New Problems in Distributed Inference for DNN Models on Robotic IoT

(Anonymous Authors)

***Abstract*—The rapid advancements in machine learning (ML) techniques have led to significant achievements in various robotic tasks. Deploying these ML approaches on real-world robots requires fast and energy-efficient inference of their deep neural network (DNN) models. To our knowledge, distributed inference, which offloads a portion of the inference computation onto powerful GPU devices, has emerged as a promising optimization to improve inference performance in modern data centers. However, when deployed on real-world robots, existing parallel methods can not simultaneously meet the latency and energy requirements of robotic and raise significant challenges.**

This paper reveals and evaluates the problems hindering the application of these parallel methods in robotic IoT, including the in-applicability of data parallelism due to small batch sizes, the unacceptable communication overhead of tensor parallelism caused by frequent synchronization, and the transmission bottlenecks in pipeline parallelism that cannot be effectively mitigated through scheduling. By raising awareness of these new problems, we aim to stimulate research efforts toward finding a new parallel method that can accelerate distributed inference in robotic IoT.

I. Introduction

The rapid progress in machine learning (ML) techniques has led to remarkable achievements in various fundamental robotic tasks, such as object detection[16], [22], [24], robotic control[18], [35], [41], and environmental perception[4], [19], [38]. 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. Equipping additional computing accelerators on robots (e.g., GPU[27], FPGA[28], SoC[13]) increases energy consumption (e.g., 62% for [24] in our experiments) due to the computationally-intensive nature of DNN models while placing the entire model to the cloud brings a long response delay.

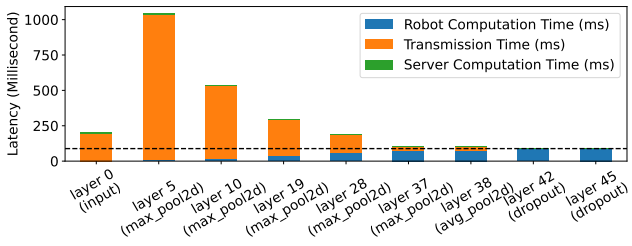
Distributed inference[**?**], which involves offloading a portion of the inference computation onto other powerful GPU devices, has emerged as a promising approach to meet the latency requirements of robotic applications and extend the battery lifetime of robots. By leveraging the high computing capabilities of powerful GPUs through the Internet of Things for these robots (robotic IoT), distributed inference accelerates the inference process while alleviating the local computational burden, thereby reducing energy consumption. This paradigm has been widely adopted in data centers[**?**], where numerous GPUs are utilized for large model inference, such as in the case of ChatGPT[37].

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 is sliced out of a input data set); Tensor parallelism (TP) splits a single DNN layer over devices; Pipeline parallelism (PP) places different layers of a DNN model over devices (layer partitioning) and pipelines the inference to reduce devices' idling time (pipeline execution). In this paper, we demonstrate several issues that impede the application of existing parallel methods in robotic IoT.

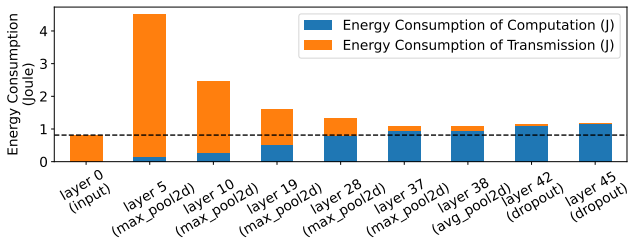
Problem 1 (DP). The small batch sizes inherent to robotic IoT applications (typically 1) hinder the computation of mini-batches, rendering DP inapplicable for robotic IoT. In the data center, DP is feasible due to the large batch sizes employed (e.g., 16 images), allowing for the division of inputs into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

