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

Anonymous Author(s)
Submission Id: 273

Abstract

Deep Neural Networks (DNNs) have been widely employed in various robotic applications, necessitating fast and energy-efficient inference when deployed on robots. Distributed inference, which leverages a GPU server to accelerate inference on the robot via the Internet of Things for robots (robotic IoT), has emerged as a promising approach to achieve this goal. However, the three main parallel computing techniques used in modern data centers, namely data parallelism (DP), tensor parallelism (TP), and pipeline parallelism (PP), are ill-suited for robotic IoT. DP and TP are inapplicable due to the real-time demands of robotic applications and the limited bandwidth of robotic IoT, while PP-based approaches still face significant transmission bottlenecks caused by the sequential execution of DNN operations.

To address these challenges, we present Intra-DP, a high-performance distributed inference system optimized for DNN inference on robotic IoT. Intra-DP employs a novel parallel computing technique based on local operators (i.e., operators whose minimum unit input is not the entire input tensor, such as the convolution kernel). By cutting their operations into several independent sub-operations and overlapping the computation and transmission of different sub-operations through parallel execution, Intra-DP significantly mitigates transmission bottlenecks in robotic IoT, achieving fast and energy-efficient inference. The evaluation demonstrates that Intra-DP reduced inference time by 5% to 61% and energy consumption per inference by up to 72% compared to baselines.

1 Introduction

Deep Neural Networks (DNNs) have been widely employed in various robotic tasks, leading to remarkable achievements in fundamental areas such as object detection [25, 37, 42], robotic control [30, 60, 69], and environmental perception [6, 31, 64]. However, deploying these applications onto real-world robots presents new challenges due to the demand for swift responses and the limited battery capacity of robots. A straightforward approach of placing an entire model on robots with computing accelerators (e.g., GPU [47], FPGA [48], SoC [19]) introduces significant energy consumption, with an average of 70% energy consumption observed on our robot.

Distributed inference, which leverages a GPU server (e.g., edge device with powerful GPU) to accelerate inference on the robot via the Internet of Things for robots (robotic IoT), has emerged as a promising approach to provide fast and energy-efficient inference. Various parallel methods have been proposed and proven efficient when deployed in modern data centers [21, 65, 75]. The three main parallel computing techniques in data centers are data parallelism (DP), which replicates the model across devices and allows each replica to handle a mini-batch (i.e., a subset of the input data set); tensor parallelism (TP), which splits a single layer of the model over devices; and pipeline parallelism (PP), which distributes different layers of the model across devices (layer partitioning) and pipelines the inference to minimize devices' idling time (pipeline execution).

However, all three parallel computing techniques in data centers are ill-suited for robotic IoT. Specifically, DP is generally limited by the total batch size [44]. In data centers, 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). As a result, DP is inapplicable for robotic applications.

Meanwhile, TP generally allows concurrent computation on different parts of a model layer across devices but requires an all-reduce communication [75] to combine computation results. Thus, TP potentially entails significant communication overhead (Fig. 1). Robots must prioritize seamless mobility and primarily depend on wireless connections with limited bandwidth (Sec. 2.1), making all-reduce synchronization an unacceptable overhead. Our evaluation in Sec. 2.2 confirms that such communication overhead in robotic IoT dramati-

cally slows down the entire inference process, resulting in inference times up to 143.9× slower than local computation.

Besides DP and TP, PP that conducts layer partitioning and pipeline execution seems a promising approach for robotic applications and has been implemented in various existing methods in robotic IoT [5, 8, 20, 23, 32, 43, 67, 74]. However, all these methods face significant transmission bottlenecks in robotic IoT. First, PP typically involves three sequential phases according to its layer partitioning: computing early layers on robots, transmitting intermediate results, and completing inference on GPU servers. Our experiments demonstrate that the transmission overhead due to the limited bandwidth of robotic IoT can become a substantial bottleneck despite advanced layer partitioning strategies [8, 32] (e.g., accounting for up to 44.2% of inference time). Second, pipeline execution (i.e., overlapping computation and transmission phases across multiple inference tasks) cannot effectively mitigate the overhead since it only enhances throughput rather than reducing the completion time of a single inference task [9], which is critical to robotic applications.

The primary reason why existing methods suffer from significant transmission bottlenecks lies in their sequential execution of DNN operations. Each model layer contains one or several operators, and the DNN needs to execute the operators in a fixed order according to the model structure to obtain the correct inference result. Existing methods treat each calculation of an operator as an operation (e.g., the convolution operation for a convolution kernel), forcing them to execute DNN operations sequentially. As a result, this sequential execution of DNN operations constrains them to follow the three sequential phases of PP, which leads to significant transmission bottlenecks in robotic IoT.

Our key observation is that local operators, whose minimum unit input is not the entire input tensor, provide the opportunity for parallel execution of DNN operations. Unlike global operators that require the entire input tensor (e.g., softmax [38] requires it to calculate the corresponding probability distribution), local operators can be computed with smaller input units (the elements in the input tensor for ReLU [10] and the blocks in the input tensor for convolution [43]). We treat each calculation of the local operator based on its minimum input unit as a local operation. When traditional methods treat the calculation of a local operator as a whole DNN operation, we cut the calculation of the local operator into several independent local operations, enabling their parallel execution.

In this paper, we present Intra-DP (Intra-Data Parallel), a high-performance distributed inference system optimized for real-world robotic IoT. Intra-DP employs a novel parallel computing technique based on local operators and overlaps the computation and transmission of different local operations through parallel execution. This overlapping not only enables fast inferences by dramatically reducing idle time on the robot but also decreases overall energy consumption. This is because the main energy consumption during the idle time on the robot comes from chips like CPU, GPU, and memory (e.g., 95% energy consumption in our experiments) rather than the network cards. This new parallel computing technique only slightly increases the energy consumption of network cards during the computation phase while dramatically reducing idle time, ultimately leading to a decrease in overall energy consumption.

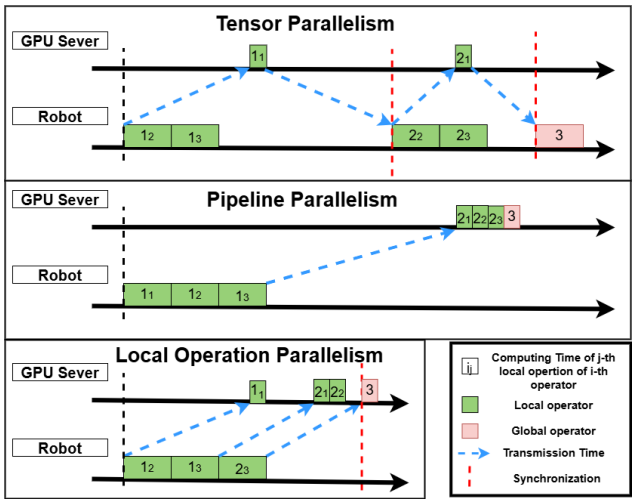


Figure 1: Workflow of TP, PP and LOP. In the three cases above, each local operator has three local operations with identical computation times on the robot and GPU server, as well as corresponding transmission time. See more details in Sec. 3.1.

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 operators. We propose Local Operation Parallelism (LOP) that determines data dependencies among local operations, where the output of one local operation serves as the input for another. For local operators, Intra-DP only changes the execution sequence of local operations and ensures each local operation gets the correct input (raw input or the output from the previous operation) according to its data dependency and the calculation logic of the local operator. For global operators, Intra-DP enforces a synchronization before them to combine the complete input, as TP's all-reduce communications do. By ensuring that all operations still receive the correct input, Intra-DP guarantees the calculation correctness of both local operators and global operators.

The second challenge is how to properly schedule the computation and transmission of each local operation to achieve fast and energy-efficient inference. Intra-DP addresses this challenge by introducing a novel Local Operation Scheduling Strategy (LOSS) that determines the partitioning of local operations between the robot and GPU server while considering the transmission cost of LOP. LOSS formulates this problem as a nonlinear optimization problem (Sec. 4.2) and schedules the computation and transmission of each local operation based on the solution obtained via the differential evolution algorithm [52]. Moreover, Intra-DP allows the re-

calculation of some local operations (letting them compute simultaneously on both the robot and GPU server, thus reducing the data transmission required by LOP), which greatly reduces the expensive transmission in robotic IoT by slightly increasing the redundant computation, ultimately optimizing the overall performance.

