

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

Anonymous Author(s)

Submission Id: 445*

Abstract

The rapid advancements in machine learning (ML) techniques have led to significant achievements in various robotic tasks, where deploying these ML approaches on real-world robots requires fast and energy-efficient inference of their deep neural network (DNN) models. Distributed inference, which involves inference across multiple powerful GPU devices, has emerged as a promising optimization to improve inference performance in modern data centers. However, when deployed over Internet of Things of these real-world robots (robotic IoT), all existing parallel methods (data parallelism, tensor parallelism, pipeline parallelism) fail to simultaneously meet the robots' latency and energy requirements and raise significant challenges due to the failure of data parallelism, the unacceptable communication overhead of tensor parallelism, and the significant transmission bottlenecks in pipeline parallelism due to the limited bandwidth of robotic IoT.

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

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

1 Introduction

The rapid progress in machine learning (ML) techniques has led to remarkable achievements in various fundamental robotic tasks, such as object detection [17, 27, 29], robotic control [21, 43, 53], and environmental perception [4, 22, 48]. However, deploying these ML applications on real-world robots requires fast and energy-efficient inference of their deep neural network (DNN) models, given the need for swift

environmental responses and the limited battery capacity of robots. Placing the entire model on robots not only requires additional computing accelerators on robots (e.g., GPU [33], FPGA [34], SoC [14]), but also introduce additional energy consumption (e.g., 162% more for [29] in our experiments) due to the computationally intensive nature of DNN models, while placing the entire model in the cloud brings an extended response delay.

Distributed inference, which involves inference across multiple GPU devices, has emerged as a promising approach to meet the latency requirements of robotic applications and extend the battery lifetime of robots. This paradigm has been widely adopted in data centers [16, 49, 54], where numerous GPUs are utilized to speed large model inference, such as in the case of ChatGPT [47]. Adopting distributed inference across robots and other powerful GPU devices through the Internet of Things for these robots (robotic IoT) not only accelerates the inference process by leveraging the high computing capabilities of powerful GPUs but also alleviates the local computational burden, thereby reducing energy consumption, making it an ideal solution for robotic applications.

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

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

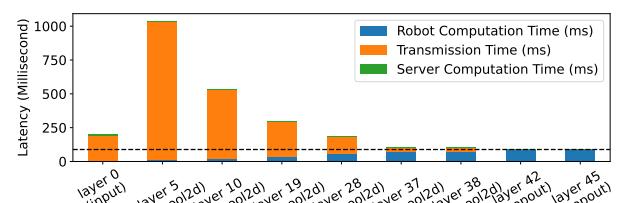
TP requires frequent synchronization among devices, leading to unacceptable communication overhead in robotic IoT.

111 By partitioning parameter tensors of a layer across GPUs,
 112 TP allows concurrent computation on different parts of this
 113 tensor but requires an all-reduce communication [54] to
 114 combine computation results from different devices, which
 115 entails significant communication overhead. Consequently,
 116 TP is used mainly for large layers that are too large to fit in
 117 one device in data centers and require dedicated high-speed
 118 interconnects (e.g., 400 Gbps for NVLink [20]) even within
 119 data centers. On the contrary, robots must prioritize seam-
 120 less mobility and primarily depend on wireless connections,
 121 which inherently possess limited bandwidth, as described in
 122 Sec. 2.1, making all-reduce synchronization an unacceptable
 123 overhead (e.g., the inference time with TP was up to 143.9X
 124 slower than local computation in Sec.2.3).

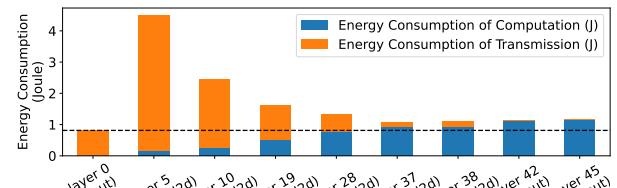
125 Consequently, existing distributed inference approaches [5,
 126 23] in robotic IoT primarily adopt the PP paradigm and fo-
 127 cuses on layer partitioning of PP, aiming to achieve fast and
 128 energy-efficient inference. This is because the PP paradigm
 129 in data centers consists of layer partitioning and pipeline
 130 execution, where the pipeline execution of PP enhances infer-
 131 ence throughput rather than reducing the completion time of
 132 a single inference [6], which is the most critical requirement
 133 in robotic IoT. Based on the fact that the amounts of output
 134 data in some intermediate layers of a DNN model are sig-
 135 nificantly smaller than that of its raw input data [15], DNN
 136 layer partitioning strategies constitute various trade-offs be-
 137 tween computation and transmission, taking into account
 138 application-specific inference speed requirements and en-
 139 ergy consumption demands, as shown in Fig. 1.

140 However, existing methods based on PP face significant
 141 transmission bottlenecks in robotic IoT due to the inherent
 142 scheduling mechanism. PP paradigm on robotic IoT consists
 143 of three sequential phases: computing earlier DNN layers
 144 on robots, transmitting intermediate results, and completing
 145 inference on the GPU server. Despite optimal layer parti-
 146 tioning strategies from [5, 23], the transmission overhead
 147 becomes a substantial bottleneck due to limited bandwidth
 148 of robotic IoT, accounting for up to 69% of inference time
 149 in our experiments. While pipeline execution can mitigate
 150 this overhead by overlapping computation and transmis-
 151 sion phases across multiple inference tasks, it cannot reduce
 152 the completion time of a single inference task [6], which is
 153 crucial for robotic applications. Furthermore, the prolonged
 154 transmission phase not only slows down inference speed but
 155 also consumes significantly more energy.

156 The key reason for the problem of the above methods is
 157 that existing methods conduct layer-granulated scheduling,
 158 which prevents overlapping the transmission and computa-
 159 tion phases within the same inference task. As transmission
 160 time constitutes a substantial portion of the total inference
 161 time (approximately half) in existing methods, a novel par-
 162 allel method with finer granularity that overlaps computation
 163 and transmission within the same inference task has the



(a) Inference Latency



(b) Energy consumption on robot

Figure 1. Existing distributed inference approaches on VGG19 [39] in our experiments, which adopt PP paradigm with various layer partitioning scheduling strategies. The X-axis of the graph represents different layer partitioning strategies, where ‘layer i’ indicates that all layers up to and including the i-th layer are computed on the robot, while the subsequent layers are processed on the GPU server.