Problem 2 (TP). TP requires frequent synchronization among devices, leading to unacceptable communication overhead in robotic IoT. By partitioninging parameter tensors of a layer across GPUs, TP allows concurrent computation on different parts of this tensor but requires an all-reduce synchronization[42] to combine computation results from different devices, which entails significant communication overhead. Consequently, TP is used mainly for large layers that are too large to fit in one device in data centers and require dedicated high-speed interconnects (e.g., 400 Gbps for NVLink[17]) even within data centers. On the contrary, robots must prioritize seamless mobility and primarily depend on wireless connections, which inherently possess limited bandwidth, as described in Sec.II-A, making all-reduce synchronization an unacceptable overhead (e.g., commonly hundreds of milliseconds for each layer of VGG19[33] in our experiments).

Consequently, existing distributed inference approaches[**?**] in robotic IoT primarily adopt the PP paradigm and focus on layer partitioning of PP, aiming to achieve fast and energy-efficient inference. This is because the PP paradigm in data



(a) Inference Latency



(b) Energy consumption

Fig. 1: Our experiments on VGG19[33] reveal the comprehensive performance of various layer partitioning methods. The X-axis of the graph represents different layer partitioning scheduling schemes, where 'layer i' signifies that all layers up to and including the i-th layer are computed on the robot, while the subsequent layers are processed on the server. Notice that different hardware conditions, network conditions and DNN model structure will lead to different performance, making this field an attractive area for a wide range of research.

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[5], which is the most critical requirement in robotic IoT. Based on the fact that the amounts of output data in some intermediate layers of a DNN model are significantly smaller than that of its raw input data[14], DNN layer partitioning methods constitute various trade-offs between computation and transmission, taking into account application-specific inference speed requirements and energy consumption demands, as shown in Fig.1.

Problem 3 (PP). Existing methods based on PP face significant challenges due to transmission bottlenecks in robotic IoT, which cannot be effectively mitigated through PP scheduling. Firstly, PP is unable to overlap the transmission and computation phases within the same inference, as it can only overlap these phases across multiple inferences via pipeline execution, which increases throughput but not the speed of a single inference[5]. Moreover, due to the limited bandwidth of robotic IoT, the transmission phase can be a substantial bottleneck (e.g., up to 70% of inference time and 50% of energy consumption in our experiments). Secondly, adjusting the scheduling scheme to reduce the traffic volume is not an ideal solution for robotic IoT, as allocating more DNN model layers to robots, while reducing the transmission volume and associated transmission time, increases energy consumption and under-utilizes the powerful GPU devices for accelerating inference, and may not necessarily reduce overall inference time.

In this paper, we take the first step to reveal and evaluate the problems hindering existing parallel methods for distributed inference applying on robotic IoT. These findings aim to raise research effort to find a new parallel method to speed up distributed inference on robotic IoT, so that the DNN models deployed on real-world robots can achieve fast and energy-efficient inference, and it will nurture diverse ML applications deployed on mobile robots in the field.

The rest of the paper is organized as follows: the chapter II provides background; the chapter III describes in detail about the problems; the chapter IV; provides with evaluation results; the chapter V concludes.

II. Background

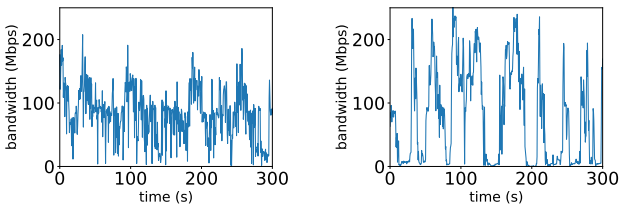
A. Characteristics of Robotic IoT Networks

Wireless transmission of Robotic IoT. In real-world robotic IoT scenarios, devices often need to navigate and move around for tasks such as search and exploration. While wireless networks provide the required high mobility, they also come with 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[21]. Furthermore, not only the limited hardware resources on the robot can not fully play the potential of Wi-Fi 6[40], but also the actual available bandwidth of wireless networks is often reduced in practice due to factors such as movement of the devices[23], [29], occlusion from by physical barriers[7], [32], and preemption of the wireless channel by other devices[2], [31].

