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

Anonymous Author(s)
Submission Id: 273

Abstract

The rapid advancements in machine learning (ML) have revolutionized various real-world robotic tasks that heavily rely on fast and energy-efficient inference of computationally intensive ML models. To enhance inference performance, distributed inference utilizing GPU servers (e.g., edge devices with powerful GPUs) via the Internet of Things for robots (robotic IoT) has emerged as a promising approach, with techniques such as data parallelism, tensor parallelism, and pipeline parallelism proving effective in modern data centers. However, when deployed on real-world robots, existing parallel methods fail to provide high inference speed and meet the energy requirements due to the limited bandwidth of robotic IoT and the real-time demands of robotic applications, necessitating the development of a novel parallel pattern tailored to the unique constraints and requirements of robotic IoT.

We present Intra-DP, a high-performance distributed inference system optimized for ML model inference on robotic IoT. Intra-DP employs a fine-grained parallel pattern at the granularity of local operators within model layers (i.e., operators that can be computed independently with partial input, such as the convolution kernel in the convolution layer), enabling concurrent computation and transmission of different local operators from various layers. By doing so, Intra-DP effectively overlaps the computation and transmission phases within the same inference task, dramatically reducing transmission bottlenecks caused by the limited bandwidth of robotic IoT. The evaluation demonstrate that Intra-DP reduces inference time by 14.9% to 41.1% and energy consumption per inference by up to 35.3% compared to the state-of-the-art baselines.

1 Introduction

The rapid progress in machine learning (ML) techniques has led to remarkable achievements in various fundamental robotic tasks, such as object detection [23,34,38], robotic control [27,55,65], and environmental perception [5,28,60]. However, deploying these ML applications on real-world robots requires fast and energy-efficient inference of their ML models, given the need for swift environmental responses and the limited battery capacity of robots. Placing the entire model on robots not only necessitates substantial economic expenditures on additional computing accelerators (e.g., GPU [42], FPGA [43], SoC [17]) but also introduces significant energy consumption (e.g., an average 2.5× increase in energy consumption per unit time on our robots) due to the computationally intensive nature of ML models.

Distributed inference utilizing GPU servers (e.g., edge devices with powerful GPUs) via the Internet of Things for robots (robotic IoT) has emerged as a promising approach to meet the latency requirements of robotic applications and extend the battery lifetime of robots. This approach accelerates the inference process by leveraging the high computing capabilities of GPU servers and alleviates the local computational burden, thereby reducing energy consumption. In addition to the all offload method that places the entire ML model onto GPU servers, various parallel methods have been proposed and proven efficient in modern data centers [19, 61, 71] to further enhance inference performance. In data centers, there are three main parallel patterns: 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).

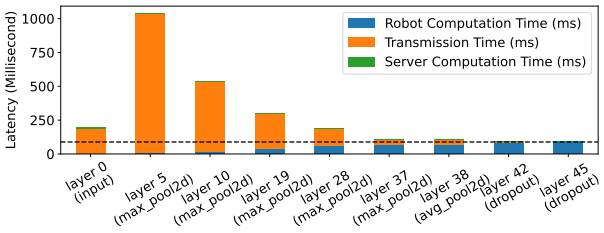
However, all existing parallel patterns for distributed inference in the data center are ill-suited for robotic IoT due to the unique constraints and requirements of robotic applications. For DP, its scalability is limited by the total batch size [40], and the small batch size inherent to robotic applications (typically 1) hinders the mini-batch computation, rendering DP inapplicable for robotic applications. 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), which cannot be computed as a complete input for each inference.

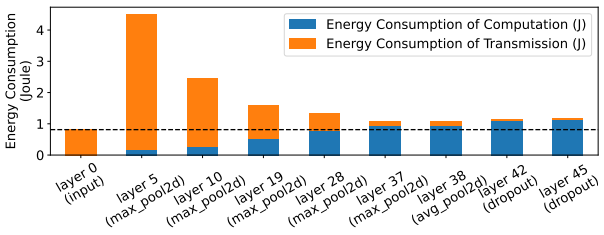
TP requires frequent synchronization among devices, leading to unacceptable communication overhead in robotic IoT. By partitioning parameter tensors of a layer across devices, TP allows concurrent computation on different parts of the tensor but requires an all-reduce communication [71] to combine computation results from different devices, entailing significant communication overhead. Consequently, TP is used mainly for large layers that are too large to fit in one GPU device in data centers (e.g., ChatGPT [59]) and require dedicated high-speed interconnects (e.g., 400 Gbps for NVLink [26]) even within data centers. In contrast, robots must prioritize seamless mobility and primarily depend on wireless connections with inherently limited bandwidth (see Sec. 2.1), making all-reduce synchronization an unacceptable overhead that dramatically slows down the entire inference process (e.g., the inference time with TP was up to 143.9× slower than local computation in Sec. 2.2).