We implemented Intra-DP in PyTorch [50] and evaluated Intra-DP on our real-world robot under two typical real-world robotic applications [42, 60] and several models common to mobile devices on a larger scale [55, 56, 58, 62, 66]. We compared Intra-DP with four baselines under different real-world robotic IoT network environments (namely indoors and outdoors): local computation, which places entire model on the robot; all offload, which places entire model on GPU server; and two advanced PP methods: DSCCS [32], aimed at accelerating inference, and SPSO-GA [8], focused on optimizing energy consumption. Our evaluation shows that:

- Intra-DP is fast. Intra-DP reduced inference time by 5% to 61% compared to baselines under indoors and outdoors environments.
- Intra-DP is energy-efficient. Intra-DP reduced up to 72% energy consumption per inference compared to baselines, due to faster inference speed and limited-increased energy consumption per unit 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 robotic applications.

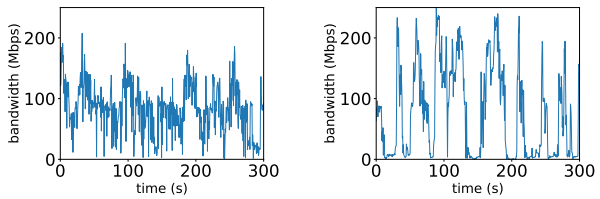
Our main contributions are LOP, a novel parallel computing technique based on local operators, and LOSS, a new scheduling strategy based on LOP optimized for DNN inference in robotic IoT. By leveraging LOP and LOSS, Intra-DP significantly mitigates transmission bottlenecks in robotic IoT by overlapping the computation and transmission of different local operations through parallel execution, 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, enabling the widespread adoption of advanced machine learning techniques in robotic applications. Intra-DP's code is released on https://github.com/nsdi25paper273/intraDP.

2 Background

2.1 Characteristics of Robotic IoT

In real-world scenarios, robots frequently navigate and move around to execute tasks such as search and exploration, relying on wireless networks that offer high mobility. However, these networks often have limited bandwidth, due to both the theoretical upper limit of wireless transmission technologies and the practical instability of wireless networks. For instance, the most advanced Wi-Fi technology, Wi-Fi 6, offers a maximum theoretical bandwidth of 1.2 Gbps for a single stream [36]. However, the limited hardware resources on robots often prevent them from fully utilizing the potential of Wi-Fi 6 [68]. Moreover, the actual available bandwidth of wireless networks is often reduced in practice due to various factors, such as the movement of devices [41, 49], occlusion by physical barriers [12, 54], and preemption of the wireless channel by other devices [2, 53].

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 utilized iperf [1] to saturate the wireless transmission between our robot and a base station, thereby measuring the real-time maximum wireless bandwidth capacity and recording these values every 0.1 seconds over a period of 5 minutes.



(a) Indoors (b) Outdoors

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

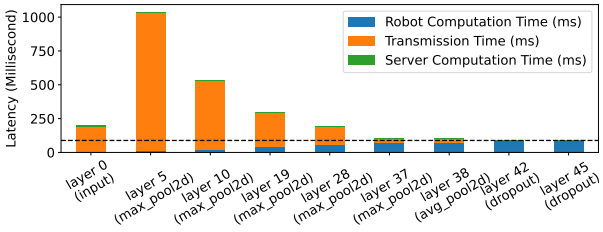
The results in Fig. 2 show average bandwidth capacities of 93 Mbps and 73 Mbps for indoor and outdoor scenarios, respectively. The outdoor environment exhibited higher instability, with bandwidth frequently dropping to extremely low values around 0 Mbps, due to the lack of walls to reflect wireless signals and the presence of obstacles like trees between communicating robots, resulting in fewer received signals compared to indoor environments. It is important to note that the bandwidth in real-world robotic IoT networks is relatively minimal compared to data center networks, where devices are equipped with high-speed networking technologies such as InfiniBand [46] or PCIe [29], offering bandwidths ranging from 40 Gbps to 500 Gbps.

2.2 Related distributed inference methods

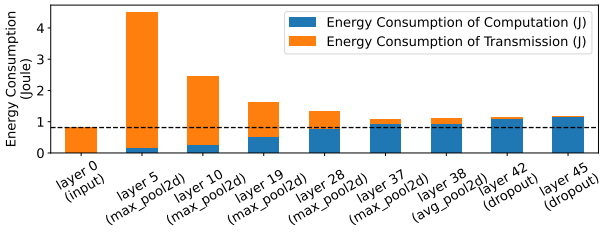
Tensor parallelism. We evaluated DINA, a state-of-the-art tensor parallelism (TP) method, on the same testbed as Sec. 6. The results in Tab. 1 show that transmission time takes up 49% to 94% of the total inference time due to all-reduce communication for each layer, making TP's inference time is 45.2× to 143.9× longer than local computation. Although TP has lower energy consumption per unit time (13.4% to 67.3% less than local computation), the extended transmission times

Model(number of parameters	Environment	Transmission time(s)	Inference time(s)	Energy consumption per unit time(W)	Energy consumption per inference(J)
MobileNet(2M)	Local	0.000(±0.000)	0.031(±0.0004)	6.05(±0.21)	0.30(±0.09)
	TP-indoors	0.698(±0.135)	1.400(±0.232)	5.24(±0.19)	7.33(±1.21)
	TP-outdoors	0.901(±0.778)	1.775(±1.370)	5.11(±0.28)	9.08(±7.00)
ResNet101(44M)	Local	0.000(±0.000)	0.065(±0.0005)	11.27(±0.51)	0.93(±0.19)
	TP-indoors	7.156(±3.348)	8.106(±3.403)	4.97(±0.16)	40.28(±16.91)
	TP-outdoors	8.470(±6.337)	9.356(±6.328)	4.90(±0.23)	45.8(±30.98)
VGG19(143M)	Local	0.000(±0.000)	0.063(±0.0002)	14.86(±0.43)	1.19(±0.18)
	TP-indoors	5.152(±4.873)	5.444(±4.831)	4.88(±0.29)	26.55(±23.56)
	TP-outdoors	5.407(±6.673)	5.759(±6.635)	4.87(±0.27)	28.06(±32.33)

Table 1: Average transmission time (Second), inference time (Second), energy consumption per unit time (Watt) and energy consumption per inference (Joule), along with the standard deviation ($\pm n$), for TP on different models in different environments. "Local" refers to place entire model on the robot.



(a) Inference Latency



(b) Energy consumption per inference on our robot

Figure 3: The performance of PP with various layer partitioning strategies on VGG19 [55]. The X-axis 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. It is worth noting that "layer 0" also represents the all offload method, as it places all layers of the model onto the GPU server, and different network bandwidths will result in varying transmission costs under the same layer partitioning strategy.

significantly increase energy consumption per inference by 28.5× to 62.7×. Although there have been efforts to reduce the communication overhead of TP in data centers, such as distributing each layer along both the spatial and temporal dimensions [59], these methods remain ineffective in robotic IoT. This is because they still require all-reduce communication to combine computation results from different devices, while Intra-DP takes a further step to eliminates the all-reduce communication for local operators.

Layer partitioning. Existing distributed inference approaches [8, 32] in robotic IoT primarily adopt the PP paradigm. Since the pipeline execution enhances inference throughput rather than reducing the completion time of a single inference [9], existing methods [5,8,20,23,32,43,67,74] on robotic IoT focus on optimizing the layer partitioning aspect of PP to achieve fast and energy-efficient inference. 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 [20], layer partitioning strategies constitute various trade-offs between computation and transmission and ultimately achieve much better performance than the all offload method (Fig. 3). These approaches can be categorized into two main groups based on their optimization goals: accelerating inference for diverse DNN structures [20,26,32,43,67] and optimizing robot energy consumption under deadline constraints [8, 33, 63]. The widespread use of layer partitioning to improve inference performance in robotic applications has attracted many researchers to study this field [5,20,23,43,67,74], as numerous factors can lead to differences in the choice of optimal strategy: model structure, hardware conditions (e.g., computing capabilities of GPU servers and robots), varying network bandwidth, and application-specific inference speed and energy consumption requirements. However, all existing layer partitioning methods suffer from the transmission bottleneck caused by sequential execution of DNN operations, which can be eliminated by Intra-DP.