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). The hardware and wireless network settings are as described in Sec. IV. We believe our setup represents the state-of-the-art (SOTA) computation and communication capabilities of robotic IoT devices. We saturated the wireless network connection with iperf [1] and recorded the average bandwidth capacity between these robots every 0.1s for 5 minutes.

The results, shown in Fig.2, indicate only 93 Mbps and 73 Mbps average bandwidth capacity for indoor and outdoor scenarios. The outdoor environment exhibited higher instability, with bandwidth frequently dropping to extremely low values around 0 Mbps. This can be attributed to the lack of walls in open outdoor areas to reflect wireless signals, and the presence of obstacles, such as trees, between communicating robots. Consequently, fewer signals can be received in outdoor areas compared to indoor environments.

In summary, the wireless transmission of robotic IoT systems is constrained by limited bandwidth, both due to the



(a) Indoors (b) Outdoors

Fig. 2: The instability of wireless transmission in robotic IoT networks.

theoretical upper limit of wireless transmission technologies and the practical instability of wireless networks.

Comparison with Datacenter Networks. Compared to robotic IoT networks, data center networks (used for large model inference, e.g., ChatGPT[37]) are wired and typically exhibit higher bandwidth capacity and lower bandwidth fluctuation. In data center networks, GPU devices are connected using high-speed networking technologies, such as InfiniBand[34] or PCIe[17], which offer bandwidths ranging from 40 Gbps to 500 Gbps. Bandwidth fluctuation in these networks is primarily caused by congestion on intermediate switches and can be mitigated through traffic scheduling on switches[26].

B. Other methods on robotic IoT

Compressed communication. Compressed communication, which is essential for practical distributed inference over wireless networks, significantly reduces communication traffic volume through techniques such as quantization[6], [10], [11] and model distillation[12], [20], [36]. Quantization 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) fixed-point representations, with minimal loss of model accuracy. Model distillation[12], [20], [36], on the other hand, 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. As they achieve faster inference speed by modifying the model and potentially sacrificing accuracy, these model compression methods are orthogonal to distributed inference and complement distributed inference, which realizes fast inference without loss of accuracy by intelligently scheduling computation tasks across multiple devices.

Job scheduling. Significant research efforts have been devoted to exploring inference parallelism and unleashing the potential of layer partition to accelerate DNN inference, such as job scheduling, which determines the scheduling mode of different inference tasks based on their delay and energy consumption requirements, as well as the load condition of various devices. For instance, [3], [8] support online scheduling of offloading inference tasks based on the current network and resource status of mobile systems while meeting user-defined energy constraints to extend battery life and enhance the capabilities of mobile device. [9] 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 aim at overall optimization in multi-task scenarios involving multi-robots at edge side, 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 them.

III. Problems in Parallel Inference Strategies for DNN Models on Robotic IoT

A. Existing parallel inference strategies in data center

Data parallelism. Data parallelism[39] is a widely adopted technique in distributed inference, which involves partitioning the input data across multiple computational devices, such as GPUs, to perform inference in parallel. In this approach, each device maintains a complete replica of the model and processes a subset of the input data, known as a mini-batch, independently. Once all devices have completed their respective inference tasks, the results are aggregated to generate the final output. Data parallelism proves particularly beneficial in scenarios involving a high volume of inference requests, as it can significantly enhance throughput by distributing the workload across multiple devices, thereby leveraging their combined computational power.

However, the scalability of data parallelism, which refers to its ability to effectively utilize an increasing number of GPU devices, is constrained by the size of the total batch[25]. This limitation becomes even more pronounced in robotic IoT applications, where smaller batch sizes are inherent due to the need for swift environmental responses and can not be further split into smaller mini-batches, which is a fundamental requirement for data parallelism to function effectively. In robotic applications, immediate inference upon receiving inputs is crucial to enable the robot to obtain real-time target points as quickly as possible. For example, in our experiments conducted by [24], the robot constantly obtains the latest images from the camera for inference, with the batch size being only 1.

