Intra-DP: A High Performance Distributed Inference System on Robotic IoT

Anonymous Author(s)
Submission Id: 445

Abstract

The rapid advancements in machine learning (ML) techniques have led to significant achievements in various robotic tasks, where deploying these ML approaches on real-world robots requires fast and energy-efficient inference of their deep neural network (DNN) models. 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 over Internet of Things of these real-world robots (robotic IoT), all existing parallel methods (data parallelism, tensor parallelism, pipeline parallelism) fail to simultaneously meet the robots' latency and energy requirements and raise significant challenges due to the failure of data parallelism, the unacceptable communication overhead of tensor parallelism, and the significant transmission bottlenecks in pipeline parallelism due to the limited bandwidth of robotic IoT.

We present Intra-DP, the first high-performance distributed inference system optimized for model inference on robotic IoT. Intra-DP introduces a fine-grained approach to transmission and computation by confining them to each local operator of DNN layers (i.e., operators that can be computed independently without the complete input, such as the convolution kernel in the convolution layer). By adaptively scheduling the computation and transmission of each local operator based on various hardware conditions and network bandwidth, Intra-DP enables different local operators of different layers to be computed and transmitted concurrently, and overlap the computation and transmission phases within the same inference task to achieve fast and energy-efficient distributed inference on robotic IoT. The evaluation shows that Intra-DP reduces inference time by 20% and energy consumption by 10% compared to the state-of-the-art baselines.

Keywords: Distributed inference, Robotic IoT, Distributed system and network

1 Introduction

The rapid progress in machine learning (ML) techniques has led to remarkable achievements in various fundamental robotic tasks, such as object detection [20, 30, 33], robotic control [24, 50, 60], and environmental perception [4, 25, 55]. 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 [37], FPGA [38], SoC [16]), but also introduce additional energy consumption (e.g., 162% more for [33] 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 [18, 56, 64], where numerous GPUs are utilized to speed large model inference, such as in the case of ChatGPT [54]. 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).

For 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.

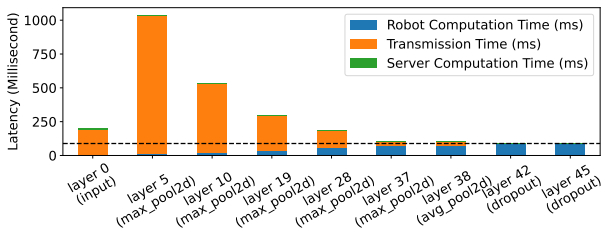
TP requires frequent synchronization among devices, leading to unacceptable communication overhead in robotic IoT.

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 [64] 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 [23]) 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. 2.1, making all-reduce synchronization an unacceptable overhead (e.g., the inference time with TP was up to 143.9X slower than local computation in Sec.2.3).

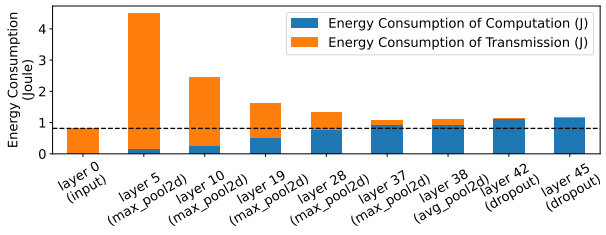
Consequently, existing distributed inference approaches [5, 26] in robotic IoT primarily adopt the PP paradigm and focus on layer partitioning of PP, aiming to achieve fast and energy-efficient inference. This is because the PP paradigm in data centers consists of layer partitioning and pipeline execution, where the pipeline execution of PP enhances inference throughput rather than reducing the completion time of a single inference [6], 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 [17], DNN layer partitioning strategies 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.

However, existing methods based on PP face significant transmission bottlenecks in robotic IoT due to the inherent scheduling mechanism. PP paradigm on robotic IoT consists of three sequential phases: computing earlier DNN layers on robots, transmitting intermediate results, and completing inference on the GPU server. Despite optimal layer partitioning strategies from [5, 26], the transmission overhead becomes a substantial bottleneck due to limited bandwidth of robotic IoT, accounting for up to 69% of inference time in our experiments. While the pipeline execution of PP can mitigate this overhead by overlapping computation and transmission phases across multiple inference tasks, it cannot reduce the completion time of a single inference task [6], which is crucial for robotic applications. Furthermore, the prolonged transmission phase not only slows down inference speed but also consumes significantly more energy.

The key reason for the problem of the above methods is that existing methods conduct layer-granulated scheduling, whose transmission time is typically longer than the computation time due to the limited bandwidth of real-world robotic IoT networks. As transmission time constitutes a substantial portion of the total inference time (approximately half) in existing methods, a novel parallel method with finer granularity that overlaps computation and transmission within the same inference task has the potential to address this



(a) Inference Latency



(b) Energy consumption on robot

Figure 1. Existing distributed inference approaches on VGG19 [45] in our experiments, which adopt PP paradigm with various layer partitioning scheduling strategies. The X-axis of the graph represents different layer partitioning strategies, where 'layer i' indicates 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.

shortcoming, achieving fast and energy-efficient inferences. Note that the robot can not enter low-power sleep mode during the transmission phase due to the need to promptly continue working upon receiving inference results, but can only enter standby mode, when chips like CPU, GPU, and memory consume non-negligible power even when not computing (e.g., 95% power consumption in our experiments). Such a parallel method would reduce the robot's standby time without significantly increasing energy consumption during the computation phase, thereby decreasing overall energy consumption.

In this paper, we present Intra-DP (Intra-Data Parallel), a high-performance distributed inference system optimized for real-world robotic IoT networks. We discovered that operators for each DNN layer (e.g., convolution, ReLU, softmax) can be categorized into two types: local operators and global operators, depending on whether they can be computed independently without the complete input. For instance, softmax [31] requires the complete input vector to calculate the corresponding probability distribution, referring to it as a global operator, while ReLU [7] and convolution [34] can be computed with part of the input tensor (the elements in the input vector for ReLU and the blocks in the input tensor

for convolution), referring to them as local operators. Local operators need to be executed several times to complete the calculation of the whole layer; in this article, we treat each execution of the local operator as an independent local operator for easy discussion. Local operators are widely used in robotic applications, especially convolution layers in computer vision [33] and point cloud tasks [50]. The local operator granularity provides a finer granularity for Intra-DP, allowing different local operators of different layers to be computed and transmitted concurrently, enabling the overlapping of computation and transmission phases within the same inference task to achieve fast and energy-efficient inference.