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

Model Compression. Quantization and model distillation are the two most commonly used methods of model compression on the robots. Quantization [11, 16, 17] 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 [18,35,61], 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 are orthogonal to distributed inference, because they achieve faster inference speed by modifying the model structure and sacrificing the accuracy of the result, while distributed inference realizes fast inference without loss of accuracy by intelligently scheduling computation burden across multiple devices.

Inference Task scheduling. It schedules the execution of multiple DNN inference tasks to enhance overall system efficiency. This approach uses various decision algorithms to strategically schedule the execution location and timing of multiple inference tasks, such as batching tasks together [73], prioritizing based on urgency [34, 39], and employing deep reinforcement learning controls [4, 13, 14]. Unlike distributed inference methods that optimize each individual inference task, these methods focus on minimizing overall inference latency and energy consumption, while adopting existing distributed inference techniques for each task execution. Intra-DP provides these methods with a new parallel computing technique (LOP), which demonstrates much higher performance in robotic IoT, and these inference task scheduling methods can be seamlessly integrated into Intra-DP by utilizing their decision algorithms to determine the execution location and start time of each inference task during the execution process of Intra-DP.

3 Overview

3.1 Workflow of Intra-DP

Fig. 1 illustrates the workflow of Intra-DP and compares it with TP and PP in robotic IoT with limited bandwidth, highlighting the transmission overhead faced by existing methods and how Intra-DP addresses them through its LOP.

To alleviate the transmission overhead in distributed inference, Intra-DP overlaps the computation and transmission of local operations (as illustrated in Fig. 1 of LOP, Intra-DP transmits the input of the local operation '$LO1_1$' to the GPU server while simultaneously computing '$LO1_2$', '$LO1_3$', and '$LO2_3$' on the robot). Compared to TP, Intra-DP eliminates the need for synchronization (shown by the red dotted lines in Fig. 1) during all-reduce communication for local operators, maintaining all-reduce communication only for global operators. This workflow in LOP allows the inputs of some local operations can be obtained directly on the current device without the need for transmission (as exemplified in Fig. 1, '$LO2_1$' directly uses the output of '$LO1_1$' on the GPU server as its input), effectively eliminates the need for all-reduce communication for local operators. Although LOP generates more communication compared to PP due to its fine-grained transmission of local operations, Intra-DP's overlapping significantly reduces transmission completion time by initiating data transfer much earlier than PP, which could only begin transmission after the robot computation phase is finished due to its sequential execution (see more details in Sec. 6.2). As a result, Intra-DP dramatically reduces the idle time on robots compared with existing distributed inference methods, achieving much faster inference.

Moreover, the idle time on the robot leads to significant energy waste. Our analysis of the robot's energy consumption in various states, presented in Tab. 3, reveals that during idle time, components such as CPU, GPU, and memory consume non-negligible power even when not computing (e.g., 95% energy consumption in our experiments) due to static power consumption caused by transistors' leakage current [27]. This energy consumption cannot be avoided or reduced by entering a low-power sleep mode, as the robot must promptly resume work upon receiving inference results. Fortunately, we discovered that the wireless network card consumes only 0.21 Watt for transmission during idle time (4.25 Watt in total), while the robot consumes 13.35 Watt during computing. Although LOP's approach of overlapping the computation and transmission phases among different local operations slightly increases ethe energy consumption of wireless network cards during computing phase (only 1.5%), it significantly reduces the robot's idle time (14.9% to 41.1% in our experiments), thereby reducing the overall energy consumption.

3.2 Architecture of Intra-DP

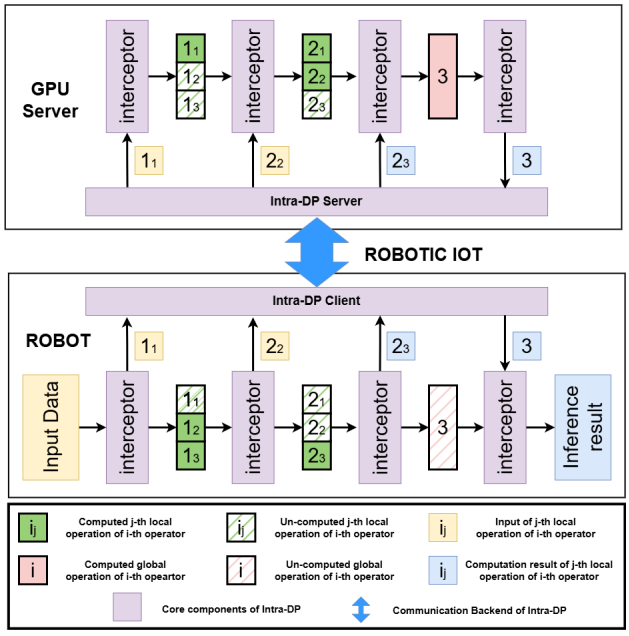


Figure 4: Architecture of Intra-DP. Intra-DP adopts the same scheduling scheme as in Fig. 1.

Fig. 4 illustrates the architecture of Intra-DP, which incorporates interceptors for each operator to enable flexible splitting

and combining of the input and output tensors. There are three stages: profiling, scheduling optimization, and runtime.

Profiling. It is an offline step that profiles essential runtime traces for optimization step and only needs to be done once. Those traces include: 1) the type and input/output sizes of operators; 2) the execution time of operations on the robot and GPU server; 3) the data dependencies among operations (i.e., the output of one operation serves as the input for another).

Scheduling Optimization. Based on the profiling traces, Intra-DP generates deployment strategies via its LOSS to determine the partitioning of local operations between robots and GPU servers, while considering the transmission cost of LOP. To account for the frequent bandwidth fluctuations in robotic IoT, Intra-DP generates various deployment strategies for different bandwidth conditions in advance. Note that the model inference time, typically in the range of tens to hundreds of milliseconds, is finer than the granularity of bandwidth fluctuation in wireless networks. Consequently, we assumes that the network bandwidth remains stable during each inference task, while acknowledging that it may vary across different inference tasks.

Runtime. During the runtime stage, Intra-DP measures the network bandwidth using mature tools [71] in wireless transmission and adopts the corresponding deployment strategy based on the current bandwidth. To enable flexible switching among various deployment strategies in response to rapid wireless network fluctuations at no cost, Intra-DP maintains a copy of the model on the GPU server during the profiling stage (Fig. 4). Compared to the original model inference process, Intra-DP only increases the time cost of interceptors for splitting input tensors and combining output tensors. The input tensor splitting time is negligible due to backend data transmission processes of the Intra-DP client and server, while other local operations continue calculations on the robot and GPU server. The output tensor combining time is mainly bound by the completion of computation and transmission on the other side, possibly causing idle waiting. Intra-DP formulates this waiting time into a nonlinear optimization problem in its LOSS, minimizing waiting time. In this way, Intra-DP achieves negligible extra system cost.

4 Detailed Design

4.1 Local Operation Parallelism

LOP guarantees the correctness of inference results by ensuring that all operations still receive the correct input. First, we analyze the minimum input unit for each operator based on its calculation characteristics and divide them into global operators and local operators according to whether the minimum unit input is the entire input tensor, while also determining their input/output sizes. Here, we summarize three classes of local operators common in models used on mobile devices:

- Element-wise local operator. The minimum input unit for this class of operators is the element of the input tensor. These operators are widely used in activation functions such as ReLU [10], Sigmoid [70], and SiLU [24]. However, it is important to note that some activation functions, like softmax [38], require the entire input tensor and are not local operators, but global operators.
- Block-wise local operator. The minimum input unit for this class of operators is the block at the corresponding position in the input tensor. These operators are widely used in layers associated with convolution operations, such as convolution [43] and maxpool [57]. The size of the input blocks is determined by the parameters set by the corresponding operator [51], including the size of the convolution kernel, padding, and dilation.
- Row-wise local operator. The minimum input unit for this class of operators is the rows of the input tensor. These operators are widely used in layers associated with matrix operations, such as addition [72] and multiplication [15]. Row-wised local operations split the input matrix by rows and share the same layer parameter matrix on different devices. According to matrix calculation principles as following, 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 operator in the same way without all-reduce communication.