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

In this paper, we present Intra-DP (Intra-Data Parallel), a high performance distributed inference system optimized for real-world robotic IoT networks. We found that calculation of each DNN layer is done by corresponding operators (e.g., matrix multiplication, convolution, pooling), and operators can be categorized into two types: local operator and global operator, depending on whether the operators that can be computed independently without the complete input. For instance, (, such as the convolution kernel in the convolution layer) Such local operator are widely exist in robotic application, especially . enable to a novel parallel method called Local Operator Parallel (LOP).

166
 167
 168
 169
 170
 171
 172
 173
 174
 175
 176
 177
 178
 179
 180
 181
 182
 183
 184
 185
 186
 187
 188
 189
 190
 191
 192
 193
 194
 195
 196
 197
 198
 199
 200
 201
 202
 203
 204
 205
 206
 207
 208
 209
 210
 211
 212
 213
 214
 215
 216
 217
 218
 219
 220

The design of Intra-DP is confronted with two major challenges. The first one is how to guarantee the correctness of inference results in Intra-DP based on LOP.

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. Nonlinear optimization problem.

We implemented Intra-DP in PyTorch [36] and evaluated Intra-DP on our real-world robots under two typical real-world robotic applications [29, 43] and several models common to mobile devices on a larger scale. We compared Intra-DP with two SOTA pipeline parallelism methods as baselines: DSCCS [23], aimed at accelerating inference, and SPSO-GA [5], focused on optimizing energy consumption, under different real-world robotic IoT networks environments (namely indoors and outdoors). Evaluation shows that:

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

Our main contribution is Intra-DP. xxx We envision that the fast and energy-efficient inference of Intra-DP will foster the deployment of diverse robotic tasks on real-world robots in the field. Intra-DP's code is released on <https://github.com/xxx/xxx>.

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

2 background

2.1 Characteristics of Robotic IoT

In real-world robotic IoT scenarios, devices often navigate and move around for tasks like search and exploration. While wireless networks provide high mobility, they also have limited bandwidth. For instance, Wi-Fi 6, the most advanced Wi-Fi technology, offers a maximum theoretical bandwidth of 1.2 Gbps for a single stream [26]. However, not only the limited hardware resources on the robot can not fully play the potential of Wi-Fi 6 [52], but also the actual available bandwidth of wireless networks is often reduced in practice

due to factors such as movement of the devices [28, 35], occlusion from by physical barriers [8, 38], and preemption of the wireless channel by other devices [2, 37].

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

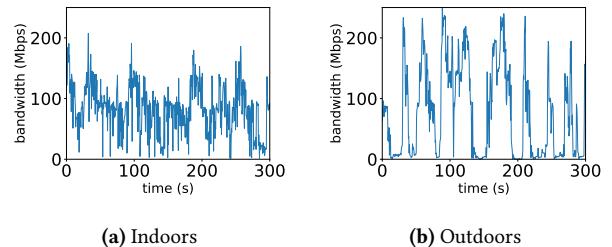


Figure 2. The instability of wireless transmission in robotic IoT networks.

The results in Fig. 2 show average bandwidth capacities of 93 Mbps and 73 Mbps for indoor and outdoor scenarios, respectively. The outdoor environment exhibited higher instability, with bandwidth frequently dropping to extremely low values around 0 Mbps, due to the lack of walls to reflect wireless signals and the presence of obstacles like trees between communicating robots, resulting in fewer received signals compared to indoor environments.

In summary, robotic IoT systems' wireless transmission is constrained by limited bandwidth, both due to the theoretical upper limit of wireless transmission technologies and the practical instability of wireless networks. Moreover, the unstable bandwidth in robotic IoT wireless networks can cause the transmission time to vary dramatically, sometimes changing by hundreds of times. In our experiments, even a relatively small model with only 0.84M parameters still suffers from its significant transmission overhead.

2.2 Characteristics of Data Center Networks

Data center networks, which are used for large model inference (e.g., ChatGPT [47]), are wired and typically exhibit higher bandwidth capacity and lower fluctuation compared to robotic IoT networks. GPU devices in data centers are interconnected using high-speed networking technologies such as InfiniBand [42] or PCIe [20], offering bandwidths ranging from 40 Gbps to 500 Gbps. The primary cause of bandwidth fluctuation in these networks is congestion on

331 intermediate switches, which can be mitigated through traffic scheduling techniques implemented on the switches [32].
 332 The stable and high-bandwidth nature of data center networks makes them well-suited for demanding tasks like
 333 large model inference, in contrast to the more variable and
 334 resource-constrained environments found in robotic IoT networks.
 335

336 2.3 Existing distributed inference methods in the 337 data center

338 **Data parallelism.** Data parallelism [49] is a widely used
 339 technique in distributed inference that partitions input data
 340 across multiple devices, such as GPUs, to perform parallel
 341 inference. Each device maintains a complete model replica
 342 and independently processes a subset of the input data (mini-
 343 batch), aggregating results to generate the final output. Data
 344 parallelism enhances throughput by distributing workload
 345 across devices, leveraging their combined computational
 346 power.

347 However, data parallelism's scalability is constrained by
 348 the total batch size [31], which is particularly problematic
 349 in robotic IoT applications where smaller batch sizes are
 350 inherent due to the need for swift environmental responses.
 351 In robotic applications, immediate inference upon receiving
 352 inputs is crucial for obtaining real-time target points quickly.
 353 For example, in our experiments, the robot constantly obtains
 354 the latest images from the camera for inference, with a batch
 355 size of only 1. These small batches cannot be further split
 356 into mini-batches, a fundamental requirement for effective
 357 data parallelism.

358 **Tensor parallelism.** Tensor parallelism [54] is a dis-
 359 tributed inference technique that divides a model's layer
 360 parameters across multiple devices, each storing and com-
 361 puting a portion of the weights. This approach requires an
 362 all-reduce communication step after each layer to combine
 363 results from different devices, introducing significant over-
 364 head, especially for large DNN layers. To mitigate this, TP
 365 is typically deployed across GPUs within the same server in
 366 data centers, using fast intra-server GPU-to-GPU links like
 367 NVLink [20], which is beneficial when the model is too large
 368 for a single device.

369 In contrast to data center networks, the limited bandwidth
 370 in robotic IoT renders the communication cost of TP prohib-
 371 itively high, as evidenced by our evaluation of DINA [30],
 372 a state-of-the-art TP method. Table 1 reveals that transmis-
 373 sion time constitutes 49% to 94% of the total inference time
 374 due to all-reduce communication for each layer, resulting
 375 in TP's inference time being 45.2X to 143.9X longer than
 376 local computation. Although Table 2 indicates lower power
 377 consumption with TP (13.4% to 67.3% less than local computa-
 378 tion), the extended transmission times significantly increase
 379 energy consumption per inference, ranging from 28.5X to
 380 62.7X. Since TP significantly extends inference time, making
 381

382 it impractical for real-world robotic applications, we did not
 383 further consider TP in this paper.