Tensor parallelism. Tensor parallelism[42] is a distributed inference technique that divides a model's layer parameters across multiple devices, with each device storing and computing a portion of the weights. This approach necessitates an all-reduce communication step after each layer to combine the results from different devices, which introduces significant

overhead, particularly for large DNN layers. To mitigate this overhead, TP is typically deployed across GPUs within the same server in data centers, leveraging fast intra-server GPU-to-GPU links such as NVLink[17]. This setup is particularly beneficial when the model is too large to fit on a single device, as it allows for efficient communication between GPUs.

However, in the context of robotic IoT, the limited bandwidth described in Sec.II-A renders the communication cost of TP prohibitively high. To illustrate this, we conducted experiments on VGG19, officially provided by PyTorch[30], in the indoor scenario of our experiments. The all-reduce communication for each layer takes hundreds of milliseconds, with the maximum layer in VGG19 taking an average of 1.05 seconds. This is more than ten times slower than computing the entire model locally, making TP impractical for robotic IoT environments.

Pipeline parallelism. Pipeline parallelism [15] is a distributed inference technique that partitions layers of a DNN model across multiple devices (layer partitioning), forming an inference pipeline that allows concurrent processing of multiple tasks. While PP can increase throughput and resource utilization, its primary focus is on enhancing overall throughput rather than reducing single-inference latency[5], which is crucial in robotic IoT. Consequently, existing distributed inference approaches[**?**] in robotic IoT mainly concentrate on the layer partitioning aspect of PP, aiming to achieve fast and energy-efficient inference by optimizing the allocation of DNN layers across devices while considering factors such as device capabilities, network bandwidth, and energy consumption, as will be further discussed in Sec.III-B.

B. Existing distributed inference on robotic IoT

[14] designed a dynamic DNN surgery strategy to partition DNN inference between the cloud and edge at the granularity of the DNN layers. This strategy reduced the system latency and improved throughput by limiting data transmission, but it paid less attention to the offloading problem for DNN layers.

Mohammed et al. [19] proposed an adaptive DNN partition scheme and a distributed algorithm based on the matching game method, where the DNN layers were offloaded to fog nodes. Neurosurgeon [4] claimed that excessive latency and energy consumption were generated when uploading massive data of DNNs to the cloud via the wireless network.

To cope with this problem, a lightweight scheduler was designed to partition DNN-based applications automatically between end devices and the cloud at the granularity of DNN layers.

based on these basic layer partitioning methods, there are mainly two kinds of development directions: more complex model structure and energy consumption on robots. for more complex model structure,

for Energy consumption on robots, Most of the aforementioned work tried to reduce the system latency in cloud/edge environments [16], [18], [23], but they did not consider reducing the system energy consumption of offloading DNN layers with deadline constraints.

An application partitioning algorithm was presented in [20] for pursuing a trade-off between energy consumption and data transmission in dynamic mobile environments. Teerapittayanon et al. [21] proposed distributed DNNs over the cloud, edge, and IoT devices. They considered the data transmission cost but not the layer execution consumption, while deploying distributed DNNs in the cloud-edge environments. In the previous work [6], a cost-driven offloading scheme was designed for DNN-based smart IoT systems with deadline constraints over the cloud, edge, and IoT devices, where a discrete PSO algorithm was developed to reduce the system cost of executing DNN layers and transferring data. Different from this work, we further consider the energy consumption of each participating server and IoT device, and introduce the layer partition operations into the offloading decision-making process for DNN layers.