The design of Intra-DP is confronted with two major challenges. The first one is how to guarantee the correctness of inference results based on local operator. We propose Local Operator Parallelism (LOP), which reduces the granularity of calculation from each layer to each local operator. LOP determines the correct input required for different local operators based on their calculation characteristics and processes at first. When a part of the local operators in a layer completes the calculation and the tensor composed of these local operators satisfies the input requirements of the local operators in the subsequent layer, the local operators in the subsequent layer can be calculated in advance, without waiting for all local operators of the current layer to be computed in LOP. For global operator layers, Intra-DP enforces a synchronization before these layers to combine the complete input for them, as TP's all-reduce communications do. In this way, Intra-DP only change the execution sequence of local operators among local operator layers and ensures the calculation correctness of local operator layers through LOP and global operator layers through synchronization. We formally prove that LOP guarantees the correctness of inference results in Sec. 4.1.

The second challenge is under LOP, how to properly schedule the computation and transmission of each local operator to achieve fast and energy-efficient inference under various hardware conditions and network bandwidths. Intra-DP places part of the local operator execution on GPU servers and transmits the corresponding part of the input tensor based on LOP, while computing the rest of the local operators with a novel Local Operator Scheduling Strategy (LOSS). LOSS formulates the problem of determining which part of the local operators should be executed on robots and which part should be executed on GPU servers as a nonlinear optimization problem (see Sec. 4.2), and schedules the computation and transmission of each local operator based on the solution obtained via the differential evolution algorithm [42].

We implemented Intra-DP in PyTorch [40] and evaluated Intra-DP on our real-world robots under two typical real-world robotic applications [33, 50] and several models common to mobile devices on a larger scale [45, 46, 48, 52, 57]. We compared Intra-DP with two SOTA pipeline parallelism methods as baselines: DSCCS [26], aimed at accelerating inference, and SPSO-GA [5], focused on optimizing energy consumption, under different real-world robotic IoT networks environments (namely indoors and outdoors). Evaluation shows that:

- Intra-DP is fast. Intra-DP reduced inference time by 80% ~98% compared to baselines under indoors and outdoors environments.
- Intra-DP is energy-efficient. Intra-DP reduced 91% ~98% energy consumption per inference compared to baselines, due to faster inference speed and no-increased power consumption against time.
- Intra-DP is robust in various robotic IoT environments. When the robotic IoT environment changed (from indoors to outdoors), Intra-DP's superior performance remained consistent.
- Intra-DP is easy to use. It took only three lines of code to apply Intra-DP to existing ML applications.

Our main contribution are LOP, a fine-grained parallel method based on local operators, and LOSS, a new scheduling strategy based on LOP optimized for distributed inference over real-world robotic IoT networks. By leveraging these contributions, Intra-DP dramatically reduces the transmission overhead in existing distributed inference on robotic IoT by overlapping the computation and transmission phases within the same inference task, achieving fast and energy-efficient distributed inference on robotic IoT. We envision that the fast and energy-efficient inference of Intra-DP will foster the deployment of diverse robotic tasks on real-world robots in the field. Intra-DP's code is released on https://github.com/xxx/xxx.

In the rest of this paper, we introduce the background of this paper in Sec. 2, give an overview of Intra-DP in Sec. 3, present the detailed design of Intra-DP in Sec. 4, evaluate Intra-DP in Sec. 6, and finally conclude in Sec. 7.

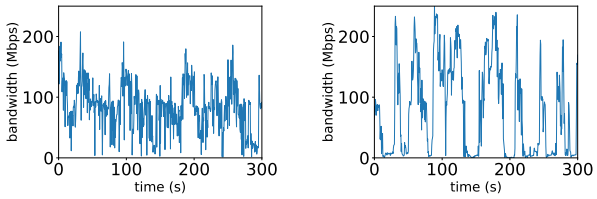
2 background

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 [29]. However, not only the limited hardware resources on the robot can not fully play the potential of Wi-Fi 6 [59], but also the actual available bandwidth of wireless networks is often reduced in practice due to factors such as movement of the devices [32, 39], occlusion from by physical barriers [9, 44], and preemption of the wireless channel by other devices [2, 43].

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. 6. 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

Figure 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. Moreover, the unstable bandwidth in robotic IoT wireless networks can cause the transmission time to vary dramatically, sometimes changing by hundreds of times. In our experiments, even a relatively small model with only 0.84M parameters still suffers from its significant transmission overhead.

2.2 Characteristics of Data Center Networks

Data center networks, which are used for large model inference (e.g., ChatGPT [54]), 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 [49] or PCIe [23], 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 [36]. 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 methods in the data center

Data parallelism. Data parallelism [56] 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 [35], 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 [64] 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 [23], 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 renders the communication cost of TP prohibitively high, as evidenced by our evaluation of DINA [34], a state-of-the-art TP method. Table 1 reveals that transmission time constitutes 49% to 94% of the 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 2 indicates lower power consumption with TP (13.4% to 67.3% less than local computation), 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, we did not further consider TP in this paper.

Pipeline parallelism. Pipeline parallelism [18] is a distributed inference technique that partitions DNN model

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 1. Average transmission time (Second), inference time (Second), percentage that transmission time accounts for of inference time and their standard deviation ($\pm n$) 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 2. Power consumption against time (Watt) and energy consumption per inference (Joule) with standard deviation ($\pm n$) with TP on different models in different environments. "Local" represents "Local computation"

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 [6], which is crucial in robotic IoT. As a result, existing distributed inference approaches [5, 26] in robotic IoT primarily adopt PP paradigm and focus on layer partitioning to achieve fast and energy-efficient inference, with two main categories based on their optimization goals: accelerating inference for diverse DNN structures [17, 21, 26, 34, 58] and optimizing robot energy consumption during inference [5, 27, 53]. Both two kinds of methods suffer from the transmission bottleneck inherent to PP's scheduling mechanism, which can be eliminated by Intra-DP.

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 [8, 13, 14] 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 [15, 28, 51], 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, [3,

10] support online scheduling of offloading inference tasks based on the current network and resource status of mobile systems while meeting user-defined energy constraints. [11] 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 Overview

3.1 Workflow of Intra-DP

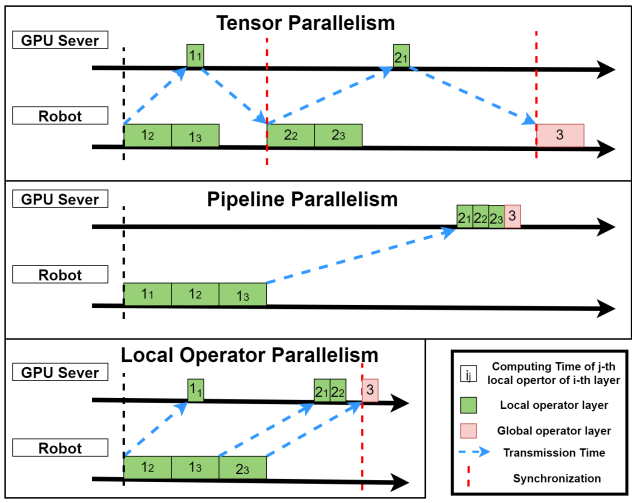


Figure 3. Workflow of Intra-DP. Each local operator layer have to complete the calculation of three local operators, and the same local operator in the three cases has the same computation time on robots and GPU servers, as well as the corresponding transmission time. The output tensor volume of layer 2 is larger than that of layer 1, resulting in longer transmission times for local operators in layer 2, and PP selects a layer partition strategy at layer 1 [26].

