New Problems in Distributed Inference for DNN Models on Robotic IoT

Zekai Sun
The University of Hong Kong
Hong Kong, China
zksun@cs.hku.hk

Xiuxian Guan
The University of Hong Kong
Hong Kong, China
xxguan@cs.hku.hk

Junming Wang
The University of Hong Kong
Hong Kong, China
jmwang@cs.hku.hk

Fangming Liu
Peng Cheng Laboratory, and
Huazhong University of Science and Technology
Wuhan, China
fmliu@hust.edu.cn

Heming Cui*
The University of Hong Kong
Hong Kong, China
heming@cs.hku.hk

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.

CCS CONCEPTS

• **Computer systems organization → Robotics**; • **Networks → Network performance analysis**.

KEYWORDS

Distributed inference, Robotic IoT, Distributed system and network

ACM Reference Format:
Zekai Sun, Xiuxian Guan, Junming Wang, Fangming Liu, and Heming Cui. 2024. New Problems in Distributed Inference for DNN Models on Robotic IoT. In *Advanced Tools, Programming Languages, and PLatforms for Implementing and Evaluating algorithms for Distributed systems (ApPLIED'24), June 17, 2024, Nantes, France.* ACM, New York, NY, USA, **??** pages. https://doi.org/10.1145/3663338.3665828

*Heming Cui is the corresponding author.

1 INTRODUCTION

The rapid progress in machine learning (ML) techniques has led to remarkable achievements in various fundamental robotic tasks, such as object detection [? ? ?], robotic control [? ? ?], and environmental perception [? ? ?]. 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 [?], FPGA [?], SoC [?]), but also introduce additional energy consumption (e.g., 162% more for [?] 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 [? ? ?], where numerous GPUs are utilized to speed large model inference, such as in the case of ChatGPT [?]. 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.

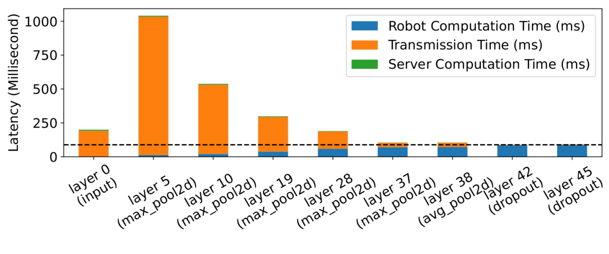
Problem 1 (DP). The small batch sizes inherent to robotic IoT applications (typically 1) hinder the mini-batch computation, rendering DP inapplicable for robotic IoT. In the data center, DP is feasible due to the large batch sizes employed (e.g., 16 images), allowing for the division of inputs into mini-batches 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 in 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 [?] 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 [?]) 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. **??**, making all-reduce synchronization an unacceptable overhead (e.g., the inference time with TP was up to 143.9X slower than local computation in our experiments).

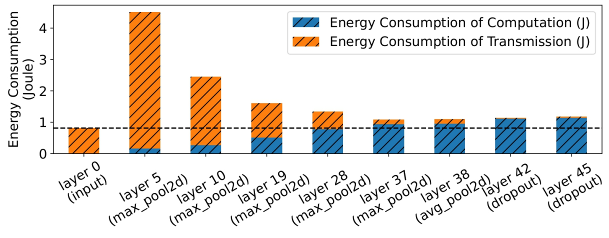
Consequently, existing distributed inference approaches [? ?] in robotic IoT primarily adopt the PP paradigm and focus on layer partitioning of PP, aiming to achieve fast and energy-efficient inference. This is because the PP paradigm in data centers consists of layer partitioning and pipeline execution, where the pipeline execution of PP enhances inference throughput rather than reducing the completion time of a single inference [?], 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 [?], 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. **??**.

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 [?]. Even with optimal layer partitioning from [? ?], 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 69% of inference time 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.



(a) Inference Latency



(b) Energy consumption

Figure 1: Our experiments on VGG19 [?] 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 server. 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.

The rest of the paper is organized as follows: Sec. **??** introduces the characteristics of robotic IoT; Sec. **??** describes in detail the problems on distributed inference on robotic IoT; Sec. **??** provides evaluation results; Sec. **??** concludes the paper.