384 **Pipeline parallelism.** Pipeline parallelism [16] is a dis-
 385 tributed inference technique that partitions DNN model
 386 layers across multiple devices(layer partitioning), forming
 387 an inference pipeline for concurrent processing of multi-
 388 ple tasks. While PP can increase throughput and resource
 389 utilization via pipeline execution, it primarily focuses on
 390 enhancing overall throughput rather than reducing single-
 391 inference latency [6], which is crucial in robotic IoT. As
 392 a result, existing distributed inference approaches [5, 23]
 393 in robotic IoT primarily adopt PP paradigm and focus on
 394 layer partitioning to achieve fast and energy-efficient in-
 395 ference, with two main categories based on their optimiza-
 396 tion goals: accelerating inference for diverse DNN struc-
 397 tures [15, 18, 23, 30, 51] and optimizing robot energy con-
 398 sumption during inference [5, 24, 46]. Both two kinds of
 399 methods suffer from the transmission bottleneck inherent
 400 to PP's scheduling mechanism, which can be eliminated by
 401 Intra-DP.

402 2.4 Other methods to speed up DNN Models 403 Inference on Robotic IoT

404 **Compressed communication.** Compressed communica-
 405 tion is crucial for efficient distributed inference in wireless
 406 networks, as it significantly reduces communication over-
 407 head through techniques such as quantization and model
 408 distillation. Quantization [7, 11, 12] is a technique that re-
 409 duces the numerical precision of model weights and ac-
 410 tivations, thereby minimizing the memory footprint and
 411 computational requirements of deep learning models. This
 412 process typically involves converting high-precision (e.g.,
 413 32-bit) floating-point values to lower-precision (e.g., 8-bit)
 414 floating-point representations, with minimal loss of model
 415 accuracy. Model distillation [13, 25, 44], on the other hand,
 416 is an approach that involves training a smaller, more ef-
 417 ficient "student" model to mimic the behavior of a larger,
 418 more accurate "teacher" model by minimizing the differ-
 419 ence between the student model's output and the teacher
 420 model's output. The distilled student model retains much of
 421 the teacher model's accuracy while requiring significantly
 422 fewer resources. These model compression methods com-
 423 plement distributed inference by achieving faster inference
 424 speed through model modifications, potentially sacrificing
 425 some accuracy with smaller models, while distributed in-
 426 ference realizes fast inference without loss of accuracy by
 427 intelligently scheduling computation tasks across multiple
 428 devices.

429 **Inference Job scheduling.** Significant research efforts
 430 have been devoted to exploring inference parallelism and
 431 unleashing the potential of layer partition to accelerate DNN
 432

433 434 435 436 437 438 439 440

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

Table 1. Average transmission time (Second), inference time (Second), percentage that transmission time accounts for of inference time and their standard deviation ($\pm n$) with TP on different models in different environments. “Local computation” refers to placing the whole layers on the robot.

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

Table 2. Power consumption against time (Watt) and energy consumption per inference (Joule) with standard deviation ($\pm n$) with TP on different models in different environments. “Local” represents “Local computation”

inference, such as inference job scheduling, aiming to accelerate multiple DNN inference tasks by optimizing their execution on various devices under different network bandwidths while considering application-specific inference speed requirements and energy consumption demands. For instance, [3, 9] support online scheduling of offloading inference tasks based on the current network and resource status of mobile systems while meeting user-defined energy constraints. [10] focused on optimizing DNN inference workloads in cloud computing using a deep reinforcement learning based scheduler for QoS-aware scheduling of heterogeneous servers, aiming to maximize inference accuracy and minimize response delay. While these methods focus on overall optimization in multi-task scenarios involving multi-robots, they do not address the optimization of single inference tasks and are thus orthogonal to distributed inference for a single inference, where improved distributed inference can provide faster and more energy-efficient inference for these scenarios.

3 Overview

3.1 Existing distributed inference on robotic IoT

However, the increasing complexity of DNN structures, now evolved into directed acyclic graphs (DAGs), poses new challenges, potentially leading to NP-hardness in performance optimization [23]. This issue is addressed by graph theory techniques [23, 51] and varying hardware and network conditions further complicate the problem.

In summary, these two categories primarily adopt the PP paradigm but suffer from the transmission bottleneck

inherent to PP’s scheduling mechanism (see Sec. 3.2). Consequently, achieving fast and energy-efficient inference on robotic IoT remains an open issue.

3.2 Dilemma on Inference Time and Energy Consumption

Regardless of the complexity of DNN models, layer partitioning methods consist of three phases: computing earlier DNN layers on robots, transmitting intermediate results, and completing inference on the GPU device. Since the GPU device’s computation time is negligible compared to the other two phases (see Fig. 1) due to the high computing capabilities of GPU devices, this paper focuses on the computation phase of robots and the data transmission phase via robotic IoT.

The data transmission phase can only begin after obtaining the calculation result of the intermediate layer when the computation phase on robots is completed, preventing overlap for a single inference task. PP can only overlap computation and data transmission phases from different inference tasks, not from the same task [6]. However, the transmission cost inherent to the PP’s scheduling mechanism becomes a bottleneck in robotic IoT due to limited bandwidth. In our experiments, even with optimal layer partitioning [5, 23], such communication cost takes up to 63% of inference time.

To make matters worse, such transmission overhead not only leads to prolonged inference time but also to high energy consumption during the data transmission phase, referred to as transmission energy consumption. Our findings reveal that such transmission energy consumption accounts for nearly one-third of the total energy consumed during inference (see Sec. ??). This is because the device cannot be put into low-power sleep mode while waiting for the final inference result from the GPU device, as it has to promptly continue working when it receives the inference results. Moreover, chips like CPU, GPU, and memory consume non-negligible power even when not computing, due to the static power consumption rooted in transistors' leakage current [19]. Consequently, both the energy consumed during the execution of DNN layers on robots, referred to as robot computation energy consumption, and the transmission energy consumption resulting from prolonged transmission times substantially impact the overall power consumption of the inference process in robotic IoT.

Only models with limited transmission overhead can mitigate the impact of these shortcomings on inference performance. However, the unstable bandwidth in robotic IoT wireless networks can cause the transmission time for layer partitioning to vary dramatically, sometimes changing by hundreds of times (see Fig. 2). In our experiments, even a relatively small model with only 0.84M parameters still suffers from its significant transmission overhead. The significant impact of transmission overhead on both inference time and energy consumption highlights the need for innovative approaches that can effectively mitigate the transmission bottleneck in robotic IoT.