Fig. 3 presents the workflow of Intra-DP and compares it with TP and PP under robotic IoT networks with limited bandwidth, illustrating why existing methods suffer from transmission overhead and how Intra-DP solves this issue via its LOP.

While TP can place some local operator execution on the GPU server, it requires an all-reduce communication [64] to combine computation results from different devices, which entails significant communication overhead (as shown by the red dotted lines for synchronization Fig. 3). Although the layer partition algorithm [26] can be used to minimize overall inference time, the transmission time still becomes a significant bottleneck, as illustrated by the extremely long transmission in Fig. 3.

To alleviate the transmission overhead in distributed inference, Intra-DP overlaps the computation and transmission of different local operators from different local operator layers, as shown in Fig. 3. Compared with TP, Intra-DP cancels the synchronization of the all-reduce communication for local operator layers and ensures that each local operator can get the required input in time and obtain the correct calculation results through LOP, rather than relying on all-reduce communication for local operator layers. Intra-DP maintains synchronization for global operator layers, which do not have local operators and require the complete input (the entire output tensor of the previous layer) to perform the calculation, ensuring the correctness of the global operator layers' inference. Compared with PP, Intra-DP starts transmitting some local operators and the corresponding input in advance, without waiting for all local operators in the current local operator layer to complete the calculation. In this way, Intra-DP achieves much faster inference compared with existing distributed inference methods.

Moreover, the idle time on the robot (when the robot is not computing, as shown in Fig. 3) consumes significant energy. This is because the robot cannot enter a low-power sleep mode while waiting for the final inference result from the GPU server, as it has to promptly continue working when it receives the inference results. During the standby phase (idle time), 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 [22]. Meanwhile, we found that wireless network cards consume only 0.21 Watt for transmission during the idle time, while the robot consumes 13.35 Watt during computing. In this way, Intra-DP dramatically reduces the idle time on the robot, alleviating the energy wasted by standby mode, and increases a negligible amount of network card transmission power consumption during computing, thereby reducing the overall energy consumption for each inference.

To achieve the workflow shown in Fig. 3, the design of Intra-DP must tackle two problems: guaranteeing the correctness of inference results based on local operators and scheduling the computation and transmission of each local operator. In Sec.4.1, we will explain how Intra-DP ensures that each local operator can still obtain the correct calculation result via LOP, and in Sec.4.2, we will discuss how Intra-DP achieves fast and energy-efficient inference through its LOSS.

3.2 Architecture of Intra-DP

Fig. 4 shows the architecture of Intra-DP, which adds an interceptor for each DNN layer to flexibly split the input tensor and combine the output tensor for each operator. Compared

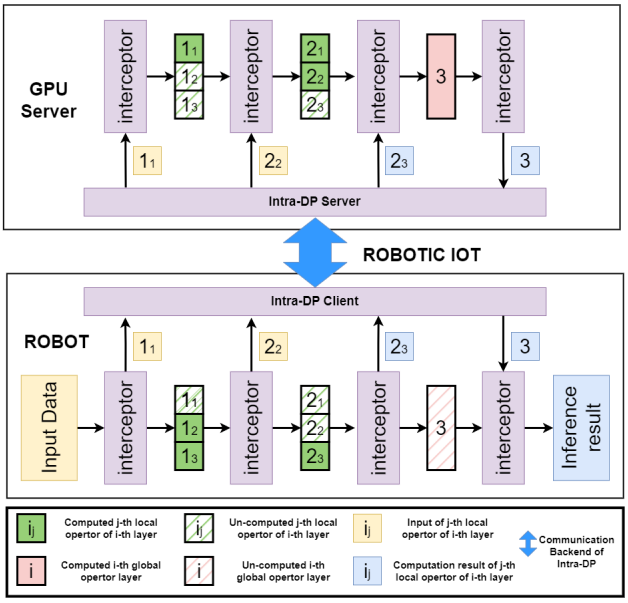


Figure 4. Architecture of Intra-DP. The core components of Intra-DP are highlighted in purple. Intra-DP adopts the same scheduling scheme as in Fig. 3.

with the original model inference process on the robot, Intra-DP only increases the time cost of interceptors, which is the time cost of splitting the input tensor and combining the output tensor. The time cost of splitting the input tensor is negligible because the data transfer can be completed through the backend processes of the Intra-DP client and server while the local operators assigned to be executed on the robot and GPU server continue to perform subsequent layer calculations. The time cost of combining the output tensor is mainly bound by the time when the device on the other side completes the corresponding computation and transmission, causing prolonged waiting time. Intra-DP formulates such waiting time into the nonlinear optimization problem in its LOSS, minimizing the waiting time and implementing scheduling schemes on local operators with a higher degree of parallelism. In this way, Intra-DP only increases negligible extra time on system cost and achieves faster inference via LOP and LOSS.

To address frequent fluctuations in real-world wireless networks of robotic IoT, Intra-DP generates optimal local operator scheduling strategies for the DNN model under different bandwidth conditions in advance. During inference, Intra-DP predicts the network bandwidth using mature tools [62] in the field of wireless transmission and adopts the corresponding scheduling strategy based on the predicted bandwidth. To ensure that Intra-DP can flexibly switch among various scheduling strategies, it keeps a copy of the model on the robot at the GPU server (Fig. 4), avoiding unnecessary transmission when migrating the parameters of local operators between robots and the GPU server. It is important to note that the model inference time, typically tens or hundreds of milliseconds, is finer (smaller) than the granularity (or frequency) of bandwidth fluctuation in real-world robotic IoT networks, as shown in Fig. 2. Therefore, we assume that the network bandwidth of robotic IoT during each inference is stable, while the network bandwidth for different inferences may differ.