Consequently, existing distributed inference approaches in robotic IoT are constrained to the PP paradigm. Since 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 [7], existing methods [4, 6, 18, 21, 29, 39, 63, 70] 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 [18], layer partitioning strategies constitute various trade-offs between computation and transmission and ultimately achieve much better performance than the all offload method, as shown in Fig. 1 (see more details in Sec. 2.2).

However, all existing methods based on the PP paradigm face significant transmission bottlenecks in robotic IoT. The PP paradigm on robotic IoT involves three sequential phases: computing early layers on robots, transmitting intermediate results, and completing inference on GPU servers, where the limited bandwidth of real-world networks often results in transmission time exceeding computation time. Despite the state-of-the-art (SOTA) layer partitioning strategies [6, 29], the transmission overhead becomes a substantial bottleneck, accounting for up to 70.45% of inference time in our experiments, due to the limited bandwidth of robotic IoT (see Sec. 2.1). This overhead not only slows down inference speed and consumes significantly more energy but also cannot be effectively mitigated by pipeline execution (overlapping computation and transmission phases across multiple inference tasks), which still fails to reduce the completion time of a single inference task [7], a crucial aspect for robotic applications.



(a) Inference Latency



(b) Energy consumption per inference on our robot

Figure 1: The performance of the PP paradigm with various layer partitioning strategies on VGG19 [50] in our experiments. 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.

The primary limitation of existing methods lies in their sequential execution of layer partitioning, which partitions a single inference task into three sequential phases at the layer granularity, hindering parallel execution within the scope of an individual inference task. As transmission time constitutes a substantial portion of the total inference time (approximately half) in these methods, a novel parallel pattern that efficiently overlaps computation and transmission within the same inference task has the potential to significantly reduce the robot's idle time during the last two phases (transmitting intermediate results and waiting for the final inference result from GPU servers). This reduction in idle time not only achieves fast inferences but also decreases overall energy consumption. Since the main energy consumption during idle time on robots comes from chips like CPU, GPU, and memory (e.g., 95% energy consumption in our experiments) rather than the network cards for transmission, this new parallel pattern will only slightly increase energy consumption during the computation phase for the network card while dramatically reducing the idle time during the transmission phase, ultimately leading to

a decrease in overall energy consumption.

To achieve such a new parallel pattern, a finer scheduling granularity is required. Fortunately, we found that local operators are a suitable choice, as they are the smallest unit of calculation in the inference process. Operators for each model 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 with partial input. For instance, softmax [35] requires the complete input vector to calculate the corresponding probability distribution, referring to it as a global operator, while ReLU [8] and convolution [39] can be computed with partial 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 are widely used in robotic applications, especially convolution layers in computer vision [38] and point cloud tasks [55], making them an ideal candidate for the proposed fine-grained parallel pattern.

In this paper, we present Intra-DP (Intra-Data Parallel), a high-performance distributed inference system optimized for real-world robotic IoT. Intra-DP adopts a fine-grained parallel pattern at the granularity of local operators and effectively overlaps the computation and transmission of different local operators from various layers within the same inference task. By doing so, Intra-DP dramatically reduces transmission bottlenecks caused by the limited bandwidth of robotic IoT, achieving 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 operators. We propose Local Operator Parallelism (LOP), which reduces the granularity of calculation from each layer to each local operator. LOP first determines data dependencies among local operators, where the output of one local operator serves as the input for the next. For local operator layers, Intra-DP only changes the execution sequence of local operators and ensures each local operator gets the correct input (raw input or the output from the operator of the previous layer) according to its data dependency. 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. By ensuring that all operators still receive the correct input, Intra-DP guarantees the calculation correctness of both local operator layers and global operator layers.

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. Intra-DP addresses this challenge by introducing a novel Local Operator Scheduling Strategy (LOSS) that determines the optimal allocation of local operators between robots and GPU servers, while considering the transmission cost of LOP to ensure each operator gets the required input in time. LOSS formulates this problem 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 [47]. Moreover, Intra-DP allows the re-calculation of some local operators (let them compute simultaneously on both the robots and GPU servers, thus reducing the transfer required by LOP), which greatly reduces the expensive transmission by slightly increasing the energy consumption of computation, ultimately optimizing the overall performance of Intra-DP in robotic IoT.

We implemented Intra-DP in PyTorch [45] and evaluated Intra-DP on our real-world robots under two typical real-world robotic applications [38, 55] and several models common to mobile devices on a larger scale [50, 51, 53, 57, 62]. We compared Intra-DP with four baselines under different real-world robotic IoT network environments (namely indoors and outdoors): local computation, which places the entire ML model on the robot; all offload, which places the entire ML model onto GPU servers; and two state-of-the-art pipeline parallelism methods, DSCCS [29], aimed at accelerating inference, and SPSO-GA [6], focused on optimizing energy consumption. Evaluation shows that:

- Intra-DP is fast. Intra-DP reduced inference time by 14.9% ~41.1% compared to baselines under indoors and outdoors environments.
- Intra-DP is energy-efficient. Intra-DP reduced up to 35.3% energy consumption per inference compared to baselines, due to faster inference speed and limited-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 pattern based on local operators, and LOSS, a new scheduling strategy based on LOP optimized for distributed inference in robotic IoT. 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 in 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, enabling the widespread adoption of advanced machine learning techniques in robotic applications. Intra-DP's code is released on https://github.com/nsdi25paper273/intraDP.