3.3 Special Cases

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

In our experiments, we found that the bandwidth conditions under which the layer partitioning scheme of different models becomes these special cases vary, and the higher

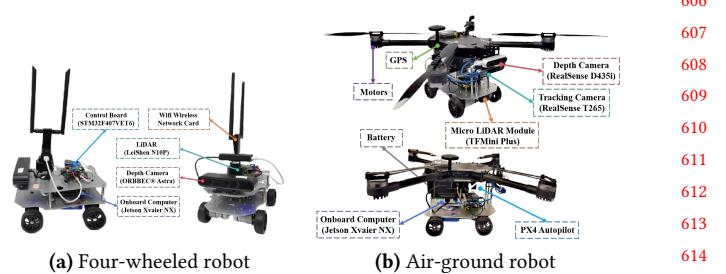


Figure 3. The detailed composition of the robot platforms

the bandwidth, the more layers are scheduled to be placed on GPU devices. We explain the reasons causing different bandwidth conditions for different models in Sec. ?? with some detailed real-world cases. The existence of these special cases highlights the importance of considering the relationship between bandwidth, model structure, and the resulting trade-offs between inference delay and energy consumption.

4 Detailed Design

5 Implementation

6 Evaluation

Testbed. The evaluation was conducted on a custom four-wheeled robot (Fig 3a), and a custom air-ground robot(Fig 3b). They are equipped with a Jetson Xavier NX [33] 8G onboard computer that is capable of AI model inference with local computation resources. The system runs Ubuntu 20.04 with ROS Noetic and a dual-band USB network card (MediaTek MT76x2U) for wireless connectivity. The Jetson Xavier NX interfaces with a Leishen N10P LiDAR, ORBBEC Astra depth camera, and an STM32F407VET6 controller via USB serial ports. Both LiDAR and depth cameras facilitate environmental perception, enabling autonomous navigation, obstacle avoidance, and SLAM mapping. The GPU server accepting offloaded computation tasks from the robot is a PC equipped with an Intel(R) i5 12400f CPU @ 4.40GHz and an NVIDIA GeForce GTX 2080 Ti 11GB GPU, connected to our robot via Wi-Fi 6 over 80MHz channel at 5GHz frequency in our experiments.

Tab. 3 presents the overall on-board energy consumption (excluding motor energy consumption for robot movement) of the robot in various states: inference (model inference with full GPU utilization, including CPU and GPU energy consumption), transmission (communication with the GPU server, including wireless network card energy consumption), and standby (robot has no tasks to execute). Notice that different models, due to varying numbers of parameters, exhibit distinct GPU utilization rates and power consumption during inference.

Experiment Environments. We evaluated two real-world environments: indoors (robots move in our laboratory with

	inference	transmission	standby
Power (W)	13.35	4.25	4.04

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

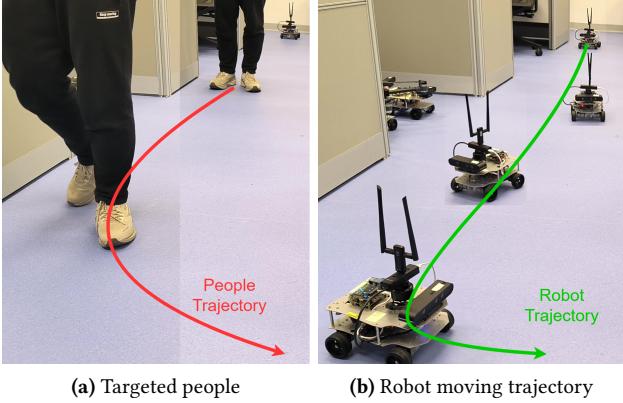


Figure 4. A real-time people-tracking robotic application on our robot based on a well-known human pose estimation ML model, Kapao [29].

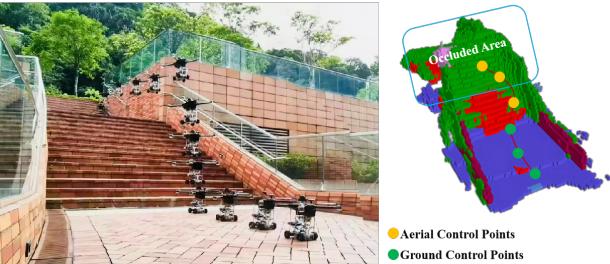


Figure 5. By predicting occlusions in advance, AGRNav [43] gains an accurate perception of the environment and avoids collisions, resulting in efficient and energy-saving paths.

desks and separators interfering with wireless signals) and outdoors (robots move in our campus garden with trees and bushes interfering with wireless signals, resulting in lower bandwidth). The corresponding bandwidths between the robot and the GPU server in indoors and outdoors scenarios are shown in Fig. 2.

Workload. We evaluated two typical real-world robotic applications on our testbed: Kapao, a real-time people-tracking application on our four-wheeled robot (Fig 4), and AGRNav, an autonomous navigation application on our air-ground robot (Fig 5). These applications feature different model input and output size patterns: Kapao takes RGB images as input and outputs key points of small data volume. In contrast, AGRNav takes point clouds as input and outputs predicted point clouds and semantics of similar data volume as input, implying that AGRNav needs to transmit more data during offloading. And we have verified several models common to mobile devices on a larger scale to further corroborate

our observations and findings: MobileNet [40], ResNet [41], VGGNet [39], ConvNeXt [45], RegNet [50].

Baselines. We selected two SOTA pipeline parallelism methods as baselines: DSCCS [23], aimed at accelerating inference, and SPSO-GA [5], focused on optimizing energy consumption. We set SPSO-GA’s deadline constraints to 1 Hz, the minimum frequency required for robot movement control. Given our primary focus on inference time and energy consumption per inference, we disabled pipeline execution to concentrate solely on assessing the performance of various layer partitioning methods.

6.1 Inference Time

Kapao. From the results in the upper part of Tab. 4, both SPSO-GA and DSCCS reduced Kapao’s inference time by 39.69% and 56.92% indoors and 28.67% and 47.46% outdoors, with DSCCS achieving 28.57% (indoors) and 26.34% (outdoors) lower inference time than SPSO-GA. While both systems significantly reduced inference time via offloading, transmission time accounts for 49.69% to 69.46% of the whole inference time, indicating that even with SOTA layer partitioning, the transmission bottleneck inherent to PP’s scheduling mechanism cannot be mitigated. The difference between DSCCS and SPSO-GA can be attributed to their optimization goals: DSCCS minimizes inference latency, while SPSO-GA minimizes power consumption under deadline constraints.