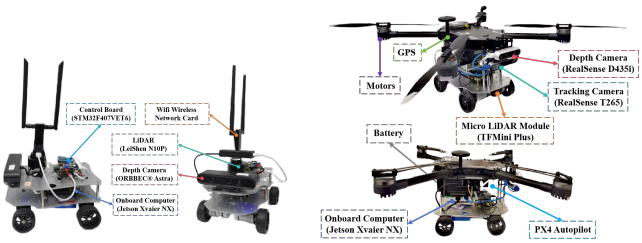
In general, most of these work focused on the layer partitioning problem in distributed inference. However, it is still an open issue to optimize the system energy consumption when offloading DNN layers with deadline constraints in robotic IoT.

C. Dilemma on Inference Time and Energy Consumption

People may think a stalling robot can be consuming little energy. However, we recorded the energy consumption when a robot is stalling due to the straggler effect and found that a stalling robot still consumed almost one third of the energy consumption when the robot was computing (see Sec. VI). That is because the device cannot be put into low power sleep mode even when stalling, as it has to wait for messages from the parameter server and promptly continue working when stragglers catch up, and 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 [32]. Consequently, besides damaging training throughput, stall caused by the straggler effect also has a major impact on the power consumption of the training process.

IV. 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[27] 8G onboard computer serving as the ROS master. The system runs Ubuntu 18.04 and utilizes a SanDisk 256G memory card, with ROS Noetic installed for application development 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 host computer processes environmental information in ROS2 Galactic, performing path planning, navigation, and obstacle avoidance before transmitting velocity and control data to corresponding ROS topics. The controller then subscribes to these topics, executing robot control tasks.



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

Fig. 3: The detailed composition of the robot platforms

	inference	communication	standby
Power (W)	13.35	4.25	4.04

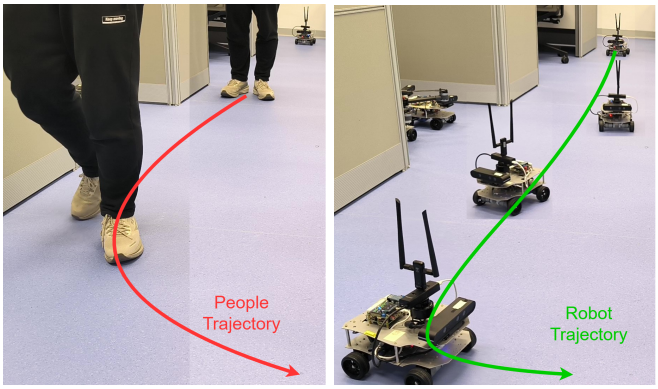
TABLE I: Power (Watt) of our robot in different states.

We documented the overall on-board energy consumption (excluding motor energy consumption for robot movement) of the robot in various states, as presented in Table I. These states include: inference, which refers to model inference with the full utilization of GPU and encompasses the energy consumption of both the CPU and GPU; communication, which involves communication with the server and includes the energy consumption of the wireless network card; and standby, during which the robot has no tasks to execute.

We setup two real-world environments for our evaluation, namely indoors and outdoors. In the indoors scenario, robots move around in our laboratory with desks and separators interfering with wireless signals. In the outdoors scenario, robots move around in our campus garden with trees and bushes interfering with wireless signals and this scenario imposes higher level of instability. The GPU device is a PC equipped with 8xIntel(R) Core(TM) i7-7700K CPU @ 4.20GHz and NVIDIA GeForce GTX 2080 Ti 11GB, connected to our robot via Wi-Fi 6 with an average bandwidth over 80MHz channel at 5GHz frequency in our experiments, and the corresponding bandwidths in indoors and outdoors scenarios are shown in Fig.2.

Real-world Robotic Applications. We evaluated two kinds of typical real-world AI applications on robots: a real-time people-tracking robotic application on our four-wheeled robot as depicted in Fig 6, referred to as Kapao, and an autonomous navigation on our air-ground robot as depicted in Fig 7, referred to as AGRNav. The two applications feature different pattern of model input sizes and output sizes: Kapao takes RGB images as inference input while outputing person keypoints of small data volume; AGRNav takes pointclouds as inference input and outputs predicted pointclouds and semantics of similar data volume as input, which implies AGRNav needs to transmit more data volume during offloading. These can also be inferred from the transmitted & received data volume Tab. II and Tab. III.