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

It is important to note that the way row-wised local operations split matrices differs from TP, which splits the layer parameter matrix and transfers a copy of the input matrix to different devices. Moreover, LOP treats operators with layer parameter matrices containing only one row as global operators.

Model	Local operator	Global operator
DenseNet [22]	428	3
ResNet [58]	341	3
ConvNeXt [62]	340	6
RegNet [66]	231	3

Table 2: The number of local and global operators in several models commonly used on mobile devices.

Local operators are widely used in robotic applications, especially in convolution layers for computer vision tasks [42] and point cloud tasks [60], providing a large optimization space for the parallel execution of local operations. Based on the above definition, we calculate their proportion in several models commonly used on mobile devices (Tab. 2).

In addition to the local/global type, input/output of each operator, and execution time on the robot and GPU server, the profiling stage also requires the data dependencies among operations. LOP establishes these data dependencies by determining whether the output of one operation serves as the

input for another. Even if inputs and outputs are only partially the same, the dependency is recognized. This allows a global operation to be treated as a special case of a local operation of which the local operator only calculates once, facilitating easier discussion and modeling in Sec. 4.2. Notably, when an output is used by multiple local operations on robots and GPU servers simultaneously, particularly in block-wise local operations, LOP allows the re-calculation of these operations, letting them compute simultaneously on both the robots and GPU servers (see more details in Sec. 4.2). This approach reduces the need for expensive data transmission in robotic IoT by incorporating a small amount of redundant computation.

4.2 Local Operation Scheduling Strategy

LOSS formulates the partitioning of local operations as a optimization problem aimed at minimizing distributed inference time. To achieve this, we first model the inference time of Intra-DP based on its workflow (Fig. 1), as follows:

First, we treat each calculation of an operator based on its minimum input unit as an operation, and define OP_i as the set of operations for the i_{th} operator, encompassing both local and global operators. When the i_{th} operator is a global operator, $|OP_i| = 1$ as it only has one operation. We define $X_i \subseteq OP_i$ as the set of operations located on robots, and $Y_i \subseteq OP_i$ as the set executed on the GPU server, where $X_i \cup Y_i = OP_i$. $X_i \cap Y_i \neq \emptyset$ when some local operations of the i_{th} operator are re-calculated, especially for block-wise local operators; otherwise, $X_i \cap Y_i = \emptyset$.

Next, we denote the completion time of the i_{th} operator on the robot as T^i_{robot} and that on the GPU server as T^i_{server} according to Fig. 1. We define $compute(X)$ as the estimated computation time of X and $transmit(X)$ as the estimated transmission time of X under the given bandwidth, leading to the following formula:

$$T^i_{robot} = \begin{cases} compute(X_0) & i = 0 \\ T^{i-1}_{robot} + compute(X_i) & i > 0, M_i = \emptyset \\ MAX\left(T^{i-1}_{robot}, T^j_{server} + \right. & \\ \left. transmit(M_i)\right) + compute(X_i) & i > 0, M_i \neq \emptyset \end{cases}$$

$$T^i_{server} = \begin{cases} transmit(Y_0) + compute(Y_0) & i = 0 \\ T^{i-1}_{server} + compute(Y_i) & i > 0, N_i = \emptyset \\ MAX\left(T^{i-1}_{server}, T^j_{robot} + \right. & \\ \left. transmit(N_i)\right) + compute(Y_i) & i > 0, N_i \neq \emptyset \end{cases}$$

Here, $M_i = parent(X_i) - X_{i-1}$ and $N_i = parent(Y_i) - Y_{i-1}$, where $parent(X_i)$ is the set of operators whose output serves as the input of any operator in X_i, and the j_{th} operator is the last operator of $parent(X_i)$. The consideration that an operation may have several parents allows Intra-DP to support DNN models with complex structures as directed acyclic graphs. The *MAX* function is used to minimize the waiting time when combining the input tensor, and $transmit(X)$ includes not only its own transmission time but also the wait time for the previous transmission to complete.

Next, we present the corresponding objective function and constraints of Intra-DP on a DNN model with N layers:

$$\min \quad T^N_{robot} \tag{1}$$
$$\text{s.t.} \quad M_i = \emptyset, \forall i \in \Pi \tag{2}$$
$$N_i = \emptyset, \forall i \in \Pi \tag{3}$$

Here, the operators in Π are those whose output data amounts are larger than the raw input data. Constraints are inspired by the key observation used in existing layer partitioning methods to limit the transmission overhead, which states that the output data amounts in some intermediate layers of a DNN model are significantly smaller than that of its raw input data [20].

To tackle the nonlinear and non-convex nature of the objective function, LOSS employs the differential evolution algorithm [52] to solve the optimization problem, and schedules the computation and transmission of each operation based on the obtained solution. We adopt the differential evolution algorithm for its superior global search capabilities and its support for parallel computation, enabling high-quality, faster solution processes. The performance of Intra-DP is highly dependent on the quality of the solution obtained by LOSS, with better solutions (closer to the global optimal solution) leading to improved performance. However, finding the global optimal solution for non-convex optimization problems in finite time remains an open challenge, and developing an enhanced algorithm that provides delivers, high-quality solutions for the non-convex optimization problem in LOSS is left for future work. It is important to note that when applying Intra-DP to a special model without any local operator, LOSS will degrade to the existing layer partitioning method.

4.3 Algorithms of Intra-DP

Here, we present the algorithm of Intra-DP for both the client side on the robot and the server side on the GPU server, as illustrated in Fig. 4. The client and server components are presented in Alg. 1 and Alg. 2, respectively. Initially, both sides undergo a profiling phase, providing runtime traces ($info\_robot$ and $info\_server$) to LOSS. Intra-DP then generates schedule plans through LOSS (Alg. 3) for various bandwidth conditions during the scheduling optimization stage. Lastly, during the runtime stage, Intra-DP selects the appropriate schedule plan based on the measured bandwidth, leveraging the model copy on the GPU server (Fig. 4, line 1 in Alg. 2) for flexible schedule plan switching without cost.

The profiling and scheduling stages of Intra-DP, although time-consuming, taking tens of minutes for each model on our testbed, are offline steps performed in advance and do not impact the inference time during runtime. These pre-calculated schedule plans for various bandwidth cases only need to be re-executed when hardware changes occur in the testbed, specif-

Algorithm 1: Intra-DP client

Input: Data input for inference *input*; DNN model *model*
Output: The inference result *ret*
Data: Z_i: input of i_{th} layer ; X_i^b, M_i^b,N_i^b: schedule plan of i_{th} layer under the b bandwidth

// profiling stage on robot.
$info\_robot = ProfileModel(model)$
$SendToServer(model, info\_robot)$
$X, M, N = ReceiveFromServer()$
// runtime stage on robot
$b = TestBandwidth()$
$Z_0 = input$
foreach i_{th} *layer in model* **do**
  if $M_i^b \neq \emptyset$ **then**
    $Z_i = combine(Z_i, ReceiveFromServer())$
  end
  if $N_i^b \neq \emptyset$ **then**
    $SendToServer(Z_i, N_i^b)$
  end
  if $X_i^b \neq \emptyset$ *and* $Z_i \neq \emptyset$ **then**
    $Z_{i+1} = compute(Z_i, X_i^b)$
  end
  else
    $Z_{i+1} = \emptyset$
  end
end
$ret = Z_{N+1}$
return *ret*

Algorithm 2: Intra-DP server

Data: Z_i: input of i_{th} layer; Y_i^b, M_i^b,N_i^b: schedule plan of i_{th} layer under the b bandwidth.