AGRNav. The performance gain of the two offloading systems varied for AGRNav, as shown in the lower part of Tab. 4. DSCCS still reduced inference time by 18.34% and 12.43% in indoors and outdoors. However, SPSO-GA achieved similar inference time (3.65% and 3.06% reduction) as local computation both indoors and outdoors. We will explain and analyze this phenomenon in Sec.6.2.

Notice that the large standard deviation in transmission time in outdoors in both offloading systems indicates that bandwidth fluctuated more frequently and more fiercely outdoors compared with indoors, which complies with Fig. 2. Additionally, the lower average bandwidth for outdoors scenarios (see Sec.2.1) results in increased transmission and inference times relative to indoor scenarios.

6.2 Breakdown

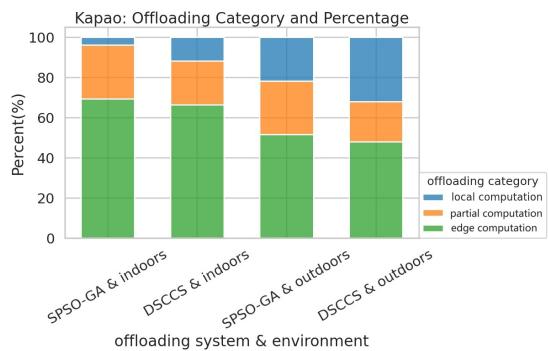
Both SPSO-GA and DSCCS automatically adapt to available bandwidth, transitioning to edge computation (placing all DNN layers on the GPU server) when bandwidth is sufficient, and to local computation (placing all DNN layers on robots) when bandwidth is low. To better understand how their layer partitioning scheduling varies with different network conditions and models, we recorded and analyzed the Categories and percentages of various layer partitioning schedules under different baselines and environments, as detailed in Fig. 6.

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

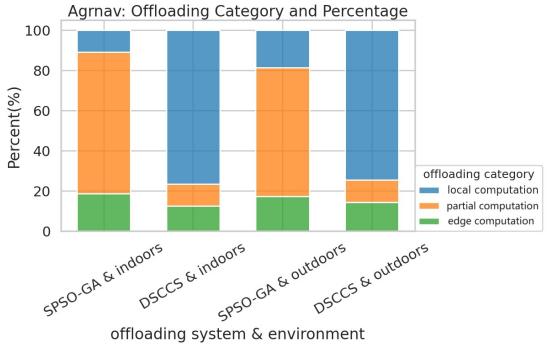
Table 4. Average transmission time (Second), inference time (Second), percentage that transmission time accounts for of inference time and their standard deviation ($\pm n$) of Kapao and AGRNav with different pipeline parallelism offloading systems and different environments. “Local computation” refers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

Model(number of parameters)	Environment	Power consumption(W)			Energy consumption(J) per inference		
		Local	DSCCS	SPSO-GA	Intra-DP	Local	DSCCS
kapao(77M)	indoors	15.04(± 0.64)	7.03(± 3.57)	5.92(± 2.18)	5.03(± 1.87)	15.03(± 0.63)	2.41(± 1.35)
	outdoors	15.04(± 0.64)	14.15(± 1.71)	5.89(± 2.3)	5.02(± 1.97)	1.56(± 0.85)	9.85(± 1.77)
agrnav(0.84M)	indoors	10.26(± 1.58)	6.6(± 1.95)	6.74(± 2.03)	5.74(± 1.78)	10.82(± 1.44)	2.86(± 0.88)
	outdoors	10.26(± 1.58)	7.36(± 2.34)	7.91(± 2.45)	6.71(± 2.1)	10.82(± 1.44)	3.77(± 5.74)

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



(a) Categories and percentage of various scheduling for Kapao



(b) Categories and percentage of various scheduling for AGRNav

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

Local computation and edge computation are special cases of layer partitioning, with the bandwidth conditions required for each model to reach these cases varying based on the model structure and partitioning method used. Analyzing Fig. 6a and Fig. 6b, both SPSO-GA and DSCCS tend to allocate more layers on the robot for AGRNav. When comparing indoor and outdoor scenarios in Fig. 6, it is evident that higher bandwidth leads to more layers being scheduled on GPU server. Additionally, when comparing SPSO-GA and DSCCS in Fig. 6, DSCCS, which focuses on optimizing energy consumption, tends to place fewer layers on the robot to reduce computation energy consumption.

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

6.3 Energy Consumption

Kapao. From the results in the upper part of Tab. 5, DSCCS consumed 3.38% and 2.02% more power per second than SPSO-GA indoors and outdoors due to more layers placed on robots shown in Fig. 6a. However, SPSO-GA consumed 58.54% and 49.74% more energy overall to process a frame than DSCCS because it only aims at minimizing the power consumption against time at the cost of possibly prolonged inference time.

AGRNav. From the results in the lower part of Tab. 5, DSCCS consumed 61.21% and 22.92% more energy per second than SPSO-GA indoors and outdoors (Tab. 5), while DSCCS consumed 34.15% and 5.43% more energy to process a frame than SPSO-GA. SPSO-GA’s advantages in power

Model(number of parameters)	Local computation time (s)	Environment	Transmission time (s)		Inference time (s)		Percentage(%)	
			SPSO-GA	DSCCS	SPSO-GA	DSCCS	SPSO-GA DSCCS	
MobileNet_V3_Small (2M)	0.033(±0.019)	indoors	0.035(±0.019)	0.016(±0.005)	0.044(±0.020)	0.031(±0.008)	79.79	53.24
		outdoors	0.035(±0.044)	0.017(±0.005)	0.047(±0.037)	0.033(±0.018)	50.04	51.49
RegNet_X_3_2GF (15M)	0.060(±0.022)	indoors	0.049(±0.026)	0.033(±0.011)	0.065(±0.028)	0.049(±0.016)	76.25	64.17
		outdoors	0.049(±0.055)	0.032(±0.032)	0.069(±0.050)	0.051(±0.030)	53.23	44.50
ResNet101 (44M)	0.060(±0.023)	indoors	0.054(±0.451)	0.033(±0.010)	0.072(±0.453)	0.050(±0.016)	75.64	57.37
		outdoors	0.052(±0.064)	0.033(±0.036)	0.077(±0.059)	0.054(±0.034)	51.54	42.48
ConvNeXt_Base (88M)	0.047(±0.004)	indoors	0.033(±0.018)	0.020(±0.006)	0.044(±0.019)	0.032(±0.009)	75.39	49.37
		outdoors	0.032(±0.038)	0.020(±0.022)	0.045(±0.033)	0.034(±0.019)	52.82	35.63
ConvNeXt_Large (197M)	0.051(±0.005)	indoors	0.033(±0.017)	0.023(±0.008)	0.046(±0.019)	0.035(±0.013)	72.96	62.68
		outdoors	0.032(±0.038)	0.023(±0.024)	0.054(±0.040)	0.041(±0.028)	48.94	43.96
RegNet_Y_128GF (644M)	0.139(±0.016)	indoors	0.076(±0.289)	0.041(±0.024)	0.305(±0.382)	0.100(±0.035)	23.58	40.76
		outdoors	0.171(±0.602)	0.016(±0.055)	0.432(±0.615)	0.117(±0.242)	32.39	9.41