4 Detailed Design

4.1 Local Operator Parallelism

LOP guarantees the correctness of inference results by determining the correct input required for different local operators based on their calculation characteristics and processes. We summarize three classes of local operators common in models used on mobile devices:

- Element-wised local operator. This class of operators compute each element of the input tensor separately, requiring only the corresponding element from the input tensor to perform the calculation. They are widely used in activation functions such as ReLU [7], Sigmoid [61], SiLU [19]. However, it is important to note that some activation functions, like softmax [31], require all elements for computation and are not considered local operators, but global operators.
- Block-wised local operator. This class of operators require a block at the corresponding position in the input tensor and are widely used in layers associated with convolution, such as convolution [34], maxpool [47]. The size of the input blocks is determined by the parameters set by the corresponding layer [41], including the size of the convolution kernel, padding, and dilation.
- Row-wised local operator. This class of operators requires rows of the input tensor and are widely used in layers associated with matrix operations, such as addition [63] and multiplication [12]. The rows required for computation, ensuring that the correct input is obtained for each local operator to perform its respective calculation, are determined by the matrix calculation principles as following:

$$\begin{pmatrix} a_1 \\ \vdots \\ a_m \end{pmatrix} \times \begin{pmatrix} b_1 & \cdots & b_n \end{pmatrix} = \begin{pmatrix} c_{11} & \cdots & c_{1n} \\ \vdots & \ddots & \vdots \\ c_{m1} & \cdots & c_{mn} \end{pmatrix}$$

Row-wised local operators split the input matrix and keep a copy of the layer parameter matrix on different devices, reducing transmission volume and avoiding the synchronization needed to combine calculation results from different devices. This is in contrast to TP, which splits the layer parameter matrix and transfers a copy of the input matrix to different devices. The calculation result of row a_1 is $(c_{11} \cdots c_{1n})$, which is also a row and can be directly computed by the next

matrix operation layer. And LOP treats matrices with only one row as global operators.

After obtaining the required input for each local operator and the corresponding input position in the previous layer through the above analysis, LOP determines which local operators need to be computed in the previous layer to obtain the input for the current local operator. This establishes the dependency between local operators, which should be considered when scheduling local operators as LOSS. And we leave the support for additional types of local operators as future work.

4.2 Local Operator Scheduling Strategy

LOSS formulates the problem of scheduling local operators as a nonlinear optimization problem, which is modeling as follows:

First of all, we identify the complete time of i-th layer on robot on robots as T^i_{robot} and that on GPU server as T^i_{server}.

Then LOP schedules the computation and transmission of each local operator based on the solution obtained via the differential evolution algorithm [42].

4.3 Algorithms of Intra-DP

Algorithm 1: Local_Operator_Scheduling_Strategy

Input: Cuda kernel function called by the corresponding operator $func$ and the required parameters $args$
Output: The execution result ret
Data: inference operator sequence $IOS = \emptyset$

if $IOS.empty()$ **then**
  // recorder
  $SendRPCtoServer(func, args)$
  $IOS = DataDependencySearch(func, args)$
  $ret = GetRPCExecutionResult()$
  $RecordReturn(ret)$
end
else
  // replayer on robot
  if $func == IOS.start()["func"]$ **then**
    $ret = StartRRTO(args, IOS)$
    // start a new inference
  end
  else if $func == IOS.end()["func"]$ **then**
    $ret = WaitingForRRTO()$
    // Waiting for the final computation result
  end
  else
    $ret = IOS.find(func)["ret"]$
  end
end
return ret

	inference	transmission	standby
Power (W)	13.35	4.25	4.04

Table 3. Power consumption (Watt) of our robot in different states.

5 Implementation

Intra-DP is easy to use and requires only three lines of code to apply to existing ML applications, as shown in Fig. 5.

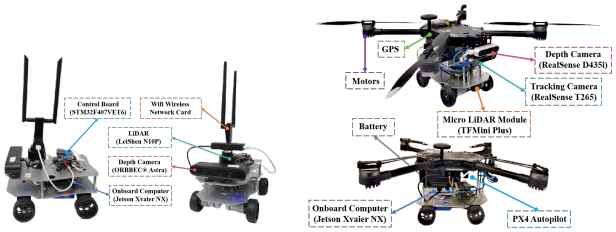
```
# Other part of pytorch as usual

# Import package of Intra-DP
import intraDP

# Define a VGG19 model as usual
vgg19 = VGG19().to(device)

# Apply Intra-DP
IDP = intraDP(ip = "192.168.50.1")
IDP.start_client(model = vgg19)

# Run model for inference as usual
result = vgg19(input)
```

Figure 5. An example of applying Intra-DP to a VGG19 [45] model, where "192.168.50.1" is the IP address of the GPU server.

6 Evaluation

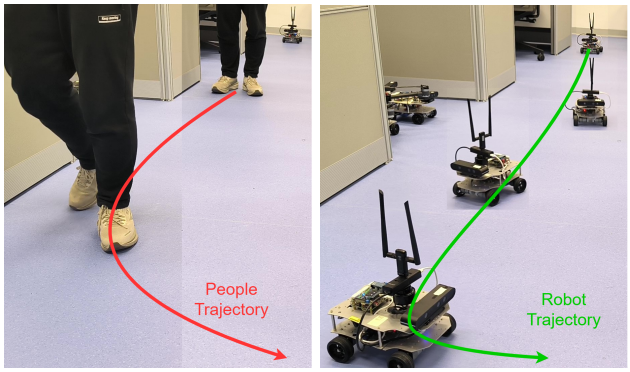
Testbed. The evaluation was conducted on a custom four-wheeled robot (Fig 6a), and a custom air-ground robot(Fig 6b). They are equipped with a Jetson Xavier NX [37] 8G onboard computer that is capable of AI model inference with local computation resources. The system runs Ubuntu 20.04 with ROS Noetic 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.

Tab. 3 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



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

Figure 6. The detailed composition of the robot platforms



(a) Targeted people **(b)** Robot moving trajectory

Figure 7. A real-time people-tracking robotic application on our robot based on a well-known human pose estimation ML model, Kapao [33].

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.

Experiment Environments. 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 robot and the GPU server in indoors and outdoors scenarios are shown in Fig. 2.

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 7), and AGRNav, an autonomous navigation application on our air-ground robot (Fig 8). 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,

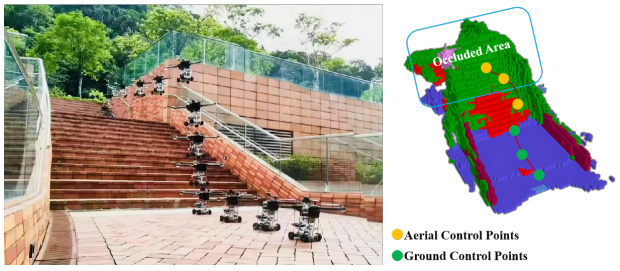


Figure 8. By predicting occlusions in advance, AGRNav [50] gains an accurate perception of the environment and avoids collisions, resulting in efficient and energy-saving paths.