Baselines. designed a dynamic DNN surgery strategy to partition DNN inference between the GPU devices and robots at the granularity of the DNN layers.



(a) Targeted people (b) Robot moving trajectory

Fig. 4: A real-time people-tracking robotic application on our robot based on a well-known human pose estimation ML model, KAPAO[24].

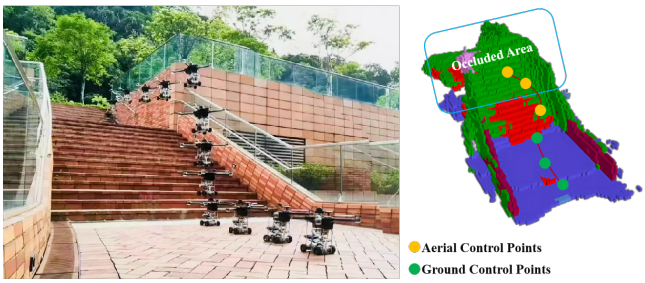
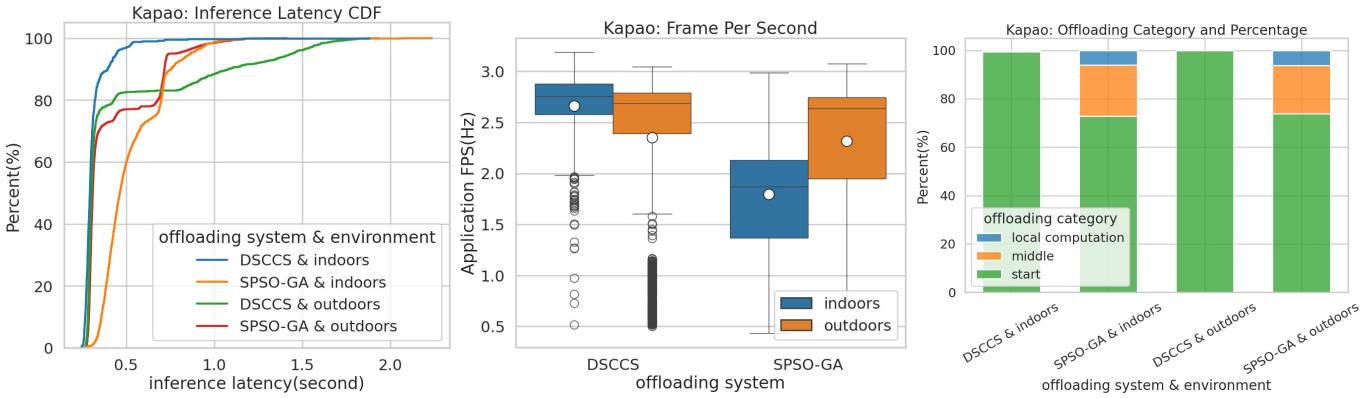


Fig. 5: By predicting occlusions in advance, AGRNav[35] gains an accurate perception of the environment and avoids collisions, resulting in efficient and energy-saving paths.

A. *Inference Time*

Tab. II records the average transmitted & received data volume when offloading computation to the server and the overall average transmission time, inference time and their

While DSCCS aims at minimizing inference latency, and SPSO-GA aims at minimizing power consumption by offloading more computation within an inference latency constraint, the two offloading systems both tend to offload all computation to the server when network bandwidth is high; and at low network bandwidth DSCCS tends to local compute but SPSO-GA still tries to offload computation to the server at the cost of inference latency. These accounts for high ratio of offloading category of start (offloading whole inference input) in indoors of both systems, which is reduced in outdoors and DSCCS has a higher ratio of local computation than SPSO-GA in both environments depicted in Fig. 6c. As a result, consistent higher inference latency and lower application FPS of SPSO-GA than DSCCS in both environments can be observed in Fig. 6a and Fig. 6b.