// profiling stage on server
$model, info\_robot = ReceiveFromClient()$
$info\_server = ProfileModel(model)$
$X, Y, M, N = LOSS(info\_robot, info\_server)$
$SendToClient(X, M, N)$
// runtime stage on server
$b = TestBandwidth()$
$Z_0 = \emptyset$
foreach i_{th} *layer in model* **do**
  if $M_i^b \neq \emptyset$ **then**
    $SendToClient(Z_i, M_i^b)$
  end
  if $N_i^b \neq \emptyset$ **then**
    $Z_i = combine(Z_i, ReceiveFromClient())$
  end
  if $Y_i^b \neq \emptyset$ *and* $Z_i \neq \emptyset$ **then**
    $Z_{i+1} = compute(Z_i, Y_i^b)$
  end
  else
    $Z_{i+1} = \emptyset$
  end
end

ically when the computing accelerator (GPU) of the robot or the GPU server is updated. Intra-DP does not employ reinforcement learning for scheduling because, when the testbed hardware remains unchanged, the only variable impacting the partitioning of local operations is the bandwidth, resulting in a limited solution space. Consequently, sub-optimal solutions can be obtained by mathematical solvers within a limited time frame, avoiding the additional overhead associated with the exploration phase of reinforcement learning [28].

5 Implementation

```
# 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 [55] model, where "192.168.50.1" is the IP address of the GPU server.

We implement Intra-DP on Python and PyTorch. Intra-DP is easy to use and requires only three lines of code to apply to existing robotic applications, as shown in Fig. 5. This is achieved by hooking around the forward method of the model, and in the first forward call we profile the model using the default PyTorch profiler and schedule; then we intercept and parallelize all the following forward calls as scheduled.

6 Evaluation

Testbed. The evaluation was conducted on a customized four-wheeled robot (Fig. 6a), and a customized air-ground robot(Fig. 6b). They are each equipped with a Jetson Xavier NX [47] 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 is a PC equipped with an Intel(R) i7-7700K CPU @ 4.20GHz and an NVIDIA GeForce GTX 1080 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

Algorithm 3: LOSS

Data: $info\_robot$,$info\_server$: runtime traces on robot and GPU server ; BW: theoretical maximum bandwidth in robotic IoT; X_i^b, Y_i^b, M_i^b,N_i^b: schedule plan of i_{th} layer under the b bandwidth; T_{robot}^N: follow the modeling in the Sec. 4.2.

// scheduling optimization stage

foreach *b bandwidth less than BW* **do**

  // Initialize

  foreach i_{th} *layer in model* **do**

    $\mathcal{X}_i^{init} = OP_i$

    $\mathcal{Y}_i^{init} = \emptyset$

  end

  // add constraints

  foreach i_{th} *layer in* Π **do**

    $constraints.add(\mathcal{M}_i = \emptyset)$

    $constraints.add(\mathcal{N}_i = \emptyset)$

  end

  // objective function

  $obj\_func = T_{robot}^N(info\_robot, info\_server, (\mathcal{X}_0,..,\mathcal{X}_N,\mathcal{Y}_0,..,\mathcal{Y}_N), b)$

  // solve optimization problems

  $X_0^b,..,X_N^b,Y_0^b,..,Y_N^b = differential\_evolution\_solver(obj\_func, constraints, (\mathcal{X}_0^{init},..,\mathcal{X}_N^{init}, \mathcal{Y}_0^{init},..,\mathcal{Y}_N^{init}))$

  foreach i_{th} *layer in model* **do**

    $M_i^b = parent\left(X_i^b\right) - X_{i-1}^b$

    $N_i^b = parent\left(Y_i^b\right) - Y_{i-1}^b$

  end

end

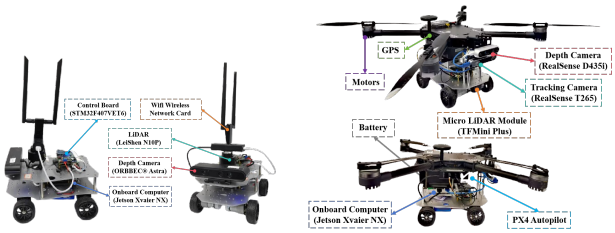
return $\bigcup_{b=1}^{BW}\bigcup_{i=0}^{N} X_i^b$, $\bigcup_{b=1}^{BW}\bigcup_{i=0}^{N} Y_i^b$, $\bigcup_{b=1}^{BW}\bigcup_{i=0}^{N} M_i^b$, $\bigcup_{b=1}^{BW}\bigcup_{i=0}^{N} N_i^b$

consumption), communication (communication with the GPU server, including wireless network card energy consumption), and standby (robot has no tasks to execute). The Jetson Xavier NX is connected to a 21.6Wh battery that supports up to 1.6 hours of model inference.

	inference	communication	standby
Energy (Watt)	13.35	4.25	4.04

Table 3: Energy consumption per unit time (Watt) of our robot in different states.

Workload. We evaluated two typical real-world robotic applications on our testbed: Kapao [42], a real-time people-tracking application on our four-wheeled robot with a CNN-based human keypoint detection model (Fig 7), and AGRNav [60], an autonomous navigation application on our air-ground robot with a CNN-based 3D semantic scene completion model (Fig 8). We also evaluated several models common to mobile devices with their implementation from Torchvision [40] on a larger scale to further corroborate our observations and findings: DenseNet [22], RegNet [66], VGGNet [55], ConvNeXt [62].



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

Figure 6: The detailed composition of the robot platforms

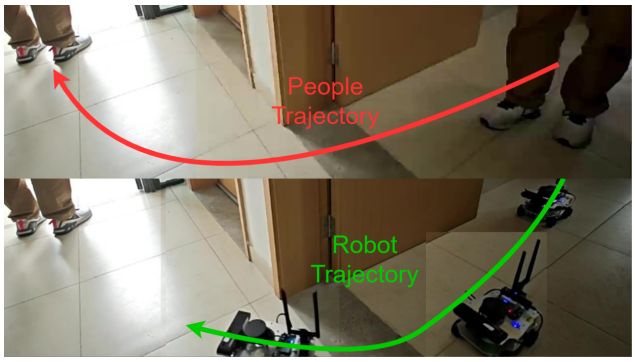


Figure 7: A real-time people-tracking robotic application on our robot based on a human pose estimation model, Kapao [42].

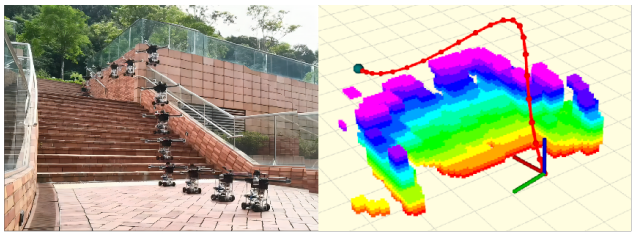


Figure 8: By predicting occlusions in advance, AGRNav [60] gains an accurate perception of the environment and avoids collisions, resulting in efficient and energy-saving paths.

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.

Baselines. We compared Intra-DP with four baselines: local computation (entire model on the robot), all offload (entire model on GPU servers), and two SOTA PP methods, DSCCS [32] (accelerating inference) and SPSO-GA [8] (optimizing energy consumption under deadline constraints). No model compression methods are involved and all these offloading systems are transmitting raw model activations without compression to avoid affecting model accuracy. Although

Model(number of parameters)	System	Inference time(s)		Energy consumption per unit time(W)		Energy consumption per inference(J)	
		indoors	outdoors	indoors	outdoors	indoors	outdoors
Kapao (77M)	Local	0.92(±0.06)	0.92(±0.06)	10.61(±0.48)	10.61(±0.48)	9.77(±0.61)	9.77(±0.61)
	ALL	0.60(±0.35)	0.76(±1.07)	4.81(±0.11)	4.74(±0.17)	2.87(±0.07)	3.6(±0.13)
	DSCCS	0.51(±0.26)	0.64(±0.69)	6.58(±2.34)	7.35(±2.49)	3.33(±1.18)	4.67(±1.58)
	SPSO-GA	0.47(±0.24)	0.59(±0.74)	5.81(±1.8)	6.32(±2.1)	2.7(±0.84)	3.75(±1.25)
	Intra-DP	0.45(±0.35)	0.55(±0.57)	6.02(±2.28)	6.85(±2.57)	2.68(±1.01)	3.8(±1.43)
AGRNav (0.84M)	Local	0.53(±0.02)	0.53(±0.02)	8.11(±0.26)	8.11(±0.26)	4.33(±0.19)	4.33(±0.19)
	ALL	0.99(±0.60)	1.16(±1.39)	4.56(±0.09)	4.51(±0.13)	4.49(±0.09)	5.24(±0.15)
	DSCCS	0.52(±0.13)	0.52(±0.66)	7.36(±1.35)	6.7(±1.71)	3.81(±0.7)	3.5(±0.89)
	SPSO-GA	0.51(±0.14)	0.52(±0.56)	7.1(±1.46)	6.83(±1.74)	3.6(±0.74)	3.57(±0.91)
	Intra-DP	0.40(±0.14)	0.46(±0.46)	6.63(±1.8)	7.29(±1.37)	2.63(±0.71)	3.37(±0.63)

Table 4: Average inference time (Second), energy consumption per unit time (Watt) and energy consumption per inference (Joule), along with the standard deviation ($\pm n$), for Kapao and AGRNav in different environments using various systems. "Local" represents "local computation", and "All" represents "all offload".

local computation and all offload can be considered special cases of layer partitioning, we still evaluated them separately to verify the contribution of the other baselines in reducing inference latency and energy consumption. We set SPSO-GA's deadline constraint 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 individual inference task.

The evaluation questions are as follows:

- RQ1: How much does Intra-DP benefit real-world robotic applications compared to baseline systems in terms of inference time and energy consumption?