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: TODO Average transmission time (Second), inference time (Second), percentage that transmission time accounts for of the total inference time and their standard deviation ($\pm n$) with TP on different models in different environments. "Local computation" refers to inference the entire model locally on the robot.

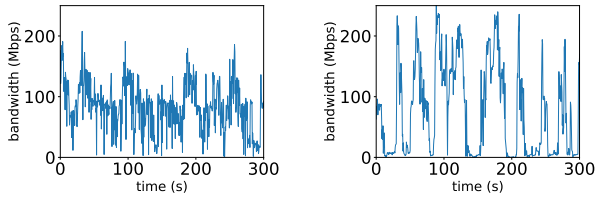
2 Background

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 [33]. However, the limited hardware resources on robots often prevent them from fully utilizing the potential of Wi-Fi 6 [64]. Moreover, the actual available bandwidth of wireless networks is often reduced in practice due to various factors, such as the movement of devices [37, 44], occlusion by physical barriers [10, 49], and preemption of the wireless channel by other devices [2, 48].

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' SOTA computation and communication capabilities. We utilized iperf [1] to saturate the wireless transmission between our robot and a base station in the indoors and the outdoors scenarios, thereby measuring the real-time maximum wireless bandwidth capacity and recording these values every 0.1 seconds over a period of 5 minutes.

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



(a) Indoors (b) Outdoors

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

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 [41] or PCIe [26], offering bandwidths ranging from 40 Gbps to 500 Gbps. The bandwidth available between robots and GPU servers in robotic IoT networks is typically more limited, both due to the theoretical upper limit of wireless transmission technologies and the practical instability of wireless networks.

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 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 [54], 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 operator layers.

Layer partitioning. Existing distributed inference approaches [6, 29] in robotic IoT primarily adopt the PP paradigm and focus on layer partitioning to achieve fast and energy-efficient inference. These approaches can be categorized into two main groups based on their optimization goals: accelerating inference for diverse DNN structures [18, 24, 29, 39, 63] and optimizing robot energy consumption during inference [6, 30, 58]. The widespread use of layer partitioning to improve inference performance in robotic applications has attracted many researchers to study this field [4, 18, 21, 39, 63, 70], 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. It is important to note that the native offload method, which places the entire ML model onto GPU servers, can be considered as a special case of layer partitioning, where all layers are placed onto the GPU server. However, all existing layer partitioning methods suffer from the transmission bottleneck inherent to PP's parallel pattern, 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 ML model compression on the robots. Quantization [9, 14, 15] 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 [16, 32, 56], 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 tasks across multiple devices.

Inference Job 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 [69], prioritizing based on urgency [31, 36], and employing deep reinforcement learning controls [3, 11, 12]. Unlike distributed inference methods aim to optimize each individual inference, these methods focus on minimizing overall inference latency and energy consumption, while adopting existing distributed inference techniques for the execution of each individual inference task. Intra-DP provides these methods with a new parallel pattern (LOP), which demonstrates much higher performance in robotic IoT, and these inference job 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

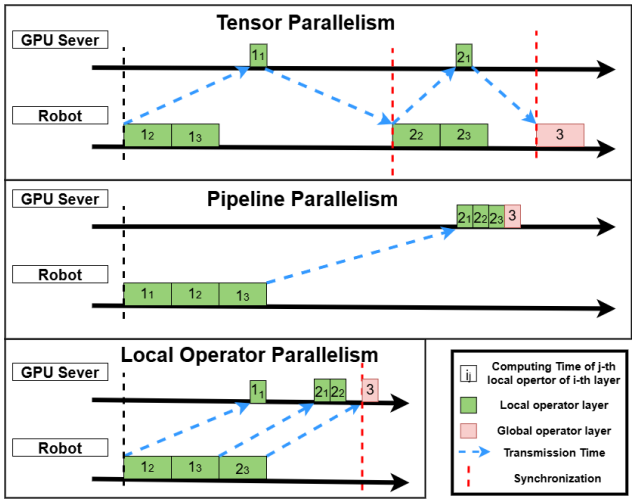


Figure 3: Workflow of TP, PP and LOP. In the three cases above, each local operator layer consists of three local operators with identical computation times on robots and GPU servers, as well as corresponding transmission times. Due to the smaller input tensor volume of layer 2 compared to layer 1, the layer partitioning strategy at layer 1 results in shorter transmission times, leading the PP method to select such strategy at layer 1 [29].

Fig. 3 illustrates the workflow of Intra-DP and compares it with TP and PP in robotic IoT networks with limited bandwidth, highlighting the transmission overhead issues faced by

existing methods and how Intra-DP addresses them through its LOP. Since local operator layers require multiple calculations of a single local operator (e.g., convolution kernel), we treat each calculation of the local operator as an independent local operator in this article for easy discussion.