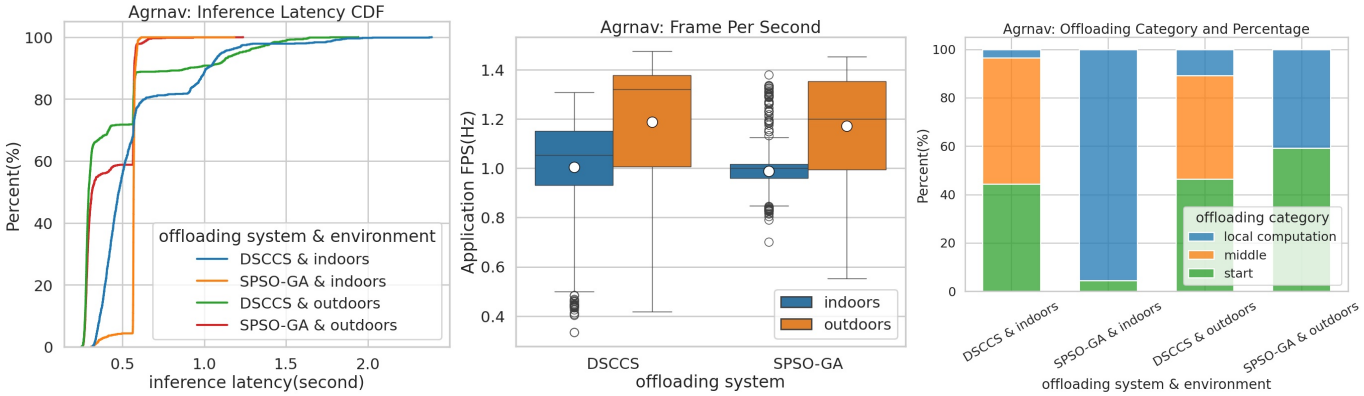


(a) CDF of end-to-end inference latency

(b) Processing frame per second (FPS). The circle marks the average.

(c) Categories and percentage of various scheduling

Fig. 6: The performance of Kapao under difference offloading systems and environments.



(a) CDF of end-to-end inference latency

(b) Processing frame per second (FPS). The circle marks the average.

(c) Categories and percentage of various scheduling

Fig. 7: The performance of AGRNav under difference offloading systems and environments.

environment	indoors		outdoors	
offloading system	DSCCS	SPSO-GA	DSCCS	SPSO-GA
transmitted data/MB	3.511	3.099	3.517	3.097
received data/MB	0.001	0.001	0.001	0.001
transmission time/s	0.164	0.304	0.306	0.173
inference time/s	0.316	0.527	0.465	0.415
percentage(%)	51.896	57.675	65.712	41.681

TABLE II: Kapao: the average statistics of transmitted & received data volume, transmission time, inference time and percentage that transmission time takes up inference time at inference of each frame. Note that the data volume are recorded excluding the cases with local computation.

environment	indoors		outdoors	
offloading system	DSCCS	SPSO-GA	DSCCS	SPSO-GA
transmitted data/MB	1.638	2.000	1.744	2.000
received data/MB	1.860	2.000	1.962	2.000
transmission time/s	0.363	0.011	0.185	0.101
inference time/s	0.584	0.563	0.445	0.412
percentage(%)	62.144	1.956	41.662	24.545

TABLE III: AGRNav: the average statistics of transmitted & received data volume, transmission time, inference time and percentage that transmission time takes up inference time at inference of each frame. Note that the data volume are recorded excluding the cases with local computation.

B. Energy Consumption

C. Lessons learned

V. Conclusion

The conclusion goes here.

bandwidth upper bound.

which type.

Acknowledgment

The authors would like to thank...