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

Model(number of parameters)	Environment	Power consumption(W)			Energy consumption(J) per inference		
		Local	SPSO-GA	DSCCS	Local	SPSO-GA	DSCCS
MobileNet_V3_Small (2M)	indoors	6.131(±0.061)	5.448(±0.168)	5.658(±0.085)	0.202(±0.002)	0.241(±0.107)	0.174(±0.046)
	outdoors	6.131(±0.061)	5.567(±0.273)	5.557(±0.186)	0.202(±0.002)	0.260(±0.204)	0.185(±0.099)
RegNet_X_3_2GF (15M)	indoors	8.208(±0.140)	5.490(±0.178)	5.714(±0.342)	0.492(±0.008)	0.356(±0.156)	0.278(±0.058)
	outdoors	8.208(±0.140)	5.878(±0.659)	6.041(±0.624)	0.492(±0.008)	0.406(±0.295)	0.311(±0.184)
ResNet101 (44M)	indoors	11.851(±0.404)	5.457(±0.240)	5.953(±0.789)	0.711(±0.024)	0.390(±2.471)	0.298(±0.094)
	outdoors	11.851(±0.404)	6.179(±1.083)	6.431(±1.060)	0.711(±0.024)	0.478(±0.364)	0.349(±0.216)
ConvNeXt_Base (88M)	indoors	15.335(±0.273)	5.507(±0.358)	7.713(±2.613)	0.721(±0.013)	0.241(±0.103)	0.250(±0.069)
	outdoors	15.335(±0.273)	7.638(±3.297)	9.148(±3.338)	0.721(±0.013)	0.346(±0.254)	0.307(±0.171)
ConvNeXt_Large (197M)	indoors	15.403(±0.082)	5.518(±0.638)	6.604(±2.860)	0.786(±0.004)	0.251(±0.104)	0.230(±0.088)
	outdoors	15.403(±0.082)	8.400(±4.345)	8.895(±4.505)	0.786(±0.004)	0.452(±0.339)	0.366(±0.248)
RegNet_Y_128GF (644M)	indoors	15.430(±0.020)	5.384(±1.071)	6.151(±2.155)	2.145(±0.003)	1.642(±0.327)	0.615(±0.216)
	outdoors	15.430(±0.020)	6.361(±2.349)	9.127(±4.724)	2.145(±0.003)	2.748(±1.015)	1.068(±0.553)

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

consumption against time shrinks in energy consumption per inference due to prolonged inference time.

6.4 Validation on a larger range of models

We evaluated PP across a broad range of models with varying parameter counts (from 0.84M to 644M, as detailed in Tab. 6 and Tab. 7), which are commonly used in mobile devices. Our findings confirm that transmission time constitutes a significant portion of the total inference time in robotic IoT when using PP. The inherent transmission overhead of PP’s scheduling mechanism significantly wastes both inference time and energy.

7 Conclusion

In this paper, we explored the problems that hinder the application of existing parallel methods for distributed inference on robotic IoT, including the failure of data parallelism due

to small batch sizes, the unacceptable communication overhead of tensor parallelism caused by all-reduce communication, and the significant transmission bottlenecks inherent to pipeline parallelism’s scheduling mechanism. By raising awareness of these issues, we aim to stimulate research efforts towards developing novel parallel methods that address these problems. We envision that fast and energy-efficient inference will foster the deployment of diverse robotic tasks on real-world robots in the field.

References

- [1] [n. d.] iPerf - Download iPerf3 and original iPerf pre-compiled binaries. <https://iperf.fr/iperf-download.php>
- [2] Toni Adame, Marc Carrascosa-Zamacois, and Boris Bellalta. 2021. Time-sensitive networking in IEEE 802.11 be: On the way to low-latency WiFi 7. *Sensors* 21, 15 (2021), 4954.
- [3] Majid Altamimi, Atef Abdrabou, Kshirasagar Naik, and Amiya Nayak. 2015. Energy cost models of smartphones for task offloading to the cloud. *IEEE Transactions on Emerging Topics in Computing* 3, 3 (2015), 990