- RQ2: How does Intra-DP achieve faster inference time compared to other baselines?
- RQ3: How does Intra-DP perform on models common to mobile devices on a larger scale?
- RQ4: What are the limitations and potentials of Intra-DP?

6.1 End-to-end Performance

Inference time. The columns about inference time on Tab.4 demonstrates that Intra-DP achieves the fastest inference time among all baselines on both Kapao and AGRNav in both indoor and outdoor scenarios. Compared to other baselines, Intra-DP reduces Kapao's inference time by 5% to 51% indoors and 7% to 40% outdoors, while reducing AGRNav's inference time by 22% to 60% indoors and 12% to 61% outdoors. DSCCS failed to achieve the fastest inference time among PP methods due to the frequent and sharp bandwidth fluctuations in robotic IoT. Furthermore, the large standard deviation and longer inference times outdoors for offloading methods (all offload, two PP methods, and Intra-DP) can be attributed to the more frequent and severe bandwidth fluctuations outdoors compared to indoors, as illustrated in Fig. 2. A detailed breakdown of how Intra-DP achieves such fast inference time will be provided in Sec. 6.2.

Energy Consumption. From the columns about energy consumption per unit time on Tab. 4, we can learn that for Kapao and AGRNav, local computation has the highest energy consumption per unit time due to the heavy computational burden on the robot, while all offload has the lowest energy consumption per unit time as it places the entire computational burden onto the GPU server. DSCCS, SPSO-GA, and Intra-DP have moderate energy consumption per unit time since they place partial computational burden on the robots. Among these three methods, DSCCS achieves the lowest energy consumption per unit time, as it is specifically designed to optimize energy consumption. Intra-DP incurs slightly higher energy consumption per unit time due to possibly extra local computation on the re-calculation of some local operations.

The last columns about energy consumption per inference of Tab. 4 demonstrates that despite the high energy consumption per unit time, Intra-DP achieves the lowest energy consumption per inference among all baselines on both Kapao and AGRNav in both indoor and outdoor scenarios. Compared to other baselines, Intra-DP reduces Kapao's energy consumption per inference by 4% to 72% indoors and up to 62% outdoors, while reducing AGRNav's energy consumption per inference by 27% to 42% indoors and 4% to 36% outdoors. While Intra-DP incurs slightly higher energy consumption per unit time, Intra-DP achieved the fastest inference time by overlapping the computation and transmission of different local operations, leading the least overall energy consumption per inference.

6.2 Breakdown

We present a detailed breakdown of each phase of the inference process in Tab. 5 to explain how Intra-DP achieved fastest inference time compared to other baselines. According to their workflow, as shown in Fig. 1, there are three sequen-

Model	System	Metrics	Robot computation		Transmission		Server computation	
			indoors	outdoors	indoors	outdoors	indoors	outdoors
Kapao (77M)	Local	Time(s)	0.92(±0.06)	0.92(±0.06)	0.00(±0.00)	0.00(±0.00)	0.00(±0.00)	0.00(±0.00)
		Percentage(%)	100.0(±0.0)	100.0(±0.0)	0.0(±0.0)	0.0(±0.0)	0.0(±0.0)	0.0(±0.0)
	ALL	Time(s)	0.06(±0.00)	0.05(±0.01)	0.26(±0.22)	0.35(±0.63)	0.31(±0.22)	0.39(±0.63)
		Percentage(%)	9.3(±0.8)	7.1(±1.1)	44.0(±36.1)	46.0(±82.9)	51.8(±36.1)	51.9(±82.9)
	DSCCS	Time(s)	0.17(±0.25)	0.19(±0.29)	0.08(±0.10)	0.13(±0.24)	0.28(±0.17)	0.35(±0.60)
		Percentage(%)	34.0(±49.2)	29.7(±46.0)	16.0(±20.4)	20.0(±37.1)	55.1(±34.0)	55.3(±94.0)
	SPSO-GA	Time(s)	0.11(±0.17)	0.13(±0.21)	0.08(±0.09)	0.11(±0.18)	0.31(±0.19)	0.39(±0.64)
		Percentage(%)	22.7(±35.5)	22.1(±36.0)	16.4(±19.5)	18.0(±30.1)	65.8(±40.0)	64.9(±108.3)
	Intra-DP	Time(s)	0.22(±0.17)	0.26(±0.27)	0.13(±0.17)	0.20(±0.38)	0.23(±0.23)	0.27(±0.40)
		Percentage(%)	48.4(±37.5)	47.2(±48.7)	29.6(±37.1)	36.6(±68.2)	51.1(±51.8)	49.3(±71.7)
AGRNav (0.84M)	Local	Time(s)	0.53(±0.02)	0.53(±0.02)	0.00(±0.00)	0.00(±0.00)	0.00(±0.00)	0.00(±0.00)
		Percentage(%)	100.0(±0.0)	100.0(±0.0)	0.0(±0.0)	0.0(±0.0)	0.0(±0.0)	0.0(±0.0)
	ALL	Time(s)	0.01(±0.00)	0.01(±0.00)	0.50(±0.33)	0.59(±0.74)	0.53(±0.33)	0.62(±0.74)
		Percentage(%)	0.8(±0.3)	0.6(±0.2)	50.9(±33.1)	51.2(±64.2)	53.3(±33.1)	53.2(±64.2)
	DSCCS	Time(s)	0.02(±0.08)	0.04(±0.21)	0.23(±0.15)	0.20(±0.35)	0.29(±0.11)	0.31(±0.49)
		Percentage(%)	4.7(±14.9)	7.4(±39.6)	44.2(±29.0)	37.7(±66.9)	56.1(±21.6)	59.9(±93.8)
	SPSO-GA	Time(s)	0.05(±0.10)	0.04(±0.20)	0.20(±0.17)	0.20(±0.27)	0.29(±0.13)	0.31(±0.43)
		Percentage(%)	9.0(±20.0)	7.3(±38.0)	39.3(±34.2)	38.0(±52.3)	56.7(±26.5)	59.8(±81.5)
	Intra-DP	Time(s)	0.11(±0.04)	0.18(±0.18)	0.32(±5.39)	0.27(±0.46)	0.22(±2.69)	0.23(±0.31)
		Percentage(%)	27.0(±9.3)	39.2(±38.8)	80.0(±1360.9)	58.1(±100.2)	56.5(±680.3)	50.0(±66.2)

Table 5: The time spent in each phase of the inference process and their percentage of the total inference time, along with the standard deviation ($\pm n$), for the Kapao and AGRNav in different environments using various systems.

tial phases: robot computation (computing early layers on robots), transmission (transmitting intermediate results), and server computation (completing inference on GPU servers). Local computation places the entire model on the robot and only has the robot computation time, while all offload places the entire model on the GPU server and has to transfer the raw input to the GPU server. 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 [20], and considering that DCCSS, SPSO-GA, and Intra-DP can respond to rapid wireless network fluctuations, these methods have lower transmission volumes of intermediate results compared to all offload, leading to shorter transmission time in Tab. 5.

Transmission time accounts for up to 44.2% of the total inference time in PP methods (SPSO-GA and DSCCS) in Tab. 5, highlighting the significant transmission bottlenecks faced by existing methods based on the PP paradigm, even with SOTA layer partitioning strategies. Although Intra-DP generates more communication compared to PP methods due to its finer-grain transmission of local operators (both Kapao and AGRNav in Tab. 5), its overlapping significantly reduces transmission completion time by initiating data transfer much earlier than PP, as shown in Fig. 1. As a result, while PP methods can only execute each phase sequentially (the percentages for all phases add up to about 100% for SPSO-GA and DSCCS), Intra-DP overlaps the transmission phase with the robot computation phase and the server computation phase (the percentages for all stages add up to more than 100% for Intra-DP), achieving faster inference time compared to other baselines.

6.3 Validation on a larger range of models