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 [46], ResNet [48], VGGNet [45], ConvNeXt [52], RegNet [57].

Baselines. We selected two SOTA pipeline parallelism methods as baselines: DSCCS [26], aimed at accelerating inference, and SPSO-GA [5], 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.

6.1 Inference Time

Kapao. From the results in the upper part of Tab. 4, 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 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. 4. 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.6.2.

Notice that the large standard deviation in transmission time in outdoors in both offloading systems indicates that

Model(number of parameters)	Local computation time (s)	Environment	Transmission time (s)			Inference time (s)		
			DSCCS	SPSO-GA	Intra-DP	DSCCS	SPSO-GA	Intr
kapao(77M)	0.78(±0.23)	indoors	0.228(±0.176)	0.235(±0.164)	0.259(±0.181)	0.343(±0.192)	0.311(±0.168)	0.264(
		outdoors	0.087(±0.178)	0.35(±1.045)	0.385(±1.15)	0.696(±0.125)	0.434(±1.046)	0.369(
agrnav(0M)	1.12(±0.11)	indoors	0.266(±0.166)	0.239(±0.149)	0.263(±0.164)	0.433(±0.134)	0.427(±0.12)	0.365(
		outdoors	0.263(±0.843)	0.218(±0.796)	0.24(±0.875)	0.512(±0.779)	0.536(±0.72)	0.456(

Table 4. Average transmission time (Second), inference time (Second), percentage that transmission time accounts for of 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	DSCCS	SPSO-GA	Intra-DP	Local	DSCCS	SPSO-GA	Intra
kapao(77M)	indoors	15.04(±0.64)	7.03(±3.57)	5.92(±2.18)	5.03(±1.87)	15.03(±0.63)	2.41(±1.35)	1.84(±1.0)	1.56(±
	outdoors	15.04(±0.64)	14.15(±1.71)	5.89(±2.3)	5.02(±1.97)	1.56(±0.85)	9.85(±1.77)	2.56(±6.17)	2.18(±
agrnav(0.84M)	indoors	10.26(±1.58)	6.6(±1.95)	6.74(±2.03)	5.74(±1.78)	10.82(±1.44)	2.86(±0.88)	2.88(±0.81)	2.45(±
	outdoors	10.26(±1.58)	7.36(±2.34)	7.91(±2.45)	6.71(±2.1)	10.82(±1.44)	3.77(±5.74)	4.25(±5.7)	3.6(±

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"

bandwidth fluctuated more frequently and more fiercely outdoors compared with indoors, which complies with Fig. 2. Additionally, the lower average bandwidth for outdoors scenarios (see Sec.2.1) results in increased transmission and inference times relative to indoor scenarios.

6.2 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. 9.

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. 9a and Fig. 9b, both SPSO-GA and DSCCS tend to allocate more layers on the robot for AGRNav. When comparing indoor and outdoor scenarios in Fig. 9, it is evident that higher bandwidth leads to more layers being scheduled on GPU server. Additionally, when comparing SPSO-GA and DSCCS in Fig. 9, 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.

6.3 Energy Consumption

Kapao. From the results in the upper part of Tab. 5, 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. 9a. 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. 5, DSCCS consumed 61.21% and 22.92% more energy per second than SPSO-GA indoors and outdoors (Tab. 5), 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.

6.4 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. 6 and Tab. 7), 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

Model(number of parameters)	Local computation time (s)	Environment	Transmission time (s) DSCCS	SPSO-GA	Intra-DP	Inference time (s) DSCCS	SPSO-GA
MobileNet_V3_Small(2M)	0.025(±0.005)	indoors	0.048(±0.035)	0.011(±0.032)	0.012(±0.035)	0.059(±0.036)	0.042(±0.725)
		outdoors	0.04(±0.161)	0.008(±0.204)	0.009(±0.224)	0.054(±0.133)	0.034(±0.573)
RegNet_X_3_2GF(15M)	0.056(±0.004)	indoors	0.049(±0.035)	0.026(±0.024)	0.028(±0.027)	0.063(±0.037)	0.055(±0.212)
		outdoors	0.046(±0.169)	0.021(±0.12)	0.024(±0.131)	0.067(±0.136)	0.063(±0.651)
ResNet101(44M)	0.078(±0.002)	indoors	0.049(±0.035)	0.033(±0.023)	0.036(±0.025)	0.064(±0.037)	0.058(±0.017)
		outdoors	0.048(±0.172)	0.0(±0.0)	0.0(±0.0)	0.072(±0.135)	0.0(±0.0)
ConvNeXt_Base(88M)	0.139(±0.002)	indoors	0.049(±0.036)	0.041(±0.041)	0.045(±0.045)	0.063(±0.037)	0.061(±0.032)
		outdoors	0.052(±0.178)	0.032(±0.124)	0.036(±0.137)	0.079(±0.141)	0.072(±0.093)
VGG19_BN(143M)	0.097(±0.002)	indoors	0.049(±0.036)	0.034(±0.024)	0.038(±0.027)	0.058(±0.036)	0.054(±0.021)
		outdoors	0.05(±0.172)	0.031(±0.117)	0.034(±0.129)	0.07(±0.134)	0.061(±0.085)
ConvNeXt_Large(197M)	0.291(±0.002)	indoors	0.048(±0.035)	0.049(±0.036)	0.054(±0.039)	0.068(±0.036)	0.068(±0.036)
		outdoors	0.056(±0.189)	0.037(±0.136)	0.04(±0.15)	0.092(±0.149)	0.09(±0.122)