2 BACKGROUND

2.1 Characteristics 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 [?]. However, not only the limited hardware resources on the robot can not fully play the potential of Wi-Fi 6 [?], but also the actual available bandwidth of wireless networks is often reduced in practice due to factors such as movement of the devices [? ?], occlusion from by physical barriers [? ?], and preemption of the wireless channel by other devices [? ?].

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. ??. We believe our setup represents robotic IoT devices' state-of-the-art computation and communication capabilities. We saturated the wireless network connection with iperf [?] and recorded the average bandwidth capacity between these robots every 0.1s for 5 minutes.

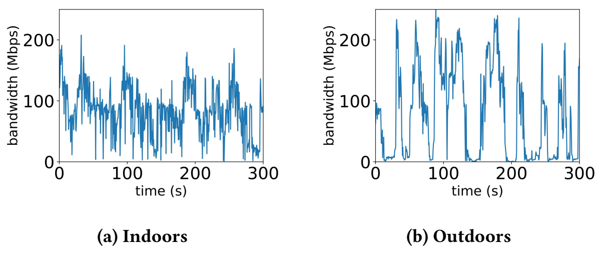


Figure 2: The instability of wireless transmission between our robot and a base station in robotic IoT networks.

The results in Fig. ?? 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.

2.2 Characteristics of Data Center Networks

Data center networks, which are used for large model inference (e.g., ChatGPT [?]), are wired and typically exhibit higher bandwidth capacity and lower fluctuation compared to robotic IoT networks. GPU devices in data centers are interconnected using high-speed networking technologies such as InfiniBand [?] or PCIe [?], offering bandwidths ranging from 40 Gbps to 500 Gbps. The primary cause of bandwidth fluctuation in these networks is congestion on intermediate switches, which can be mitigated through traffic scheduling techniques implemented on the switches [?]. The stable and high-bandwidth nature of data center networks makes them well-suited for demanding tasks like large model inference, in contrast to the more variable and resource-constrained environments found in robotic IoT networks.

2.3 Existing distributed inference strategies in the data center

Data parallelism. Data parallelism [?] 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 [?], 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, 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 [?] is a distributed inference technique that divides a model's layer parameters across multiple devices, each storing and computing a portion of the parameter tensors. 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 [?], which is beneficial when the model is too large for a single device.

In contrast to data center networks, the limited bandwidth in robotic IoT (see Sec. ??) renders the communication cost of TP prohibitively high. Our experiments demonstrate that the all-reduce communication cost of TP can consume up to 94% of the total inference time, leading to a upper to 143.9x increase in inference time and 62.7x higher energy consumption per inference compared to computing the entire model locally on the robot (see Sec.??). Such significant overhead introduced by TP's communication requirements makes it impractical for deployment in bandwidth-constrained robotic IoT environments.

Pipeline parallelism. Pipeline parallelism [?] 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 via pipeline execution, it primarily focuses on enhancing overall throughput rather than reducing single-inference latency [?], which is crucial in robotic IoT. As a result, existing distributed inference approaches [? ?] 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. ??.

2.4 Other methods to speed up DNN Models Inference on Robotic IoT

Compressed communication. Compressed communication is crucial for efficient distributed inference in wireless networks, as it significantly reduces communication overhead through techniques such as quantization and model distillation. Quantization [? ? ?] is a technique that reduces the numerical precision of model weights and activations, thereby minimizing the memory footprint and computational requirements of deep learning models. This process typically involves converting high-precision (e.g., 32-bit) floating-point values to lower-precision (e.g., 8-bit) floating-point representations, with minimal loss of model accuracy. Model distillation [? ? ?], on the other hand, is an approach that involves training a smaller, more efficient “student” model to mimic the behavior of a larger, more accurate “teacher” model by minimizing the difference between the student model's output and the teacher model's output. The distilled student model retains much of the teacher model's accuracy while requiring significantly fewer resources. These model compression methods complement distributed inference by achieving faster inference speed through model modifications, potentially sacrificing some accuracy with smaller models, while distributed inference realizes fast inference without loss of accuracy by intelligently scheduling computation tasks across multiple devices.