Our comprehensive evaluation of Intra-DP and other baselines, conducted across a diverse set of models commonly used in mobile devices with varying parameter counts (Table 6), confirmed Intra-DP's consistent advantages across various models. We observed that offloading methods achieved relatively larger performance gains compared to local computation for models with higher local computation time, while they even performed slower inference than local computation on DenseNet. This suggests that models with larger computational burden benefit more significantly from offloading, while those with fewer computational burden may suffer from the additional communication cost, particularly in robotic IoT systems with limited bandwidth. Furthermore, although Intra-DP consistently outperformed other baselines, its performance gains compared to other baselines were relatively smaller for models with fewer parameters. This observation can be attributed to the fact that Intra-DP's performance gain is primarily achieved through the parallel execution of local operations, making it dependent on the model's size and structure. When a model has fewer parameters, the number of local operations available for parallel execution is reduced, limiting the optimization potential for Intra-DP to enhance performance.

Model(number of parameters)	System	Inference time(ms)		Energy consumption per unit time(W)		Energy consumption per inference(J)	
		indoors	outdoors	indoors	outdoors	indoors	outdoors
DenseNet(7.98M)	Local	56.29(±4.34)	56.29(±4.34)	8.2(±0.27)	8.2(±0.27)	0.46(±0.04)	0.46(±0.04)
	ALL	115.18(±48.54)	128.50(±45.00)	5.43(±0.24)	5.46(±0.28)	0.63(±0.03)	0.7(±0.04)
	DSCCS	95.25(±39.06)	100.73(±33.78)	5.03(±0.17)	4.74(±0.2)	0.48(±0.02)	0.48(±0.02)
	SPSO-GA	98.69(±34.63)	106.73(±71.82)	6.91(±0.45)	6.86(±0.46)	0.68(±0.04)	0.73(±0.05)
	Intra-DP	85.86(±27.63)	92.35(±22.95)	4.77(±0.7)	5.14(±0.22)	0.41(±0.06)	0.48(±0.02)
RegNet(54M)	Local	151.05(±11.53)	151.05(±11.53)	9.0(±0.3)	9.0(±0.3)	1.36(±0.1)	1.36(±0.1)
	ALL	103.94(±46.84)	114.85(±45.10)	5.46(±0.22)	5.44(±0.22)	0.57(±0.02)	0.62(±0.02)
	DSCCS	83.88(±36.04)	89.90(±31.34)	5.02(±0.19)	4.8(±0.18)	0.42(±0.02)	0.43(±0.02)
	SPSO-GA	82.74(±36.90)	90.10(±30.92)	5.86(±1.8)	5.36(±1.34)	0.49(±0.15)	0.48(±0.12)
	Intra-DP	66.98(±27.66)	72.12(±24.27)	4.67(±1.28)	4.69(±1.34)	0.31(±0.09)	0.34(±0.1)
VGG19(143M)	Local	96.34(±6.04)	96.34(±6.04)	9.78(±0.34)	9.78(±0.34)	0.94(±0.06)	0.94(±0.06)
	ALL	87.90(±44.72)	96.35(±37.58)	5.8(±0.27)	5.87(±0.23)	0.51(±0.02)	0.57(±0.02)
	DSCCS	71.97(±33.75)	76.92(±29.90)	5.32(±0.23)	4.83(±0.2)	0.38(±0.02)	0.37(±0.02)
	SPSO-GA	69.08(±59.95)	73.77(±24.43)	6.6(±2.14)	6.94(±2.34)	0.46(±0.15)	0.51(±0.17)
	Intra-DP	54.62(±14.71)	58.82(±11.14)	5.8(±1.54)	6.51(±1.35)	0.32(±0.08)	0.38(±0.08)
ConvNeXt(197M)	Local	287.70(±29.58)	287.70(±29.58)	10.72(±0.38)	10.72(±0.38)	3.08(±0.32)	3.08(±0.32)
	ALL	117.12(±46.59)	127.35(±43.77)	5.67(±0.34)	5.65(±0.25)	0.66(±0.04)	0.72(±0.03)
	DSCCS	94.06(±36.36)	99.19(±31.79)	5.04(±0.16)	4.99(±0.21)	0.47(±0.01)	0.5(±0.02)
	SPSO-GA	94.70(±48.38)	102.09(±43.78)	5.06(±0.31)	5.02(±0.37)	0.48(±0.03)	0.51(±0.04)
	Intra-DP	75.43(±40.09)	80.46(±35.78)	4.06(±0.21)	4.04(±0.22)	0.31(±0.02)	0.32(±0.02)

]

Table 6: Average inference time (Millisecond), energy consumption per unit time (Watt) and energy consumption per inference (Joule), along with the standard deviation ($\pm n$), for torchvision models in different environments using various systems.

6.4 Lessons learned

Limited bandwidth. Due to hardware limitations, we evaluated Intra-DP's performance only on the most common and easily accessible Wi-Fi networks on robots under different environments, rather than a wider variety of wireless networks such as 6G [7] or WiMAX [3]. Although these wireless networks differ in throughput and communication range due to their various transmission protocols, they share common issues due to the decay of wireless signals caused by device movement [41, 49], occlusion by physical barriers [12, 54], and wireless channel preemption by other devices [2, 53]. The resulting bandwidth fluctuations in practice (Fig. 2) lead to limited bandwidth in these wireless networks, where Intra-DP proves beneficial. As models grow larger and GPU devices become more powerful, the desired bandwidth for distributed inference will continue to increase [32]. And when GPU servers are deployed in commercial clouds, robotic IoT rely not only on wireless networks but also on Internet access to the cloud, where available bandwidth are significantly constrained due to network congestion and routing issues [45], making Intra-DP still beneficial.

Local Operator. During the implementation and evaluation of Intra-DP, we discovered that the presence of more local operator allows for increased parallel execution during model inference, thereby enhancing the performance improvement of Intra-DP. Future work should focus on supporting additional types of local operators and exploring the possibility of transforming global operators into local ones through lightweight synchronization techniques, based on their computational characteristics (e.g., synchronize the sum results in softmax [38] instead of directly transferring the full input tensor).

Future Work. It is of interest to explore enhancing Intra-DP by developing a distributed inference system for multiple robots with several models, aiming to minimize overall inference time and energy consumption. Intra-DP's finer-grained scheduling granularity enables more efficient utilization of computing resources, which is particularly advantageous in resource-constrained robotic scenarios.

7 Conclusion

In this paper, we present Intra-DP, a high-performance distributed inference system optimized for robotic IoT networks. By breaking down the granularity of distributed inference based on local operators via LOP and applying adaptive scheduling to the computation and transmission of each local operation via LOSS, Intra-DP dramatically reduces the transmission overhead in robotic IoT by overlapping the computation and transmission of different local operations, achieving fast and energy-efficient distributed inference. We envision that the fast and energy-efficient inference of Intra-DP will foster the real-world deployment of diverse AI robotic tasks in the field.

References

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

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

[3] Syed A. Ahson and Mohammad Ilyas. *WiMAX: Standards and Security*. The WiMAX handbook. CRC Press.

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

[5] Amin Banitalebi-Dehkordi, Naveen Vedula, Jian Pei, Fei Xia, Lanjun Wang, and Yong Zhang. Auto-split: A general framework of collaborative edge-cloud ai. In *Proceedings of the 27th ACM SIGKDD Conference on Knowledge Discovery & Data Mining*, pages 2543–2553, 2021.

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

[7] Robin Chataut, Mary Nankya, and Robert Akl. 6g networks and the ai revolution—exploring technologies, applications, and emerging challenges. *Sensors*, 24(6):1888, 2024.

[8] Xing Chen, Jianshan Zhang, Bing Lin, Zheyi Chen, Katinka Wolter, and Geyong Min. Energy-efficient offloading for dnn-based smart iot systems in cloud-edge environments. *IEEE Transactions on Parallel and Distributed Systems*, 33(3):683–697, 2021.

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

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

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

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

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

[14] Zhou Fang, Tong Yu, Ole J Mengshoel, and Rajesh K Gupta. 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*, pages 2067–2070, 2017.

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

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