- 991 384–398.
- 992 [4] Anh-Quan Cao and Raoul de Charette. 2022. Monoscene: Monocular 3d semantic scene completion. In *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition*. 3991–4001.
- 993 [5] Xing Chen, Jianshan Zhang, Bing Lin, Zheyi Chen, Katinka Wolter, and Geyong Min. 2021. Energy-efficient offloading for DNN-based smart IoT systems in cloud-edge environments. *IEEE Transactions on Parallel and Distributed Systems* 33, 3 (2021), 683–697.
- 994 [6] Daniel Crankshaw, Gur-Eyal Sela, Xiangxi Mo, Corey Zumar, Ion Stoica, Joseph Gonzalez, and Alexey Tumanov. 2020. InferLine: latency-aware provisioning and scaling for prediction serving pipelines. In *Proceedings of the 11th ACM Symposium on Cloud Computing*. 477–491.
- 995 [7] Alexandre Défossez, Yossi Adi, and Gabriel Synnaeve. 2021. Differentiable model compression via pseudo quantization noise. *arXiv preprint arXiv:2104.09987* (2021).
- 996 [8] Ming Ding, Peng Wang, David López-Pérez, Guoqiang Mao, and Zihua Lin. 2015. Performance impact of LoS and NLoS transmissions in dense cellular networks. *IEEE Transactions on Wireless Communications* 15, 3 (2015), 2365–2380.
- 997 [9] Khalid Elgazzar, Patrick Martin, and Hossam S Hassanein. 2014. Cloud-assisted computation offloading to support mobile services. *IEEE Transactions on Cloud Computing* 4, 3 (2014), 279–292.
- 998 [10] Zhou Fang, Tong Yu, Ole J Mengshoel, and Rajesh K Gupta. 2017. Qos-aware scheduling of heterogeneous servers for inference in deep neural networks. In *Proceedings of the 2017 ACM on Conference on Information and Knowledge Management*. 2067–2070.
- 999 [11] Stefan Gheorghe and Mihai Ivanovici. 2021. Model-based weight quantization for convolutional neural network compression. In *2021 16th International Conference on Engineering of Modern Electric Systems (EMES)*. IEEE, 1–4.
- 1000 [12] Cheng Gong, Yao Chen, Ye Lu, Tao Li, Cong Hao, and Deming Chen. 2020. VecQ: Minimal loss DNN model compression with vectorized weight quantization. *IEEE Trans. Comput.* 70, 5 (2020), 696–710.
- 1001 [13] Jianping Gou, Baosheng Yu, Stephen J Maybank, and Dacheng Tao. 2021. Knowledge distillation: A survey. *International Journal of Computer Vision* 129, 6 (2021), 1789–1819.
- 1002 [14] Vinayak Honkote, Dileep Kurian, Sriram Muthukumar, Dibyendu Ghosh, Satish Yada, Kartik Jain, Bradley Jackson, Ilya Klotchkov, Mallikarjuna Rao Nimmagadda, Shreela Dattawadkar, et al. 2019. 2.4 a distributed autonomous and collaborative multi-robot system featuring a low-power robot soc in 22nm cmos for integrated battery-powered minibots. In *2019 IEEE International Solid-State Circuits Conference-(ISSCC)*. IEEE, 48–50.
- 1003 [15] Chuang Hu, Wei Bao, Dan Wang, and Fengming Liu. 2019. Dynamic adaptive DNN surgery for inference acceleration on the edge. In *IEEE INFOCOM 2019-IEEE Conference on Computer Communications*. IEEE, 1423–1431.
- 1004 [16] Yang Hu, Connor Imes, Xuanang Zhao, Souvik Kundu, Peter A Beerel, Stephen P Crago, and John Paul Walters. 2022. Pipeedge: Pipeline parallelism for large-scale model inference on heterogeneous edge devices. In *2022 25th Euromicro Conference on Digital System Design (DSD)*. IEEE, 298–307.
- 1005 [17] KJJoseph, Salman Khan, Fahad Shahbaz Khan, and Vineeth N Balasubramanian. 2021. Towards Open World Object Detection. In *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition (CVPR)*. 5830–5840.
- 1006 [18] Yiping Kang, Johann Hauswald, Cao Gao, Austin Rovinski, Trevor Mudge, Jason Mars, and Lingjia Tang. 2017. Neurosurgeon: Collaborative intelligence between the cloud and mobile edge. *ACM SIGARCH Computer Architecture News* 45, 1 (2017), 615–629.
- 1007 [19] Nam Sung Kim, Todd Austin, David Baauw, Trevor Mudge, Krisztián Flautner, Jie S Hu, Mary Jane Irwin, Mahmut Kandemir, and Vijaykrishnan Narayanan. 2003. Leakage current: Moore’s law meets static power. *computer* 36, 12 (2003), 68–75.
- 1008 [20] Ang Li, Shuaiwen Leon Song, Jieyang Chen, Jiajia Li, Xu Liu, Nathan R Tallent, and Kevin J Barker. 2019. Evaluating modern gpu interconnect: Pcie, nvlink, nv-sli, nvswitch and gpudirect. *IEEE Transactions on Parallel and Distributed Systems* 31, 1 (2019), 94–110.
- 1009 [21] Qingbiao Li, Fernando Gama, Alejandro Ribeiro, and Amanda Prorok. 2020. Graph neural networks for decentralized multi-robot path planning. In *2020 IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS)*. IEEE, 11785–11792.
- 1010 [22] Yiming Li, Zhiding Yu, Christopher Choy, Chaowei Xiao, Jose M Alvarez, Sanja Fidler, Chen Feng, and Anima Anandkumar. 2023. Voxformer: Sparse voxel transformer for camera-based 3d semantic scene completion. In *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition*. 9087–9098.
- 1011 [23] Huanghuang Liang, Qianlong Sang, Chuang Hu, Dazhao Cheng, Xiaobo Zhou, Dan Wang, Wei Bao, and Yu Wang. 2023. DNN surgery: Accelerating DNN inference on the edge through layer partitioning. *IEEE transactions on Cloud Computing* (2023).
- 1012 [24] Bing Lin, Yinhao Huang, Jianshan Zhang, Junqin Hu, Xing Chen, and Jun Li. 2019. Cost-driven off-loading for DNN-based applications over cloud, edge, and end devices. *IEEE Transactions on Industrial Informatics* 16, 8 (2019), 5456–5466.
- 1013 [25] Tao Lin, Lingjing Kong, Sebastian U Stich, and Martin Jaggi. 2020. Ensemble distillation for robust model fusion in federated learning. *Advances in Neural Information Processing Systems* 33 (2020), 2351–2363.
- 1014 [26] Ruofeng Liu and Nakjung Choi. 2023. A First Look at Wi-Fi 6 in Action: Throughput, Latency, Energy Efficiency, and Security. *Proceedings of the ACM on Measurement and Analysis of Computing Systems* 7, 1 (2023), 1–25.
- 1015 [27] Shuai Liu, Xin Li, Huchuan Lu, and You He. 2022. Multi-Object Tracking Meets Moving UAV. In *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition (CVPR)*. 8876–8885.
- 1016 [28] Antoni Masiukiewicz. 2019. Throughput comparison between the new HEW 802.11 ax standard and 802.11 n/ac standards in selected distance windows. *International Journal of Electronics and Telecommunications* 65, 1 (2019), 79–84.
- 1017 [29] William McNally, Kanav Vats, Alexander Wong, and John McPhee. 2022. Rethinking keypoint representations: Modeling keypoints and poses as objects for multi-person human pose estimation. In *European Conference on Computer Vision*. Springer, 37–54.
- 1018 [30] Thaha Mohammed, Carlee Joe-Wong, Rohit Babbar, and Mario Di Francesco. 2020. Distributed inference acceleration with adaptive DNN partitioning and offloading. In *IEEE INFOCOM 2020-IEEE Conference on Computer Communications*. IEEE, 854–863.
- 1019 [31] Deepak Narayanan, Mohammad Shoeybi, Jared Casper, Patrick LeGresley, Mostofa Patwary, Vijay Korthikanti, Dmitri Vainbrand, Prethvi Kashinkunti, Julie Bernauer, Bryan Catanzaro, et al. 2021. Efficient large-scale language model training on gpu clusters using megatron-lm. In *Proceedings of the International Conference for High Performance Computing, Networking, Storage and Analysis*. 1–15.
- 1020 [32] Mohammad Noormohammadi and Cauligi S Raghavendra. 2017. Datacenter traffic control: Understanding techniques and tradeoffs. *IEEE Communications Surveys & Tutorials* 20, 2 (2017), 1492–1525.
- 1021 [33] NVIDIA. 2024. The World’s Smallest AI Supercomputer. <https://www.nvidia.com/en-us/autonomous-machines/embedded-systems/jetson-xavier-series/>.
- 1022 [34] Takeshi Ohkawa, Kazushi Yamashina, Hitomi Kimura, Kanemitsu Ootsu, and Takashi Yokota. 2018. FPGA components for integrating FPGAs into robot systems. *IEICE TRANSACTIONS on Information and Systems* 101, 2 (2018), 363–375.
- 1023 [35] Yuanpeng Pei, Matt W Mutka, and Ning Xi. 2013. Connectivity and bandwidth-aware real-time exploration in mobile robot networks. *Wireless Communications and Mobile Computing* 13, 9 (2013), 847–863.