Despite TP's ability to execute some local operators on the GPU server, it incurs significant communication overhead due to the all-reduce communication [71] required to combine computation results from different devices (shown by the red dotted lines for synchronization in Fig. 3). PP's layer partition algorithm [29] minimizes overall inference time but still suffers from transmission time bottlenecks (shown by the extremely long blue dotted lines for transmission in Fig. 3).

To alleviate the transmission overhead in distributed inference, Intra-DP overlaps the computation and transmission of local operators from different local operator layers (as illustrated in Fig. 3 of LOP, Intra-DP transmits the input of the local operator '$LO1_1$' 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 during all-reduce communication for local operator layers, maintaining all-reduce communication only for global operator layers. However, this workflow in LOP necessitates that local operators on the GPU server receive the correct input in a timely manner (as exemplified in Fig. 3, '$LO2_1$' directly uses the computational result of '$LO1_1$' on the GPU server as its input), which will be discussed in greater detail in Sec. 4. Although LOP requires more communication compared to PP, Intra-DP's overlapping significantly reduces transmission completion time by initiating data transfer much earlier than PP, which can only begin transmission after the entire computation of layer 1 is finished in Fig. 3 (see more details in Sec. 6.2). As a result, Intra-DP dramatically reduce the idle time on robots compared with existing distributed inference methods, achieving much faster inference.

Moreover, the idle time on the robot (when not computing) 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 [25]. 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, as shown in Tab. 3. Although LOP's approach of overlapping the computation and transmission phases among different local operators of various layers slightly increases energy consumption from the wireless network card during computing (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.

To realize the workflow depicted in Fig. 3, Intra-DP must address two key challenges: guaranteeing the correctness of inference results based on local operators (see Sec. 4.1) and efficiently scheduling the computation and transmission of each local operator (see Sec. 4.2).

3.2 Architecture of Intra-DP

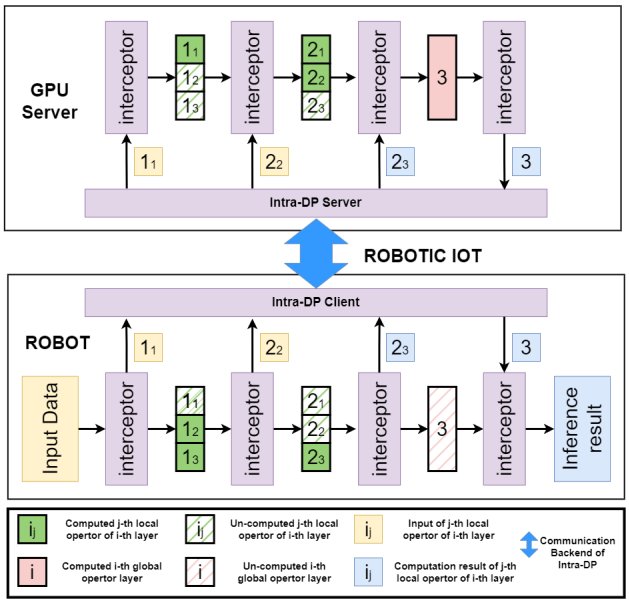


Figure 4: Architecture of Intra-DP. The core components of Intra-DP are highlighted in purple. Intra-DP adopts the same scheduling scheme as in Fig. 3.

Fig. 4 illustrates the architecture of Intra-DP, which incorporates interceptors for each layer to enable flexible splitting of the input tensor and combining of the output tensor for each operator. There are three stages of the framework, including profiling, scheduling optimization, and runtime.

Profiling. The profiling stage collects essential runtime traces for optimization step and only needs to be done once. Those traces include: 1) the execution time of operators on the robot and GPU server; 2) the type and input/output sizes of operators; 3) the data dependencies among operators (i.e., the output of one operator serves as the input for the another).

Scheduling Optimization. Based on the profiling traces, Intra-DP generates deployment strategies via its LOSS to determine the optimal allocation of local operators 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. It is crucial to 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 these 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 predicts network bandwidth using mature tools [67] in wireless transmission and adopts corresponding deployment strategy based on the predicted bandwidth. To ensure flexible switching among various deployment strategies, Intra-DP maintains a copy of the model on the GPU server at 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 transfer processes of the Intra-DP client and server, while local operators 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, causing prolonged waiting. Intra-DP formulates this waiting time into a nonlinear optimization problem in its LOSS, minimizing waiting time and implementing highly parallel scheduling schemes on local operators. In this way, Intra-DP achieves negligible extra system cost and faster inference via LOP and LOSS.

4 Detailed Design

4.1 Local Operator Parallelism

LOP guarantees the correctness of inference results by ensuring that all operators still receive the correct input. First, we analyze the input required for each operator based on their calculation characteristics and processes, and divide them into local operators and global operators according to whether they can be computed independently with partial input. Here, we summarize three classes of local operators common in models used on mobile devices:

- Element-wise local operator. This class of operators computes 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 [8], Sigmoid [66], and SiLU [22]. However, it is important to note that some activation functions, like softmax [35], require all elements for computation and are not considered local operators, but global operators.