Inference Job scheduling. Significant research efforts have been devoted to exploring inference parallelism and unleashing the potential of layer partition to accelerate DNN inference, such as inference job scheduling, aiming to accelerate multiple DNN inference tasks by optimizing their execution on various devices under different network bandwidths while considering application-specific inference speed requirements and energy consumption demands. For instance, [? ?] support online scheduling of offloading inference tasks based on the current network and resource status of mobile systems while meeting user-defined energy constraints. [?] focused on optimizing DNN inference workloads in cloud computing using a deep reinforcement learning based scheduler for QoS-aware scheduling of heterogeneous servers, aiming to maximize inference accuracy and minimize response delay. While these methods focus on overall optimization in multi-task scenarios involving multi-robots, they do not address the optimization of single inference tasks and are thus orthogonal to distributed inference for a single inference, where improved distributed inference can provide faster and more energy-efficient inference for these scenarios.

3 PROBLEMS IN EXISTING DISTRIBUTED INFERENCE FOR DNN MODELS ON ROBOTIC IOT

3.1 Existing distributed inference on robotic IoT

Existing distributed inference approaches [? ?] in robotic IoT primarily adopt the PP paradigm and focus on layer partitioning to achieve fast and energy-efficient inference. These approaches can be divided into two main categories based on their optimization goals: accelerating inference for diverse DNN structures and optimizing robot energy consumption during inference.

To accelerate inference, earlier methods [? ? ?] focused on simple chain-like DNN models by exploiting the smaller output data sizes of intermediate layers compared to raw input data [?], creating trade-offs between computation and transmission to minimize overall inference time (see Fig. ??). 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 [?]. This issue is addressed by graph theory techniques [? ?] and varying hardware and network conditions further complicate the problem.

To optimize energy consumption, existing methods [? ? ?] build upon the aforementioned techniques and consider reducing the system energy consumption of the entire layer partitioning execution process under deadline constraints. While [?] only considers transmission energy consumption, [? ?] 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 (see Sec. ??). Consequently, achieving fast and energy-efficient inference on robotic IoT remains an open issue.

3.2 Dilemma on Inference Time and Energy Consumption

Regardless of the complexity of DNN models, layer partitioning methods consist of three phases: computing earlier DNN layers on robots, transmitting intermediate results, and completing inference on the GPU device. Since the GPU device's computation time is negligible compared to the other two phases (see Fig. ??) due to the high computing capabilities of GPU devices, this paper focuses on the computation phase of robots and the data transmission phase via robotic IoT.

The data transmission phase can only begin after obtaining the calculation result of the intermediate layer when 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 [?]. However, the transmission cost inherent to the PP's scheduling mechanism becomes a bottleneck in robotic IoT due to limited bandwidth. In our experiments, even with optimal layer partitioning [? ?], such communication cost takes up to 63% of inference time.

To make matters worse, such transmission overhead not only leads to prolonged inference time but also to high energy consumption during the data transmission phase, referred to as transmission energy consumption. Our findings reveal that such transmission energy consumption accounts for nearly one-third of the total energy consumed during inference (see Sec.??). This is because the device cannot be put into low-power sleep mode while waiting for the final inference result from the GPU device, as it has to promptly continue working when it receives the inference results. Moreover, 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 [?]. Consequently, both the energy consumed during the execution of DNN layers on robots, referred to as robot computation energy consumption, and the transmission energy consumption resulting from prolonged transmission times substantially impact the overall power consumption of the inference process in robotic IoT.

Only models with limited transmission overhead can mitigate the impact of these shortcomings on inference performance. However, the unstable bandwidth in robotic IoT wireless networks can cause the transmission time for layer partitioning to vary dramatically, sometimes changing by hundreds of times (see Fig. ??). In our experiments, even a relatively small model with only 0.84M parameters still suffers from its significant transmission overhead. The significant impact of transmission overhead on both inference time and energy consumption highlights the need for innovative approaches that can effectively mitigate the transmission bottleneck in robotic IoT.