Table 6. Average transmission time (Second), inference time (Second), percentage that transmission time accounts for of inference time and their standard deviation (±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)	Energy consumption(J) per inference	Environment	Power consumption(W) Local	SPSO-GA	DSCCS	Local	SPSO-GA	DS
MobileNet_V3_Small(2M)	6.463(±0.121)	indoors	5.21(±0.304)	5.51(±0.645)	4.965(±0.732)	0.309(±0.188)	0.232(	
		outdoors	5.328(±0.611)	5.792(±0.742)	5.222(±0.832)	0.288(±0.708)	0.195	
RegNet_X_3_2GF(15M)	8.889(±0.195)	indoors	5.108(±0.272)	6.598(±1.196)	5.921(±1.223)	0.323(±0.19)	0.365(	
		outdoors	5.83(±1.568)	7.034(±1.601)	6.313(±1.565)	0.393(±0.793)	0.445(	
ResNet101(44M)	9.775(±0.256)	indoors	5.077(±0.261)	6.312(±1.743)	5.664(±1.629)	0.324(±0.188)	0.364(	
		outdoors	6.051(±1.904)	4.416(±0.339)	3.972(±0.471)	0.438(±0.817)	0.0(	
ConvNeXt_Base(88M)	10.635(±0.246)	indoors	5.108(±0.25)	6.09(±1.635)	5.512(±1.583)	0.324(±0.189)	0.372(	
		outdoors	6.191(±2.098)	7.385(±2.534)	6.663(±2.359)	0.487(±0.87)	0.531(	
VGG19_BN(143M)	10.596(±0.372)	indoors	5.152(±0.269)	6.517(±2.064)	5.861(±1.937)	0.301(±0.183)	0.353(	
		outdoors	6.276(±2.203)	7.451(±2.564)	6.708(±2.389)	0.44(±0.843)	0.458(	
ConvNeXt_Large(197M)	10.732(±0.438)	indoors	5.114(±0.234)	5.268(±0.197)	4.745(±0.463)	0.349(±0.183)	0.36(	
		outdoors	6.332(±2.337)	7.497(±2.748)	6.754(±2.571)	0.583(±0.942)	0.674(	

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

when using PP. The inherent transmission overhead of PP's scheduling mechanism significantly wastes both inference time and energy.

6.5 Lessons learned

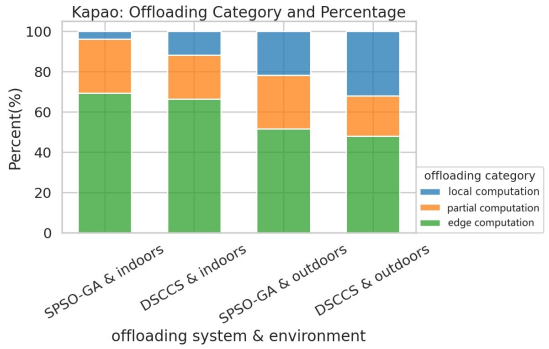
Global optimal solution. for nonlinear optimization problem

Wireless bandwidth prediction.

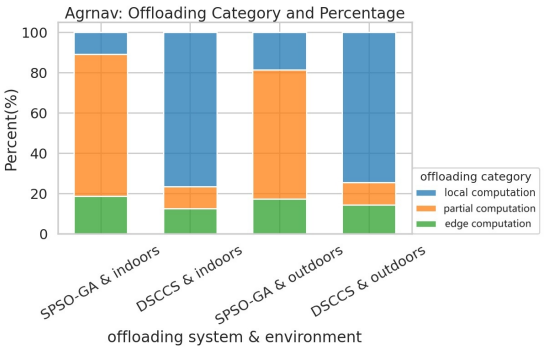
Distributed inference system for multiple robots.

7 Conclusion

In this paper, we present Intra-DP, a high-performance distributed inference system optimized for robotic IoT networks. By breaking up the granularity of model inference into local operators via LOP and applying adaptive scheduling to the computation and transmission of each local operator via LOSS, Intra-DP dramatically reduces the transmission overhead in existing distributed inference on robotic IoT by overlapping the computation and transmission phases within the same inference task, achieving fast and energy-efficient distributed inference. We envision that the fast and energy-efficient inference of Intra-DP will foster the deployment of diverse robotic tasks on real-world robots in the field.



(a) Categories and percentage of various scheduling for Kapao



(b) Categories and percentage of various scheduling for AGRNav

Figure 9. 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.

References

[1] [n. d.]. iPerf - Download iPerf3 and original iPerf pre-compiled binaries. https://iperf.fr/iperf-download.php

[2] Toni Adame, Marc Carrascosa-Zamacois, and Boris Bellalta. 2021. Time-sensitive networking in IEEE 802.11 be: On the way to low-latency WiFi 7. *Sensors* 21, 15 (2021), 4954.

[3] Majid Altamimi, Atef Abdrabou, Kshirasagar Naik, and Amiya Nayak. 2015. Energy cost models of smartphones for task offloading to the cloud. *IEEE Transactions on Emerging Topics in Computing* 3, 3 (2015), 384–398.

[4] Anh-Quan Cao and Raoul de Charette. 2022. Monoscene: Monocular 3d semantic scene completion. In *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition*. 3991–4001.

[5] Xing Chen, Jianshan Zhang, Bing Lin, Zheyi Chen, Katinka Wolter, and Geyong Min. 2021. 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.

[6] Daniel Crankshaw, Gur-Eyal Sela, Xiangxi Mo, Corey Zumar, Ion Stoica, Joseph Gonzalez, and Alexey Tumanov. 2020. InferLine: latency-aware provisioning and scaling for prediction serving pipelines. In *Proceedings of the 11th ACM Symposium on Cloud Computing*. 477–491.

[7] Ingrid Daubechies, Ronald DeVore, Simon Foucart, Boris Hanin, and Guergana Petrova. 2022. Nonlinear approximation and (deep) ReLU networks. *Constructive Approximation* 55, 1 (2022), 127–172.

[8] Alexandre Défossez, Yossi Adi, and Gabriel Synnaeve. 2021. Differentiable model compression via pseudo quantization noise. *arXiv preprint arXiv:2104.09987* (2021).

[9] Ming Ding, Peng Wang, David López-Pérez, Guoqiang Mao, and Zihuai Lin. 2015. Performance impact of LoS and NLoS transmissions in dense cellular networks. *IEEE Transactions on Wireless Communications* 15, 3 (2015), 2365–2380.

[10] Khalid Elgazzar, Patrick Martin, and Hossam S Hassanein. 2014. Cloud-assisted computation offloading to support mobile services. *IEEE Transactions on Cloud Computing* 4, 3 (2014), 279–292.

[11] Zhou Fang, Tong Yu, Ole J Mengshoel, and Rajesh K Gupta. 2017. Qos-aware scheduling of heterogeneous servers for inference in deep neural networks. In *Proceedings of the 2017 ACM on Conference on Information and Knowledge Management*. 2067–2070.

[12] Kayvon Fatahalian, Jeremy Sugerman, and Pat Hanrahan. 2004. Understanding the efficiency of GPU algorithms for matrix-matrix multiplication. In *Proceedings of the ACM SIGGRAPH/EUROGRAPHICS conference on Graphics hardware*. 133–137.

[13] Ștefan Gheorghe and Mihai Ivanovici. 2021. Model-based weight quantization for convolutional neural network compression. In *2021 16th International Conference on Engineering of Modern Electric Systems (EMES)*. IEEE, 1–4.