- Block-wise local operator. This class of operators requires a block at the corresponding position in the input tensor and is widely used in layers associated with convolution, such as convolution [39] and maxpool [52]. The size of the input blocks is determined by the parameters set by the corresponding layer [46], including the size of the convolution kernel, padding, and dilation.

- Row-wise 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 [68] and multiplication [13]. Row-wised local operators 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 operation layer in the same way.

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

  It is important to note that the way row-wised local operators split matrices is in contrast to TP, which splits the layer parameter matrix and transfers a copy of the input matrix to different devices. Splitting in rows allows the next matrix to be directly computed based on the partial result of the previous one without all-reduce communication. Moreover, LOP treats layers with layer parameter matrices containing only one row as global operators.

Based on the above definition of local operators, we calculate their proportion in the models commonly used on mobile devices, as shown in Tab. 2, and find that they exist widely in these models. It is important to note that global operators are not the last layer, but the layers that must be computed with complete input.

Model	Number of local operator layers	Number of global operator layers
DenseNet [20]	428	3
ResNet [53]	341	3
ConvNeXt [57]	340	6
RegNet [62]	231	3

Table 2: TODO: The number of local and global operator layers in various AI models.

Despite the execution time on the robot and GPU server, the local/global type, and the input/output of each operator, the data dependencies among operators are also needed in the profiling stage. LOP builds the data dependencies among operators by determining if the output of one operator serves as the input of another. The input and output of these two operators may not be exactly the same, but as long as they are partially the same, the dependency is established. It is important to note that when the output is used by several local operators on the robots and GPU servers simultaneously, especially for block-wise local operators, LOP allows the re-calculation of these operators, letting them compute simultaneously on both the robots and GPU servers (see more details in Sec. 4.2). In

this way, Intra-DP reduces synchronization with high transmission costs in robotic IoT by introducing a small amount of redundant calculation.

4.2 Local Operator Scheduling Strategy

LOSS formulates the problem of scheduling local operators as a nonlinear optimization problem, modeled as follows:

First, we define OP_i as the set of operators in the i_{th} layer, including both local and global operators. When the i_{th} layer is a global operator layer, $|OP_i| = 1$, as it only has one operator. We then define $X_i \subseteq OP_i$ as the set of operators located on robots and $Y_i \subseteq OP_i$ as the set of operators executed on the GPU server, where $X_i \cup Y_i = OP_i$. $X_i \cap Y_i \neq \emptyset$ when some local operators of i_{th} layer 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} layer on robots as T^i_{robot} and that on the GPU server as T^i_{server}, as shown in Fig. 3. 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} layer is the last layer of $parent(X_i)$. The consideration that an operator may have several parents allows Intra-DP to support DNN models with complex structures as directed acyclic graphs, such as MobileNet [51] and ResNet [53]. The *MAX* function is used to minimize the idle time when combining the input tensor of this layer, 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 a DNN model with N layers as a nonlinear optimization problem:

$$\min \quad T^N_{robot} \tag{1}$$

$$\text{s.t.} \quad M_i = \emptyset, \forall i \in \Pi \tag{2}$$

$$N_i = \emptyset, \forall i \in \Pi \tag{3}$$

Here, the layers 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 [18].

To tackle the nonlinear and non-convex nature of the objective function, LOSS employs the differential evolution algorithm [47] to solve the optimization problem, and schedules the computation and transmission of each local operator based on the obtained solution. It is important to note that when applying Intra-DP to a special model without any local operator layers, LOSS will degrade to the existing layer partitioning method.