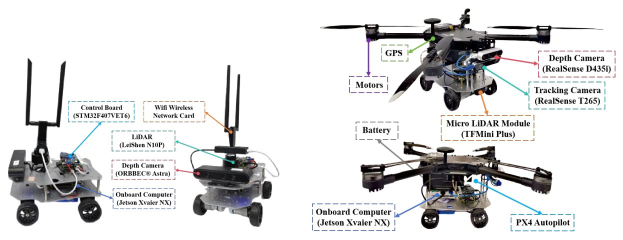
3.3 Special Cases

Since layer partitioning methods schedule at the granularity of model layers, "local computation" and "edge computation" are special cases of layer partitioning. "Local computation" refers to placing the whole layers on the robot when the bandwidth is too low, while "edge computation" means placing the whole layers on GPU devices when the bandwidth is sufficient. Local computation avoids the impact of network transmission on inference time but consumes the maximum computation energy consumption. On the other hand, edge computation minimizes computation energy consumption but requires a high enough bandwidth to ensure the lowest possible transmission energy consumption and overall inference time. These two special cases are indispensable for existing methods to cope with different network conditions, when they are too low or sufficient, and to address the need for various trade-offs between inference delay and energy consumption.

In our experiments, we found that the bandwidth conditions under which the layer partitioning scheme of different models becomes these special cases vary, and the higher the bandwidth, the more layers are scheduled to be placed on GPU devices. We explain the reasons causing different bandwidth conditions for different models in Sec. ?? with some detailed real-world cases. The existence of these special cases highlights the importance of considering the relationship between bandwidth, model structure, and the resulting trade-offs between inference delay and energy consumption.

4 EVALUATION

Testbed. The evaluation was conducted on a custom four-wheeled robot (Fig ??), and a custom air-ground robot(Fig ??). They are equipped with a Jetson Xavier NX [?] 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 server 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.

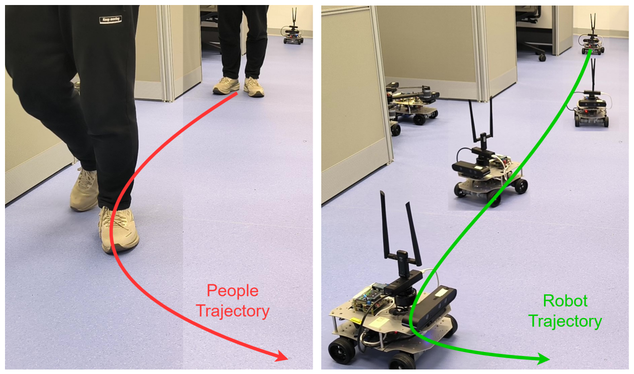

Control Board (STM32F407VET6)
Wifi Wireless Network Card
LiDAR (LeiShen N10P)
Depth Camera (ORBBEC® Astra)
Onboard Computer (Jetson Xvaier NX)
GPS
Motors
Depth Camera (RealSense D435i)
Tracking Camera (RealSense T265)
Micro LiDAR Module (TFMini Plus)
Battery
Onboard Computer (Jetson Xvaier NX)
PX4 Autopilot

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

Figure 3: The detailed composition of the robot platforms

	inference	transmission	standby
Power (W)	13.35	4.25	4.04

Table 1: Power consumption (Watt) of our robot in different states.



(a) Targeted people (b) Robot moving trajectory

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

Tab. ?? 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 server, including wireless network card energy consumption), and standby (robot has no tasks to execute). Notice that different models, due to varying numbers of parameters, exhibit distinct GPU utilization rates and power consumption during inference.