[14] Cheng Gong, Yao Chen, Ye Lu, Tao Li, Cong Hao, and Deming Chen. 2020. VecQ: Minimal loss DNN model compression with vectorized weight quantization. *IEEE Trans. Comput.* 70, 5 (2020), 696–710.

[15] Jianping Gou, Baosheng Yu, Stephen J Maybank, and Dacheng Tao. 2021. Knowledge distillation: A survey. *International Journal of Computer Vision* 129, 6 (2021), 1789–1819.

[16] Vinayak Honkote, Dileep Kurian, Sriram Muthukumar, Dibyendu Ghosh, Satish Yada, Kartik Jain, Bradley Jackson, Ilya Klotchkov, Mallikarjuna Rao Nimmagadda, Shreela Dattawadkar, et al. 2019. 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)*. IEEE, 48–50.

[17] Chuang Hu, Wei Bao, Dan Wang, and Fengming Liu. 2019. Dynamic adaptive DNN surgery for inference acceleration on the edge. In *IEEE INFOCOM 2019-IEEE Conference on Computer Communications*. IEEE, 1423–1431.

[18] Yang Hu, Connor Imes, Xuanang Zhao, Souvik Kundu, Peter A Beerel, Stephen P Crago, and John Paul Walters. 2022. Pipeedge: Pipeline parallelism for large-scale model inference on heterogeneous edge devices. In *2022 25th Euromicro Conference on Digital System Design (DSD)*. IEEE, 298–307.

[19] Glenn Jocher, Alex Stoken, Jirka Borovec, Liu Changyu, Adam Hogan, Ayush Chaurasia, Laurentiu Diaconu, Francisco Ingham, Adrien Colmagro, Hu Ye, et al. 2021. ultralytics/yolov5: v4. 0-nn. SiLU () activations, Weights & Biases logging, PyTorch Hub integration. *Zenodo* (2021).

[20] K J Joseph, Salman Khan, Fahad Shahbaz Khan, and Vineeth N Balasubramanian. 2021. Towards Open World Object Detection. In *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition (CVPR)*. 5830–5840.

[21] Yiping Kang, Johann Hauswald, Cao Gao, Austin Rovinski, Trevor Mudge, Jason Mars, and Lingjia Tang. 2017. Neurosurgeon: Collaborative intelligence between the cloud and mobile edge. *ACM SIGARCH Computer Architecture News* 45, 1 (2017), 615–629.

[22] Nam Sung Kim, Todd Austin, David Baauw, Trevor Mudge, Krisztián Flautner, Jie S Hu, Mary Jane Irwin, Mahmut Kandemir, and Vijaykrishnan Narayanan. 2003. Leakage current: Moore's law meets static power. *computer* 36, 12 (2003), 68–75.

[23] Ang Li, Shuaiwen Leon Song, Jieyang Chen, Jiajia Li, Xu Liu, Nathan R Tallent, and Kevin J Barker. 2019. Evaluating modern gpu interconnect: Pcie, nvlink, nv-sli, nvswitch and gpudirect. *IEEE Transactions on Parallel and Distributed Systems* 31, 1 (2019), 94–110.

[24] Qingbiao Li, Fernando Gama, Alejandro Ribeiro, and Amanda Prorok. 2020. Graph neural networks for decentralized multi-robot path planning. In *2020 IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS)*. IEEE, 11785–11792.

[25] Yiming Li, Zhiding Yu, Christopher Choy, Chaowei Xiao, Jose M Alvarez, Sanja Fidler, Chen Feng, and Anima Anandkumar. 2023. Voxformer: Sparse voxel transformer for camera-based 3d semantic scene completion. In *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition*. 9087–9098.

[26] Huanghuang Liang, Qianlong Sang, Chuang Hu, Dazhao Cheng, Xiaobo Zhou, Dan Wang, Wei Bao, and Yu Wang. 2023. DNN surgery: Accelerating DNN inference on the edge through layer partitioning. *IEEE transactions on Cloud Computing* (2023).

[27] Bing Lin, Yinhao Huang, Jianshan Zhang, Junqin Hu, Xing Chen, and Jun Li. 2019. Cost-driven off-loading for DNN-based applications over cloud, edge, and end devices. *IEEE Transactions on Industrial Informatics* 16, 8 (2019), 5456–5466.

[28] Tao Lin, Lingjing Kong, Sebastian U Stich, and Martin Jaggi. 2020. Ensemble distillation for robust model fusion in federated learning. *Advances in Neural Information Processing Systems* 33 (2020), 2351–2363.

[29] Ruofeng Liu and Nakjung Choi. 2023. 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.

[30] Shuai Liu, Xin Li, Huchuan Lu, and You He. 2022. Multi-Object Tracking Meets Moving UAV. In *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition (CVPR)*. 8876–8885.

[31] Weiyang Liu, Yandong Wen, Zhiding Yu, and Meng Yang. 2016. Large-margin softmax loss for convolutional neural networks. *arXiv preprint arXiv:1612.02295* (2016).

[32] Antoni Masiukiewicz. 2019. 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.

[33] William McNally, Kanav Vats, Alexander Wong, and John McPhee. 2022. Rethinking keypoint representations: Modeling keypoints and poses as objects for multi-person human pose estimation. In *European Conference on Computer Vision*. Springer, 37–54.

[34] Thaha Mohammed, Carlee Joe-Wong, Rohit Babbar, and Mario Di Francesco. 2020. Distributed inference acceleration with adaptive DNN partitioning and offloading. In *IEEE INFOCOM 2020-IEEE Conference on Computer Communications*. IEEE, 854–863.

[35] Deepak Narayanan, Mohammad Shoeybi, Jared Casper, Patrick LeGresley, Mostofa Patwary, Vijay Korthikanti, Dmitri Vainbrand, Prethvi Kashinkunti, Julie Bernauer, Bryan Catanzaro, et al. 2021. 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*. 1–15.

[36] Mohammad Noormohammadpour and Cauligi S Raghavendra. 2017. Datacenter traffic control: Understanding techniques and tradeoffs. *IEEE Communications Surveys & Tutorials* 20, 2 (2017), 1492–1525.

[37] NVIDIA. 2024. The World's Smallest AI Supercomputer. https://www.nvidia.com/en-us/autonomous-machines/embedded-systems/jetson-xavier-series/.

[38] Takeshi Ohkawa, Kazushi Yamashina, Hitomi Kimura, Kanemitsu Ootsu, and Takashi Yokota. 2018. FPGA components for integrating FPGAs into robot systems. *IEICE TRANSACTIONS on Information and Systems* 101, 2 (2018), 363–375.