4.3 Algorithms of Intra-DP

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^b_i, M^b_i,N^b_i: 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 = PredictsBandwidth()
Z_0 = input
foreach i_th layer in model do
    if M_i^b ≠ ∅ then
        Z_i = combine(Z_i, ReceiveFromServer())
    end
    if N_i^b ≠ ∅ then
        SendToServer(Z_i, N_i^b)
    end
    if X_i^b ≠ ∅ and Z_i ≠ ∅ then
        Z_{i+1} = compute(Z_i, X_i^b)
    end
    else
        Z_{i+1} = ∅
    end
end
ret = Z_{N+1}
return ret
```

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.

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

1 $model, info\_robot = ReceiveFromClient()$
2 $info\_server = ProfileModel(model)$
3 $X, Y, M, N = LOSS(info\_robot, info\_server)$
4 $SendToClient(X, M, N)$
// runtime stage on server
5 $b = TestBandwidth()$
6 $Z_0 = \emptyset$
7 **foreach** i_{th} *layer in model* **do**
8 **if** $M_i^b \neq \emptyset$ **then**
9 $SendToClient(Z_i, M_i^b)$
10 **end**
11 **if** $N_i^b \neq \emptyset$ **then**
12 $Z_i = combine(Z_i, ReceiveFromClient())$
13 **end**
14 **if** $Y_i^b \neq \emptyset$ *and* $Z_i \neq \emptyset$ **then**
15 $Z_{i+1} = compute(Z_i, Y_i^b)$
16 **end**
17 **else**
18 $Z_{i+1} = \emptyset$
19 **end**
20 **end**

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 nonlinear optimization problems in finite time remains an open challenge, and developing an enhanced algorithm that provides fast, high-quality solutions for the non-convex optimization problem in LOSS is left for future work. Lastly, during the runtime stage, Intra-DP selects the appropriate schedule plan based on the actual bandwidth conditions, leveraging the model copy on the GPU server (Fig. 4, line 1 in Alg. 2) for flexible schedule plan switching.

5 Implementation

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 ML 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.

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

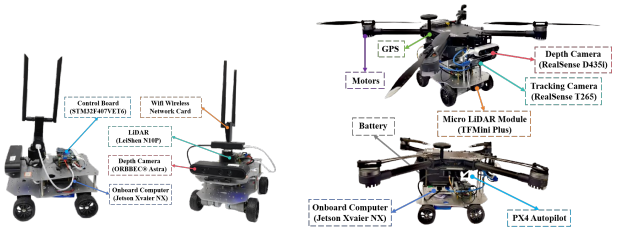
1 **foreach** b *bandwidth less than* BW **do**
// Initialize
2 **foreach** i_{th} *layer in model* **do**
3 $\mathcal{X}_i^{init} = OP_i$
4 $\mathcal{Y}_i^{init} = \emptyset$
5 **end**
// add constraints
6 **foreach** i_{th} *layer in* Π **do**
7 $constraints.add(\mathcal{M}_i = \emptyset)$
8 $constraints.add(\mathcal{N}_i = \emptyset)$
9 **end**
// objective function
10 $obj\_func = T_{robot}^N(info\_robot, info\_server, (\mathcal{X}_0, .., \mathcal{X}_N, \mathcal{Y}_0, .., \mathcal{Y}_N), b)$
// solve optimization problems
11 $X_0^b, .., X_N^b, Y_0^b, .., Y_N^b = differential\_evolution\_solver(obj\_func, constraints, (\mathcal{X}_0^{init}, .., \mathcal{X}_N^{init}, \mathcal{Y}_0^{init}, .., \mathcal{Y}_N^{init}))$
12 **foreach** i_{th} *layer in model* **do**
13 $M_i^b = parent\left(X_i^b\right) - X_{i-1}^b$
14 $N_i^b = parent\left(Y_i^b\right) - Y_{i-1}^b$
15 **end**
16 **end**
17 **return** $\bigcup_{b=1}^{BW}\bigcup_{i=0}^{N} X_i^b$, $\bigcup_{b=1}^{BW}\bigcup_{i=0}^{N} Y_i^b$, $\bigcup_{b=1}^{BW}\bigcup_{i=0}^{N} M_i^b$, $\bigcup_{b=1}^{BW}\bigcup_{i=0}^{N} N_i^b$

```
# 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 [50] 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 [42] 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) i5 12400f CPU @ 4.40GHz and an NVIDIA GeForce GTX 2080 Ti 11GB GPU, connected to our robot via Wi-Fi 6 over 80MHz channel at 5GHz frequency in our experiments.



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

Figure 6: The detailed composition of the robot platforms

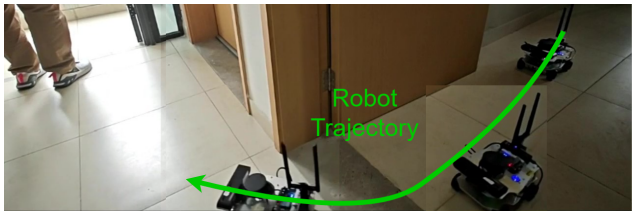
	inference	transmission	standby
Power (W)	13.35	4.25	4.04

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

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 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.

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

(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 [38].

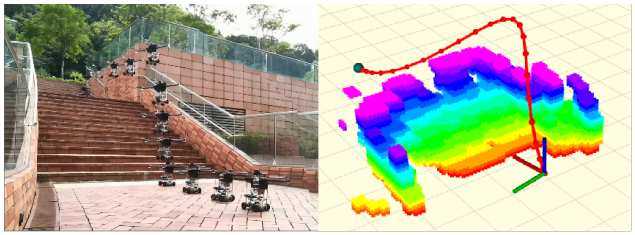


Figure 8: By predicting occlusions in advance, AGRNav [55] gains an accurate perception of the environment and avoids collisions, resulting in efficient and energy-saving paths.

outputs key points of small data volume. In contrast, AGRNav takes point clouds as input and outputs predicted point clouds and semantics of similar data volume as input, implying that AGRNav needs to transmit more data during distributed inference. And we have verified several models common to mobile devices on a larger scale to further corroborate our observations and findings: DenseNet [20], VGGNet [50], ConvNeXt [57], RegNet [62].