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 signals, resulting in lower bandwidth). The corresponding bandwidths between the

Model(number of parameters	Local computation time(s)	Environment	Transmission time (s) with TP	Inference time (s) with TP	Percentage(%) with TP
MobileNet_V3_Small(2M)	0.031(±0.004)	indoors	0.698(±0.135)	1.400(±0.232)	49.85
		outdoors	0.901(±0.778)	1.775(±1.370)	51.23
ResNet101(44M)	0.065(±0.005)	indoors	7.156(±3.348)	8.106(±3.403)	87.95
		outdoors	8.470(±6.337)	9.356(±6.328)	90.46
VGG19_BN(143M)	0.063(±0.002)	indoors	5.152(±4.873)	5.444(±4.831)	70.18
		outdoors	5.407(±6.673)	5.759(±6.635)	93.70

Table 2: Average transmission time (Second), inference time (Second), percentage that transmission time accounts for of the total inference time and their standard deviation (±n) with TP on different models in different environments. "Local computation" refers to placing the whole layers on the robot.

Model(number of parameters)	Environment	Power consumption(W)		Energy consumption(J) per inference	
		Local	TP	Local	TP
MobileNet_V3_Small(2M)	indoors	6.05(±0.21)	5.24(±0.19)	0.3(±0.09)	7.33(±1.21)
	outdoors	6.05(±0.21)	5.11(±0.28)	0.3(±0.09)	9.08(±7.0)
ResNet101(44M)	indoors	11.27(±0.51)	4.97(±0.16)	0.93(±0.19)	40.28(±16.91)
	outdoors	11.27(±0.51)	4.9(±0.23)	0.93(±0.19)	45.8(±30.98)
VGG19_BN(143M)	indoors	14.86(±0.43)	4.88(±0.29)	1.19(±0.18)	26.55(±23.56)
	outdoors	14.86(±0.43)	4.87(±0.27)	1.19(±0.18)	28.06(±32.33)

Table 3: Power consumption against time (Watt) and energy consumption per inference (Joule) with standard deviation (±n) with TP on different models in different environments. "Local" represents "Local computation"

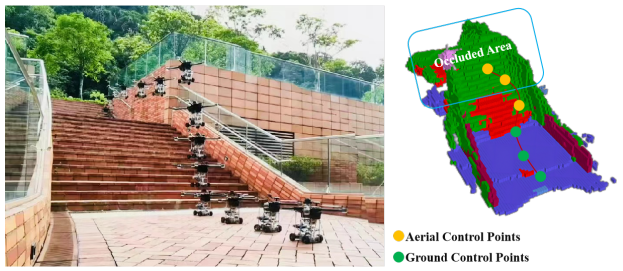


Figure 5: By predicting occlusions in advance, AGRNav [?] gains an accurate perception of the environment and avoids collisions, resulting in efficient and energy-saving paths.

robot and the GPU server in indoors and outdoors scenarios are shown in Fig. ??.

Workload. We evaluated two typical real-world robotic applications on our testbed: Kapao, a real-time people-tracking application on our four-wheeled robot (Fig ??), and AGRNav, an autonomous navigation application on our air-ground robot (Fig ??). 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. And we have verified several models common to mobile devices on a larger scale to further corroborate our observations and findings: MobileNet [?], ResNet [?], VGGNet [?], ConvNeXt [?], RegNet [?].

Notice that in our experiment, the robot continuously captures the latest images from the camera for inference with a batch size of 1, precluding the adoption and evaluation of data parallelism methods.

4.1 Tensor Parallelism

We chose a state-of-the-art tensor parallelism method, DINA [?], as our baseline; Table ?? reveals that transmission time constitutes 49% to 94% of total inference time due to all-reduce communication for each layer, resulting in TP's inference time being 45.2X to 143.9X longer than local computation. Although Table ?? indicates lower power consumption with TP (13.4% to 67.3% less than local computation, because TP spent much more time on transmission when have lower power consumption in Tab.??), the extended transmission times significantly increase energy consumption per inference, ranging from 28.5X to 62.7X. Since TP significantly extends inference time, making it impractical for real-world robotic applications that require real-time control, we did not further evaluate TP in these contexts.

4.2 Pipeline Parallelism

We selected two SOTA pipeline parallelism methods as baselines: DSCCS [?], aimed at accelerating inference, and SPSO-GA [?], focused on optimizing energy consumption. We set SPSO-GA's deadline constraints to 1 Hz, the minimum frequency required for robot movement control. Given our primary focus on inference time and energy consumption per inference, we disabled pipeline execution to concentrate solely on assessing the performance of various layer partitioning methods.