1046
1047
1048
1049
1050
1051
1052
1053
1054
1055
1056
1057
1058
1059
1060
1061
1062
1063
1064
1065
1066
1067
1068
1069
1070
1071
1072
1073
1074
1075
1076
1077
1078
1079
1080
1081
1082
1083
1084
1085
1086
1087
1088
1089
1090
1091
1092
1093
1094
1095
1096
1097
1098
1099
1100

- 1101 [36] pytorch. 2024. pytorch. <https://pytorch.org/>.
 1102 [37] Yi Ren, Chih-Wei Tung, Jyh-Cheng Chen, and Frank Y Li. 2018. Proportional and preemption-enabled traffic offloading for IP flow mobility: Algorithms and performance evaluation. *IEEE Transactions on Vehicular Technology* 67, 12 (2018), 12095–12108.
 1103 [38] Nurul I Sarkar and Osman Mussa. 2013. The effect of people movement
 1104 on Wi-Fi link throughput in indoor propagation environments. In *IEEE
 1105 2013 Tencon-Spring*. IEEE, 562–566.
 1106 [39] Karen Simonyan and Andrew Zisserman. 2015. Very Deep
 1107 Convolutional Networks for Large-Scale Image Recognition.
 1108 arXiv:1409.1556 [cs.CV]
 1109 [40] Debjyoti Sinha and Mohamed El-Sharkawy. 2019. Thin mobilenet: An
 1110 enhanced mobilenet architecture. In *2019 IEEE 10th annual ubiquitous
 1111 computing, electronics & mobile communication conference (UEMCON)*.
 1112 IEEE, 0280–0285.
 1113 [41] Sasha Targ, Diogo Almeida, and Kevin Lyman. 2016. Resnet in resnet:
 1114 Generalizing residual architectures. *arXiv preprint arXiv:1603.08029*
 1115 (2016).
 1116 [42] Hao Wang, Sreeram Potluri, Miao Luo, Ashish Kumar Singh, Sayantan
 1117 Sur, and Dhabaleswar K Panda. 2011. MVAPICH2-GPU: optimized
 1118 GPU to GPU communication for InfiniBand clusters. *Computer Science-
 1119 Research and Development* 26, 3 (2011), 257–266.
 1120 [43] Junming Wang, Zekai Sun, Xiuxian Guan, Tianxiang Shen, Zongyuan
 1121 Zhang, Tianyang Duan, Dong Huang, Shixiong Zhao, and Heming Cui.
 1122 2024. AGRNav: Efficient and Energy-Saving Autonomous Navigation
 1123 for Air-Ground Robots in Occlusion-Prone Environments. In *IEEE
 1124 International Conference on Robotics and Automation (ICRA)*.
 1125 [44] Lin Wang and Kuk-Jin Yoon. 2021. Knowledge distillation and student-
 1126 teacher learning for visual intelligence: A review and new outlooks.
 1127 *IEEE transactions on pattern analysis and machine intelligence* 44, 6
 1128 (2021), 3048–3068.
 1129 [45] Sanghyun Woo, Shoubhik Debnath, Ronghang Hu, Xinlei Chen,
 1130 Zhuang Liu, In So Kweon, and Saining Xie. 2023. Convnext v2: Co-
 1131 designing and scaling convnets with masked autoencoders. In *Pro-
 1132 ceedings of the IEEE/CVF Conference on Computer Vision and Pattern
 1133 Recognition*. 16133–16142.
 1134 [46] Huaming Wu, William J Knottenbelt, and Katinka Wolter. 2019. An
 1135 efficient application partitioning algorithm in mobile environments.
 1136 *IEEE Transactions on Parallel and Distributed Systems* 30, 7 (2019),
 1137 1464–1480.
 1138 [47] Tianyu Wu, Shizhu He, Jingping Liu, Siqi Sun, Kang Liu, Qing-Long
 1139 Han, and Yang Tang. 2023. A brief overview of ChatGPT: The history,
 1140 status quo and potential future development. *IEEE/CAA Journal of
 1141 Automatica Sinica* 10, 5 (2023), 1122–1136.
 1142 [48] Zhaoyang Xia, Youquan Liu, Xin Li, Xinge Zhu, Yuexin Ma, Yikang Li,
 1143 Yuenan Hou, and Yu Qiao. 2023. SCPNet: Semantic Scene Completion
 1144 on Point Cloud. In *Proceedings of the IEEE/CVF Conference on Computer
 1145 Vision and Pattern Recognition*. 17642–17651.
 1146 [49] Yecheng Xiang and Hyoseung Kim. 2019. Pipelined data-parallel
 1147 CPU/GPU scheduling for multi-DNN real-time inference. In *2019 IEEE
 1148 Real-Time Systems Symposium (RTSS)*. IEEE, 392–405.
 1149 [50] Jing Xu, Yu Pan, Xinglin Pan, Steven Hoi, Zhang Yi, and Zenglin Xu.
 1150 2022. RegNet: self-regulated network for image classification. *IEEE
 1151 Transactions on Neural Networks and Learning Systems* (2022).
 1152 [51] Min Xue, Huaming Wu, Guang Peng, and Katinka Wolter. 2021.
 1153 DDPQN: An efficient DNN offloading strategy in local-edge-cloud
 1154 collaborative environments. *IEEE Transactions on Services Computing*
 1155 15, 2 (2021), 640–655.
 1156 [52] Xinlei Yang, Hao Lin, Zhenhua Li, Feng Qian, Xingyao Li, Zhiming He,
 1157 Xudong Wu, Xianlong Wang, Yunhao Liu, Zhi Liao, et al. 2022. Mobile
 1158 access bandwidth in practice: Measurement, analysis, and implications.
 1159 In *Proceedings of the ACM SIGCOMM 2022 Conference*. 114–128.
 1160 [53] Yang Yang, Li Juntao, and Peng Lingling. 2020. Multi-robot path
 1161 planning based on a deep reinforcement learning DQN algorithm.
 1162