
RegNet_Y_128GF(644M)	0.139(±0.016)	indoors	0.076(±0.289)	0.041(±0.024)	0.305(±0.882)	0.100(±0.035)	23.58	40.76
		outdoors	0.171(±0.602)	0.016(±0.055)	0.432(±0.615)	0.117(±0.242)	32.39	9.41

TABLE VI: Average running statistics of the common AI models in different environments with different offloading systems. Percentage means the percentage of inference time that transmission time accounts for.

[20] MASIUKIEWICZ, A. Throughput comparison between the new hew 802.11 ax standard and 802.11 n/ac standards in selected distance windows. *International Journal of Electronics and Telecommunications 65*, 1 (2019), 79–84.

[21] MCNALLY, W., VATS, K., WONG, A., AND MCPHEE, J. Rethinking keypoint representations: Modeling keypoints and poses as objects for multi-person human pose estimation. In *European Conference on Computer Vision* (2022), Springer, pp. 37–54.

[22] MOHAMMED, T., JOE-WONG, C., BABBAR, R., AND DI FRANCESCO, M. Distributed inference acceleration with adaptive dnn partitioning and offloading. In *IEEE INFOCOM 2020-IEEE Conference on Computer Communications* (2020), IEEE, pp. 854–863.

[23] NARAYANAN, D., SHOEYBI, M., CASPER, J., LEGRESLEY, P., PATWARY, M., KORTHIKANTI, V., VAINBRAND, D., KASHINKUNTI, P., BERNAUER, J., CATANZARO, B., ET AL. Efficient large-scale language model training on gpu clusters using megatron-lm. In *Proceedings of the International Conference for High Performance Computing, Networking, Storage and Analysis* (2021), pp. 1–15.

[24] NOORMOHAMMADPOUR, M., AND RAGHAVENDRA, C. S. Datacenter traffic control: Understanding techniques and tradeoffs. *IEEE Communications Surveys & Tutorials 20*, 2 (2017), 1492–1525.

[25] NVIDIA. The world's smallest ai supercomputer. https://www.nvidia.com/en-us/autonomous-machines/embedded-systems/jetson-xavier-series/, 2024.

[26] OHKAWA, T., YAMASHINA, K., KIMURA, H., OOTSU, K., AND YOKOTA, T. Fpga components for integrating fpgas into robot systems. *IEICE TRANSACTIONS on Information and Systems 101*, 2 (2018), 363–375.

[27] PEI, Y., MUTKA, M. W., AND XI, N. Connectivity and bandwidth-aware real-time exploration in mobile robot networks. *Wireless Communications and Mobile Computing 13*, 9 (2013), 847–863.

[28] PYTORCH. pytroch. https://pytorch.org/, 2024.

[29] REN, Y., TUNG, C.-W., CHEN, J.-C., AND LI, F. Y. Proportional and preemption-enabled traffic offloading for ip flow mobility: Algorithms and performance evaluation. *IEEE Transactions on Vehicular Technology 67*, 12 (2018), 12095–12108.

[30] SARKAR, N. I., AND MUSSA, O. The effect of people movement on wi-fi link throughput in indoor propagation environments. In *IEEE 2013 Tencon-Spring* (2013), IEEE, pp. 562–566.

[31] SIMONYAN, K., AND ZISSERMAN, A. Very deep convolutional networks for large-scale image recognition, 2015.

[32] WANG, H., POTLURI, S., LUO, M., SINGH, A. K., SUR, S., AND PANDA, D. K. Mvapich2-gpu: optimized gpu to gpu communication for infiniband clusters. *Computer Science-Research and Development 26*, 3 (2011), 257–266.

[33] WANG, J., SUN, Z., GUAN, X., SHEN, T., ZHANG, Z., DUAN, T., HUANG, D., ZHAO, S., AND CUI, H. Agrnav: Efficient and energy-saving autonomous navigation for air-ground robots in occlusion-prone environments. In *IEEE International Conference on Robotics and Automation (ICRA)* (2024).

[34] WU, H., KNOTTENBELT, W. J., AND WOLTER, K. An efficient application partitioning algorithm in mobile environments. *IEEE Transactions on Parallel and Distributed Systems 30*, 7 (2019), 1464–1480.

[35] WU, T., HE, S., LIU, J., SUN, S., LIU, K., HAN, Q.-L., AND TANG, Y. A brief overview of chatgpt: The history, status quo and potential future development. *IEEE/CAA Journal of Automatica Sinica 10*, 5 (2023), 1122–1136.

[36] XIA, Z., LIU, Y., LI, X., ZHU, X., MA, Y., LI, Y., HOU, Y., AND QIAO, Y. Scpnet: Semantic scene completion on point cloud. In *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition* (2023), pp. 17642–17651.

[37] XIANG, Y., AND KIM, H. Pipelined data-parallel cpu/gpu scheduling for multi-dnn real-time inference. In *2019 IEEE Real-Time Systems Symposium (RTSS)* (2019), IEEE, pp. 392–405.

[38] XUE, M., WU, H., PENG, G., AND WOLTER, K. Ddpqn: An efficient dnn offloading strategy in local-edge-cloud collaborative environments. *IEEE Transactions on Services Computing 15*, 2 (2021), 640–655.

[39] YANG, X., LIN, H., LI, Z., QIAN, F., LI, X., HE, Z., WU, X., WANG, X., LIU, Y., LIAO, Z., ET AL. Mobile access bandwidth in practice: Measurement, analysis, and implications. In *Proceedings of the ACM SIGCOMM 2022 Conference* (2022), pp. 114–128.

[40] YANG, Y., JUNTAO, L., AND LINGLING, P. Multi-robot path planning based on a deep reinforcement learning dqn algorithm. *CAAI Transactions on Intelligence Technology 5*, 3 (2020), 177–183.

[41] ZHUANG, Y., ZHAO, H., ZHENG, L., LI, Z., XING, E., HO, Q., GONZALEZ, J., STOICA, I., AND ZHANG, H. On optimizing the communication of model parallelism. *Proceedings of Machine Learning and Systems 5* (2023).