4.2.1 Inference Time.

Kapao. From the results in the upper part of Tab. ??, both SPSO-GA and DSCCS reduced Kapao's inference time by 39.69% and 56.92% indoors and 28.67% and 47.46% outdoors, with DSCCS achieving 28.57% (indoors) and 26.34% (outdoors) lower inference time than

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
Kapao(77M)	0.708(±0.023)	indoors	0.212(±0.085)	0.204(±0.088)	0.427(±0.122)	0.305(±0.113)	49.69	66.68
		outdoors	0.271(±0.563)	0.259(±0.531)	0.505(±0.573)	0.372(±0.535)	53.49	69.46
AGRNav(0.84M)	1.014(±0.034)	indoors	0.273(±0.166)	0.133(±0.793)	0.977(±0.32)	0.828(±0.646)	28.04	15.99
		outdoors	0.177(±0.762)	0.089(±0.085)	0.983(±0.759)	0.888(±0.067)	17.93	10.02

Table 4: Average transmission time (Second), inference time (Second), percentage that transmission time accounts for of the total inference time and their standard deviation ($\pm n$) of Kapao and AGRNav with different pipeline parallelism offloading systems and different environments. "Local computation" refers to placing the whole layers on the robot.

Model(number of parameters)	Environment	Power consumption(W)			Energy consumption(J) per inference		
		Local	SPSO-GA	DSCCS	Local	SPSO-GA	DSCCS
Kapao(77M)	indoors	15.03(±0.63)	6.21(±2.76)	6.42(±3.09)	9.26(±0.2)	1.95(±0.76)	1.23(±0.71)
	outdoors	15.03(±0.63)	7.91(±4.2)	8.07(±4.35)	9.26(±0.2)	2.92(±4.53)	1.95(±4.32)
AGRNav(0.84M)	indoors	10.82(±1.44)	6.47(±2.06)	10.43(±2.4)	10.97(±0.37)	2.95(±1.93)	4.48(±6.71)
	outdoors	10.82(±1.44)	8.77(±3.07)	10.78(±1.47)	10.97(±0.37)	4.7(±6.62)	4.97(±0.47)

Table 5: The power consumption against time (Watt) and energy consumption per inference (Joule) with standard deviation ($\pm n$) of Kapao and AGRNav at different baselines and environments. "Local" represents "Local computation"

SPSO-GA. While both systems significantly reduced inference time via offloading, transmission time accounts for 49.69% to 69.46% 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.

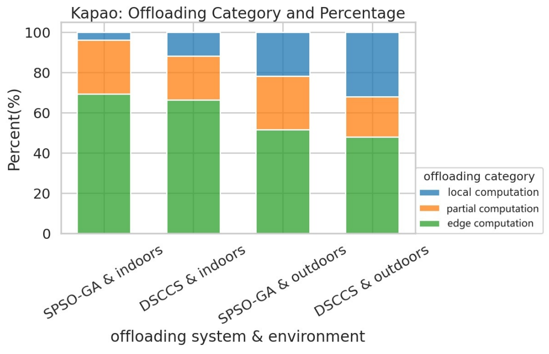
AGRNav. The performance gain of the two offloading systems varied for AGRNav, as shown in the lower part of Tab. ??. DSCCS still reduced inference time by 18.34% and 12.43% in indoors and outdoors. However, SPSO-GA achieved similar inference time (3.65% and 3.06% reduction) as local computation both indoors and outdoors. We will explain and analyze this phenomenon in Sec.??.

Notice that the large standard deviation in transmission time in outdoors in both offloading systems indicates that bandwidth fluctuated more frequently and more fiercely outdoors compared with indoors, which complies with Fig. ??. Additionally, the lower average bandwidth for outdoors scenarios (see Sec.??) results in increased transmission and inference times relative to indoor scenarios.