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 selected two SOTA pipeline parallelism methods as baselines: DSCCS [29], aimed at accelerating inference, and SPSO-GA [6], focused on optimizing energy consumption. We set SPSO-GA's deadline constraints to 1

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	0.89(±0.16)	0.89(±0.16)	10.61(±0.48)	10.61(±0.48)	9.48(±1.73)	9.48(±1.73)
	ALL	0.49(±0.26)	0.52(±0.20)	5.49(±0.24)	5.38(±0.2)	2.67(±0.11)	2.78(±0.1)
	DSCCS	0.43(±0.29)	0.45(±0.25)	6.38(±2.21)	6.64(±2.38)	2.74(±0.95)	3.0(±1.07)
	SPSO-GA	0.38(±0.21)	0.41(±0.17)	5.5(±1.53)	5.35(±1.37)	2.1(±0.59)	2.18(±0.56)
	Intra-DP	0.33(±0.18)	0.37(±0.17)	7.4(±1.31)	7.94(±0.79)	2.42(±0.43)	2.91(±0.29)
agrnav(0.84M)	Local	0.52(±0.07)	0.52(±0.07)	8.11(±0.26)	8.11(±0.26)	4.26(±0.59)	4.26(±0.59)
	ALL	0.83(±0.49)	0.98(±0.48)	4.57(±0.14)	4.57(±0.12)	3.81(±0.11)	4.46(±0.12)
	DSCCS	0.43(±0.16)	0.51(±0.21)	5.42(±1.35)	5.19(±1.29)	2.31(±0.57)	2.64(±0.66)
	SPSO-GA	0.44(±0.16)	0.47(±0.11)	6.32(±1.65)	6.7(±1.61)	2.77(±0.72)	3.18(±0.76)
	Intra-DP	0.34(±0.15)	0.36(±0.12)	8.1(±1.15)	8.4(±0.81)	2.75(±0.39)	2.99(±0.29)

Table 4: TODO: Average transmission time, inference time, percentage that transmission time accounts for of the total inference time and their standard deviation ($\pm n$) of Kapao and AGRNav in different environments with different systems. "Local computation" refers to inference the entire model locally on the robot.

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.

The evaluation questions are as follows:

- RQ1: How 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 reduce the transmission bottleneck in robotic IoT?
- 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

6.1.1 Inference Time

The upper part of Tab.4 demonstrates that Intra-DP significantly reduced Kapao's inference time compared to SPSO-GA and DSCCS, both indoors and outdoors. Specifically, Intra-DP achieved a 16.3% and 14.9% reduction indoors, and a 23.7% and 15% reduction outdoors, compared to SPSO-GA and DSCCS, respectively. For AGRNav, the performance gain of Intra-DP and baselines varied, as shown in the lower part of Tab.4. Intra-DP reduced AGRNav's inference time by 36.2% and 27.8% indoors, and 41.1% and 30.8% outdoors, compared to SPSO-GA and DSCCS, respectively.

Transmission time accounts for up to 70.45% of the total inference time in SPSO-GA and DSCCS, highlighting the significant transmission bottlenecks faced by existing methods based on the PP paradigm, even with state-of-the-art layer partitioning. 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. Intra-DP's transmission time cannot reach close to 100% of its inference time because it can only overlap the execution of local operators, and not all layers in the models of Kapao and AGRNav are local operator layers. Consequently, Intra-DP can only parallelize execution in some layers, not all layers.

The large standard deviation in transmission time outdoors for all systems indicates that bandwidth fluctuated more frequently and more fiercely outdoors compared to indoors, which is consistent with the observations in Fig. 2. Furthermore, the lower average bandwidth for outdoor scenarios (see Sec. 2.1) results in increased transmission and inference times relative to indoor scenarios.

6.1.2 Energy Consumption

Table **??** presents the power consumption over time and energy consumption per inference for Kapao and AGRNav using Intra-DP and baseline methods. Compared to SPSO-GA and DSCCS, Intra-DP exhibits higher power consumption over time. This can be attributed to Intra-DP's computation at the operator granularity, where finer granularity results in lower GPU resource utilization and enables repeated computation of certain operators to avoids synchronization in LOSS.

Despite the higher power consumption over time, Intra-DP achieves the lowest energy consumption per inference for both Kapao and AGRNav, primarily due to its shortest inference time. Intra-DP avoids the need for synchronization with high transmission costs in robotic IoT by introducing a small amount of redundant computation. The additional energy consumed during Intra-DP's computation phase is