[17] Cheng Gong, Yao Chen, Ye Lu, Tao Li, Cong Hao, and Deming Chen. Vecq: Minimal loss dnn model compression with vectorized weight quantization. *IEEE Transactions on Computers*, 70(5):696–710, 2020.

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

[19] Vinayak Honkote, Dileep Kurian, Sriram Muthukumar, Dibyendu Ghosh, Satish Yada, Kartik Jain, Bradley Jackson, Ilya Klotchkov, Mallikarjuna Rao Nimmagadda, Shreela Dattawadkar, et al. 2.4 a distributed autonomous and collaborative multi-robot system featuring a low-power robot soc in 22nm cmos for integrated battery-powered minibots. In *2019 IEEE International Solid-State Circuits Conference-(ISSCC)*, pages 48–50. IEEE, 2019.

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

[21] Yang Hu, Connor Imes, Xuanang Zhao, Souvik Kundu, Peter A Beerel, Stephen P Crago, and John Paul Walters.

Pipeedge: Pipeline parallelism for large-scale model inference on heterogeneous edge devices. In *2022 25th Euromicro Conference on Digital System Design (DSD)*, pages 298–307. IEEE, 2022.

[22] Gao Huang, Zhuang Liu, Laurens van der Maaten, and Kilian Q. Weinberger. Densely connected convolutional networks, 2018.

[23] Jin Huang, Colin Samplawski, Deepak Ganesan, Benjamin Marlin, and Heesung Kwon. Clio: Enabling automatic compilation of deep learning pipelines across iot and cloud. In *Proceedings of the 26th Annual International Conference on Mobile Computing and Networking*, pages 1–12, 2020.

[24] Glenn Jocher, Alex Stoken, Jirka Borovec, Liu Changyu, Adam Hogan, Ayush Chaurasia, Laurentiu Diaconu, Francisco Ingham, Adrien Colmagro, Hu Ye, et al. ultralytics/yolov5: v4. 0-nn. silu () activations, weights & biases logging, pytorch hub integration. *Zenodo*, 2021.

[25] K J Joseph, Salman Khan, Fahad Shahbaz Khan, and Vineeth N Balasubramanian. Towards open world object detection. In *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition (CVPR)*, pages 5830–5840, June 2021.

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

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

[28] Pawel Ladosz, Lilian Weng, Minwoo Kim, and Hyondong Oh. Exploration in deep reinforcement learning: A survey. *Information Fusion*, 85:1–22, 2022.

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

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

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

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

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

[34] Li Lin, Xiaofei Liao, Hai Jin, and Peng Li. Computation offloading toward edge computing. *Proceedings of the IEEE*, 107(8):1584–1607, 2019.

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

[36] Ruofeng Liu and Nakjung Choi. 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):1–25, 2023.

[37] Shuai Liu, Xin Li, Huchuan Lu, and You He. Multi-object tracking meets moving uav. In *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition (CVPR)*, pages 8876–8885, June 2022.

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

[39] Pavel Mach and Zdenek Becvar. Mobile edge computing: A survey on architecture and computation offloading. *IEEE communications surveys & tutorials*, 19(3):1628–1656, 2017.

[40] Sébastien Marcel and Yann Rodriguez. Torchvision the machine-vision package of torch. In *Proceedings of the 18th ACM International Conference on Multimedia*, MM '10, page 1485–1488, New York, NY, USA, 2010. Association for Computing Machinery.

[41] Antoni Masiukiewicz. 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):79–84, 2019.

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

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

[44] Deepak Narayanan, Mohammad Shoeybi, Jared Casper, Patrick LeGresley, Mostofa Patwary, Vijay Korthikanti, Dmitri Vainbrand, Prethvi Kashinkunti, Julie Bernauer, Bryan Catanzaro, et al. Efficient large-scale language model training on gpu clusters using megatron-lm. In *Proceedings of the International Conference for High Performance Computing, Networking, Storage and Analysis*, pages 1–15, 2021.

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

[46] NVIDIA. Infiniband networking solutions. https://www.nvidia.com/en-us/networking/products/infiniband/, 2024.

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

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

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

[50] pytorch. pytroch. https://pytorch.org/, 2024.

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

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

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

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

[55] Karen Simonyan and Andrew Zisserman. Very deep convolutional networks for large-scale image recognition, 2015.

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

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

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

[59] Haoran Wang, Lei Wang, Haobo Xu, Ying Wang, Yuming Li, and Yinhe Han. Primepar: Efficient spatial-temporal tensor partitioning for large transformer model training. In *Proceedings of the 29th ACM International Conference on Architectural Support for Programming Languages and Operating Systems, Volume 3*, pages 801–817, 2024.

[60] Junming Wang, Zekai Sun, Xiuxian Guan, Tianxiang Shen, Zongyuan Zhang, Tianyang Duan, Dong Huang, Shixiong Zhao, and Heming Cui. Agrnav: Efficient and energy-saving autonomous navigation for air-ground robots in occlusion-prone environments. In *IEEE International Conference on Robotics and Automation (ICRA)*, 2024.

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

[62] Sanghyun Woo, Shoubhik Debnath, Ronghang Hu, Xinlei Chen, Zhuang Liu, In So Kweon, and Saining Xie. Convnext v2: Co-designing and scaling convnets with masked autoencoders. In *Proceedings of the IEEE/CVF*

Conference on Computer Vision and Pattern Recognition, pages 16133–16142, 2023.

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

[64] Zhaoyang Xia, Youquan Liu, Xin Li, Xinge Zhu, Yuexin Ma, Yikang Li, Yuenan Hou, and Yu Qiao. Scpnet: Semantic scene completion on point cloud. In *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition*, pages 17642–17651, 2023.

[65] Yecheng Xiang and Hyoseung Kim. Pipelined data-parallel cpu/gpu scheduling for multi-dnn real-time inference. In *2019 IEEE Real-Time Systems Symposium (RTSS)*, pages 392–405. IEEE, 2019.

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

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

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

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

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

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

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

[73] Daniel Zhang, Nathan Vance, Yang Zhang, Md Tahmid Rashid, and Dong Wang. Edgebatch: Towards ai-empowered optimal task batching in intelligent edge systems. In *2019 IEEE Real-Time Systems Symposium (RTSS)*, pages 366–379. IEEE, 2019.

[74] Letian Zhang, Lixing Chen, and Jie Xu. Autodidactic neurosurgeon: Collaborative deep inference for mobile edge intelligence via online learning. In *Proceedings of the Web Conference 2021*, pages 3111–3123, 2021.

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