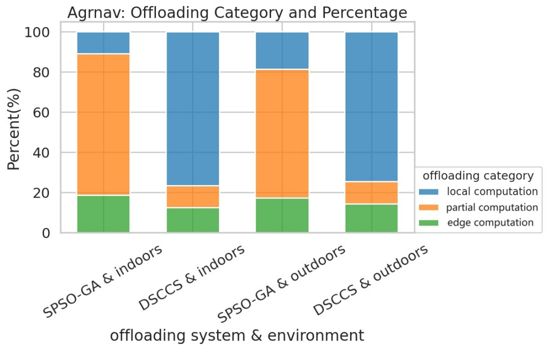
4.2.2 Breakdown.

Both SPSO-GA and DSCCS automatically adapt to available bandwidth, transitioning to edge computation (placing all DNN layers on the GPU server) when bandwidth is sufficient, and to local computation (placing all DNN layers on robots) when bandwidth is low. To better understand how their layer partitioning scheduling varies with different network conditions and models, we recorded and analyzed the Categories and percentages of various layer partitioning schedules under different baselines and environments, as detailed in Fig. ??.

Local computation and edge computation are special cases of layer partitioning, with the bandwidth conditions required for each model to reach these cases varying based on the model structure and partitioning method used. Analyzing Fig. ?? and Fig. ??, both SPSO-GA and DSCCS tend to allocate more layers on the robot



(a) Categories and percentage of various scheduling for Kapao



(b) Categories and percentage of various scheduling for AGRNav

Figure 6: The layer partitioning scheduling under difference baselines and environments. "Local computation" refers to placing the whole layers on the robot when the bandwidth is too low, "edge computation" means placing the whole layers on GPU server when the bandwidth is sufficient, and "partial computation" means placing part of the layers on the robot and part on GPU server.

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(±0.382)	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 6: Average transmission time (Second), inference time (Second), percentage that transmission time accounts for of the total inference time and their standard deviation ($\pm n$) of common AI models in different environments with different offloading systems. "Local computation" refers to placing the whole layers on the robot.

Model(number of parameters)	Environment	Power consumption(W)			Energy consumption(J) per inference		
		Local	SPSO-GA	DSCCS	Local	SPSO-GA	DSCCS
MobileNet_V3_Small (2M)	indoors	6.131(±0.061)	5.448(±0.168)	5.658(±0.085)	0.202(±0.002)	0.241(±0.107)	0.174(±0.046)
	outdoors	6.131(±0.061)	5.567(±0.273)	5.557(±0.186)	0.202(±0.002	0.260(±0.204)	0.185(±0.099)
RegNet_X_3_2GF (15M)	indoors	8.208(±0.140)	5.490(±0.178)	5.714(±0.342)	0.492(±0.008)	0.356(±0.156)	0.278(±0.091)
	outdoors	8.208(±0.140)	5.878(±0.659)	6.041(±0.624)	0.492(±0.008)	0.406(±0.295)	0.311(±0.184)
ResNet101 (44M)	indoors	11.851(±0.404)	5.457(±0.240)	5.953(±0.789)	0.711(±0.024)	0.390(±2.471)	0.298(±0.094)
	outdoors	11.851(±0.404)	6.179(±1.083)	6.431(±1.060)	0.711(±0.024)	0.478(±0.364)	0.349(±0.216)
ConvNeXt_Base (88M)	indoors	15.335(±0.273)	5.507(±0.358)	7.713(±2.613)	0.721(±0.013)	0.241(±0.103)	0.250(±0.069)
	outdoors	15.335(±0.273)	7.638(±3.297)	9.148(±3.338)	0.721(±0.013)	0.346(±0.254)	0.307(±0.171)
ConvNeXt_Large (197M)	indoors	15.403(±0.082)	5.518(±0.638)	6.604(±2.860)	0.786(±0.004)	0.251(±0.104)	0.230(±0.088)
	outdoors	15.403(±0.082)	8.400(±4.345)	8.895(±4.505)	0.786(±0.004)	0.452(±0.339)	0.366(±0.248)
RegNet_Y_128GF (644M)	indoors	15.430(±0.020)	5.384(±1.071)	6.151(±2.155)	2.145(±0.003)	1.642(±0.327)	0.615(±0.216)
	outdoors	15.430(±0.020)	6.361(±2.349)	9.127(±4.724)	2.145(±0.003)	2.748(±1.015)	1.068(±0.553)