Model	System	Metrics	Robot computation		Transmission		Server computation	
			indoors	outdoors	indoors	outdoors	indoors	outdoors
kapao(77M)	Local	Time(s)	0.89(±0.16)	0.89(±0.16)	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(±0.01)	0.06(±0.01)	0.25(±0.16)	0.27(±0.12)	0.20(±0.16)	0.22(±0.12)
		Percentage(%)	11.5(±1.1)	10.8(±1.1)	51.6(±33.0)	51.7(±23.2)	41.9(±32.9)	42.6(±23.0)
	DSCCS	Time(s)	0.17(±0.24)	0.16(±0.24)	0.13(±0.09)	0.17(±0.09)	0.16(±0.24)	0.15(±0.24)
		Percentage(%)	38.5(±55.7)	34.8(±52.3)	30.0(±20.2)	37.7(±19.6)	36.4(±56.0)	32.6(±52.7)
	SPSO-GA	Time(s)	0.10(±0.15)	0.10(±0.16)	0.25(±0.17)	0.28(±0.14)	0.05(±0.01)	0.05(±0.01)
		Percentage(%)	25.5(±40.5)	24.6(±39.9)	66.5(±43.5)	68.2(±33.2)	13.0(±2.9)	12.2(±2.8)
	Intra-DP	Time(s)	0.17(±0.09)	0.19(±0.09)	0.15(±0.10)	0.17(±0.10)	0.09(±0.12)	0.11(±0.13)
		Percentage(%)	52.2(±28.3)	50.9(±23.2)	46.0(±29.5)	47.6(±27.8)	28.9(±36.6)	29.3(±34.9)
agrnav(0.84M)	Local	Time(s)	0.52(±0.07)	0.52(±0.07)	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(±0.01)	0.44(±0.27)	0.52(±0.26)	0.42(±0.26)	0.50(±0.26)
		Percentage(%)	1.0(±0.5)	1.1(±0.9)	53.4(±31.9)	53.1(±26.5)	50.6(±31.8)	50.8(±26.4)
	DSCCS	Time(s)	0.07(±0.22)	0.06(±0.22)	0.28(±0.17)	0.35(±0.17)	0.10(±0.12)	0.12(±0.11)
		Percentage(%)	15.8(±51.6)	12.4(±42.9)	65.4(±41.0)	68.5(±33.3)	23.8(±28.8)	24.2(±22.0)
	SPSO-GA	Time(s)	0.12(±0.28)	0.14(±0.28)	0.24(±0.20)	0.26(±0.23)	0.10(±0.12)	0.10(±0.11)
		Percentage(%)	26.7(±63.5)	28.9(±58.9)	54.7(±46.1)	54.3(±48.9)	23.6(±27.6)	21.8(±23.8)
	Intra-DP	Time(s)	0.16(±0.07)	0.18(±0.06)	0.17(±0.11)	0.18(±0.07)	0.11(±0.21)	0.11(±0.16)
		Percentage(%)	48.3(±21.5)	49.6(±16.3)	50.3(±33.6)	49.3(±19.0)	31.2(±61.3)	29.8(±43.9)

Table 5: TODO: breakdown

significantly lower than the energy wasted by the prolonged inference times of SPSO-GA and DSCCS. Although SPSO-GA aims to optimize energy consumption, its advantages in power consumption over time diminish when considering energy consumption per inference due to extended inference times. This is because SPSO-GA solely focuses on minimizing power consumption over time, potentially at the cost of prolonged inference time.

6.2 Breakdown

6.3 Validation on a larger range of models

We evaluated Intra-DP and baselines on a wide range of models commonly used in mobile devices, with parameter counts varying (detailed in Tab. 6 and Tab. **??**). Our results confirm that transmission time constitutes a significant portion of the total inference time in both DSCCS and SPSO-GA, leading to wasteful inference time and energy consumption compared to Intra-DP. Although Intra-DP's outperformance remains consistent across various models, we observed that the performance gain is relatively smaller on models with fewer parameters. This is because Intra-DP's performance improvement is primarily achieved through the parallel execution of local operators. When a model employs more global operator layers or has fewer parameters, the number of local operators available for parallel execution is reduced, limiting the optimization potential for Intra-DP to enhance performance.

6.4 Lessons learned

Model structure. During the implementation and evaluation of Intra-DP, we discovered that the presence of more local operator layers 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 instead of directly transferring the full input tensor and re-calculating).

Future work. It is of interest to explore further improvements of Intra-DP, such as a distributed inference system for multi-robot to minimize overall inference time and energy consumption. Such advancements could enable faster and more robust wireless distributed inference in real-world robotic IoT.

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

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: TODO: Average transmission time, inference time, percentage that transmission time accounts for of the total inference time and their standard deviation ($\pm n$) of common AI models in different environments with different systems.

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 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] 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.

[4] 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.

[5] 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.

[6] 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.

[7] 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.

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

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

[10] 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.

[11] 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.

[12] 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.

[13] 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.

[14] 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.

[15] 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.

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

[17] 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.

[18] 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.

[19] 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.

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

[21] 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.

[22] 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.

[23] 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.

[24] 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.

[25] 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.

[26] 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.

[27] 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.

[28] 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.

[29] 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.

[30] 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.

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

[32] 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.

[33] 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.

[34] 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.

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

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

[37] 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.

[38] 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.

[39] 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.

[40] 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.

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

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

[43] 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.

[44] 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.

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

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

[47] 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.

[48] 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.

[49] 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.

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

[51] 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.

[52] 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.

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

[54] 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.

[55] 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.

[56] 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.

[57] 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.

[58] 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.

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

[60] 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.

[61] 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.

[62] 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.

[63] 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.

[64] 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.

[65] 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.

[66] 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.

[67] 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.

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

[69] 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.

[70] 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.

[71] 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.