[39] Yuanteng Pei, Matt W Mutka, and Ning Xi. 2013. Connectivity and bandwidth-aware real-time exploration in mobile robot networks. *Wireless Communications and Mobile Computing* 13, 9 (2013), 847–863.

[40] pytorch. 2024. pytroch. https://pytorch.org/.

[41] pytorch. 2024. pytroch. https://pytorch.org/docs/stable/generated/torch.nn.Conv2d.html.

[42] A Kai Qin, Vicky Ling Huang, and Ponnuthurai N Suganthan. 2008. Differential evolution algorithm with strategy adaptation for global numerical optimization. *IEEE transactions on Evolutionary Computation* 13, 2 (2008), 398–417.

[43] Yi Ren, Chih-Wei Tung, Jyh-Cheng Chen, and Frank Y Li. 2018. Proportional and preemption-enabled traffic offloading for IP flow mobility: Algorithms and performance evaluation. *IEEE Transactions on Vehicular Technology* 67, 12 (2018), 12095–12108.

[44] Nurul I Sarkar and Osman Mussa. 2013. The effect of people movement on Wi-Fi link throughput in indoor propagation environments. In *IEEE 2013 Tencon-Spring*. IEEE, 562–566.

[45] Karen Simonyan and Andrew Zisserman. 2015. Very Deep Convolutional Networks for Large-Scale Image Recognition. arXiv:1409.1556 [cs.CV]

[46] Debjyoti Sinha and Mohamed El-Sharkawy. 2019. Thin mobilenet: An enhanced mobilenet architecture. In *2019 IEEE 10th annual ubiquitous computing, electronics & mobile communication conference (UEMCON)*. IEEE, 0280–0285.

[47] Luna Sun, Zhenxue Chen, QM Jonathan Wu, Hongjian Zhao, Weikai He, and Xinghe Yan. 2021. AMPNet: Average-and max-pool networks for salient object detection. *IEEE Transactions on Circuits and Systems for Video Technology* 31, 11 (2021), 4321–4333.

[48] Sasha Targ, Diogo Almeida, and Kevin Lyman. 2016. Resnet in resnet: Generalizing residual architectures. *arXiv preprint arXiv:1603.08029* (2016).

[49] Hao Wang, Sreeram Potluri, Miao Luo, Ashish Kumar Singh, Sayantan Sur, and Dhabaleswar K Panda. 2011. MVAPICH2-GPU: optimized GPU to GPU communication for InfiniBand clusters. *Computer Science-Research and Development* 26, 3 (2011), 257–266.

[50] Junming Wang, Zekai Sun, Xiuxian Guan, Tianxiang Shen, Zongyuan Zhang, Tianyang Duan, Dong Huang, Shixiong Zhao, and Heming Cui. 2024. AGRNav: Efficient and Energy-Saving Autonomous Navigation for Air-Ground Robots in Occlusion-Prone Environments. In *IEEE International Conference on Robotics and Automation (ICRA)*.

[51] Lin Wang and Kuk-Jin Yoon. 2021. Knowledge distillation and student-teacher learning for visual intelligence: A review and new outlooks. *IEEE transactions on pattern analysis and machine intelligence* 44, 6 (2021), 3048–3068.

[52] Sanghyun Woo, Shoubhik Debnath, Ronghang Hu, Xinlei Chen, Zhuang Liu, In So Kweon, and Saining Xie. 2023. Convnext v2: Co-designing and scaling convnets with masked autoencoders. In *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition*. 16133–16142.

[53] Huaming Wu, William J Knottenbelt, and Katinka Wolter. 2019. An efficient application partitioning algorithm in mobile environments. *IEEE Transactions on Parallel and Distributed Systems* 30, 7 (2019), 1464–1480.

[54] Tianyu Wu, Shizhu He, Jingping Liu, Siqi Sun, Kang Liu, Qing-Long Han, and Yang Tang. 2023. A brief overview of ChatGPT: The history, status quo and potential future development. *IEEE/CAA Journal of Automatica Sinica* 10, 5 (2023), 1122–1136.

[55] Zhaoyang Xia, Youquan Liu, Xin Li, Xinge Zhu, Yuexin Ma, Yikang Li, Yuenan Hou, and Yu Qiao. 2023. SCPNet: Semantic Scene Completion on Point Cloud. In *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition*. 17642–17651.

[56] Yecheng Xiang and Hyoseung Kim. 2019. Pipelined data-parallel CPU/GPU scheduling for multi-DNN real-time inference. In *2019 IEEE*

Real-Time Systems Symposium (RTSS). IEEE, 392–405.

[57] Jing Xu, Yu Pan, Xinglin Pan, Steven Hoi, Zhang Yi, and Zenglin Xu. 2022. RegNet: self-regulated network for image classification. *IEEE Transactions on Neural Networks and Learning Systems* (2022).

[58] Min Xue, Huaming Wu, Guang Peng, and Katinka Wolter. 2021. DDPQN: An efficient DNN offloading strategy in local-edge-cloud collaborative environments. *IEEE Transactions on Services Computing* 15, 2 (2021), 640–655.

[59] Xinlei Yang, Hao Lin, Zhenhua Li, Feng Qian, Xingyao Li, Zhiming He, Xudong Wu, Xianlong Wang, Yunhao Liu, Zhi Liao, et al. 2022. Mobile access bandwidth in practice: Measurement, analysis, and implications. In *Proceedings of the ACM SIGCOMM 2022 Conference*. 114–128.

[60] Yang Yang, Li Juntao, and Peng Lingling. 2020. Multi-robot path planning based on a deep reinforcement learning DQN algorithm. *CAAI Transactions on Intelligence Technology* 5, 3 (2020), 177–183.

[61] Xinyou Yin, JAN Goudriaan, Egbert A Lantinga, JAN Vos, and Huub J Spiertz. 2003. A flexible sigmoid function of determinate growth. *Annals of botany* 91, 3 (2003), 361–371.

[62] Chaoqun Yue, Ruofan Jin, Kyoungwon Suh, Yanyuan Qin, Bing Wang, and Wei Wei. 2017. LinkForecast: Cellular link bandwidth prediction in LTE networks. *IEEE Transactions on Mobile Computing* 17, 7 (2017), 1582–1594.

[63] Anthony Zee. 1996. Law of addition in random matrix theory. *Nuclear Physics B* 474, 3 (1996), 726–744.

[64] Yonghao Zhuang, Hexu Zhao, Lianmin Zheng, Zhuohan Li, Eric Xing, Qirong Ho, Joseph Gonzalez, Ion Stoica, and Hao Zhang. 2023. On optimizing the communication of model parallelism. *Proceedings of Machine Learning and Systems* 5 (2023).