Table 7: The power consumption against time (Watt) and energy consumption per inference (Joule) with standard deviation ($\pm n$) of common AI models at different baselines and environments. "Local" represents "Local computation"

for AGRNav. When comparing indoor and outdoor scenarios in Fig. ??, it is evident that higher bandwidth leads to more layers being scheduled on GPU server. Additionally, when comparing SPSO-GA and DSCCS in Fig. ??, DSCCS, which focuses on optimizing energy consumption, tends to place fewer layers on the robot to reduce computation energy consumption.

In summary, the conditions under which layer partitioning schemes make these special cases are influenced by multiple factors: model structure, and the trade-offs between inference delay and energy consumption. And the higher the bandwidth, the more layers are scheduled to be placed on GPU server.

4.2.3 Energy Consumption.

Kapao. From the results in the upper part of Tab. ??, DSCCS consumed 3.38% and 2.02% more power per second than SPSO-GA indoors and outdoors due to more layers placed on robots shown in Fig. ??. However, SPSO-GA consumed 58.54% and 49.74% more energy overall to process a frame than DSCCS because it only aims at minimizing the power consumption against time at the cost of possibly prolonged inference time.

AGRNav. From the results in the lower part of Tab. ??, DSCCS consumed 61.21% and 22.92% more energy per second than SPSO-GA indoors and outdoors (Tab. ??), while DSCCS consumed 34.15% and 5.43% more energy to process a frame than SPSO-GA. SPSO-GA's advantages in power consumption aganist time shrinks in energy consumption per inference due to prolonged inference time.

4.2.4 Validation on a larger range of models.

We evaluated PP across a broad range of models with varying parameter counts (from 0.84M to 644M, as detailed in Tab. ?? and Tab. ??), which are commonly used in mobile devices. Our findings confirm that transmission time constitutes a significant portion of the total inference time in robotic IoT when using PP. The inherent transmission overhead of PP's scheduling mechanism significantly wastes both inference time and energy.

4.3 Possible Solutions

To address the transmission overhead issue of PP, approaches that can reduce or overlap communication costs within a single inference task appear to be viable solutions. Given the requirement to maintain the integrity of the final inference results, intermediate results cannot be altered during transmission. Consequently, only lossless compression methods [?] can be utilized to reduce the transmission volume.

Regarding the overlapping of communication costs within a single inference task, since transmission time constitutes a significant portion of the total inference time (approximately half) in existing PP's layer partitioning, a novel parallel method that overlaps the computation and transmission phases has significant potential for optimizing and speeding up inference time. Furthermore, it is essential to recognize that transmission energy consumption encompasses the energy used by the device during the data transmission phase, not merely the energy expended for the transmission itself (such as that by the wireless network card). The comparison of transmission and standby energy consumption in Tab. **??** also indicates that wireless network cards consume only 0.21W during our experiments. This suggests that overlapping the two phases will not significantly increase the energy consumption during the robot computation phase but will reduce the robot's waiting time for the final inference result during the data transmission phase, thereby decreasing overall energy consumption.

In summary, a new parallel method that overlaps the computation and transmission phases within the existing PP's layer partitioning framework has the potential to address the shortcomings of current distributed inference approaches. By implementing this method, we can achieve faster and more energy-efficient inference, facilitating more effective deployment of DNN models on robotic IoT.

5 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.

ACKNOWLEDGEMENTS

We thank all anonymous reviewers for their helpful comments. The work is supported in part by National Key R&D Program of China (2022ZD0160200), National Key R&D Program of China (No.2023YFB4503902), HK RIF (R7030-22), HK ITF (GHP/169/20SZ), the Huawei Flagship Research Grants in 2021 and 2023, and HK RGC GRF (Ref: HKU 17208223), the HKU-SCF FinTech AcademyR&D Funding Schemes in 2021 and 2022, the HKU-CAS Joint Laboratory for Intelligent System Software, and the Shanghai Artificial Intelligence Laboratory.