References

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

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

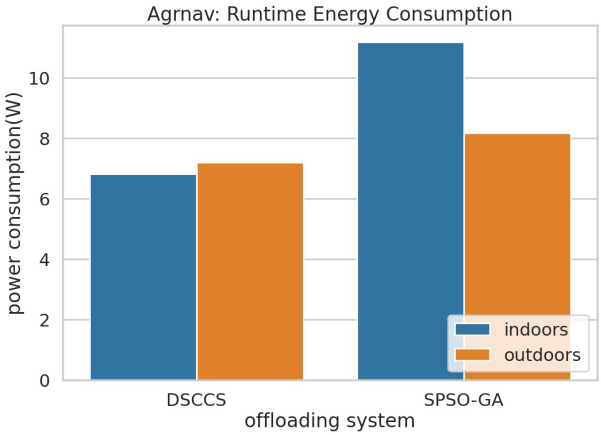


Fig. 8: AGRNav: power consumption at different offloading systems and environments.

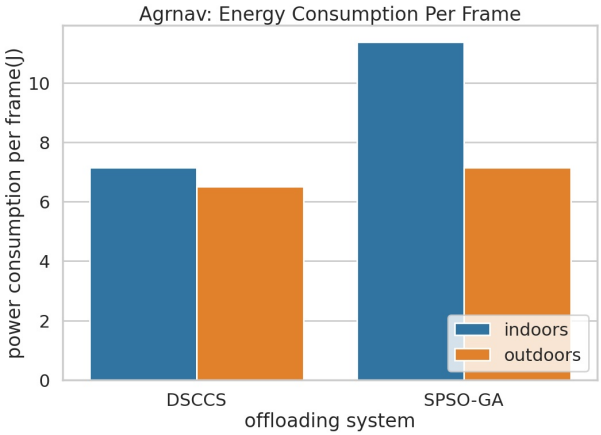


Fig. 9: AGRNav: power consumption per frame at different offloading systems and environments.

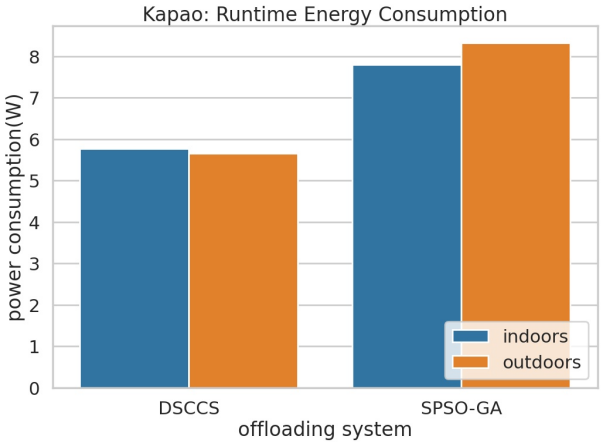


Fig. 10: Kapao: power consumption at different offloading systems and environments.

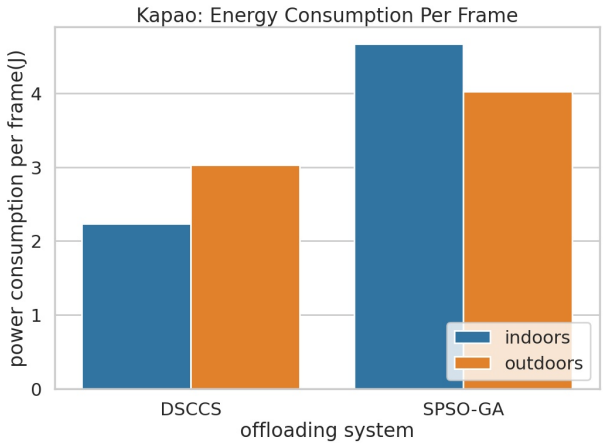


Fig. 11: Kapao: power consumption per frame at different offloading systems and environments.

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

[4] Cao, A.-Q., and de Charette, R. Monoscene: Monocular 3d semantic scene completion. In *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition* (2022), pp. 3991–4001.

[5] Crankshaw, D., Sela, G.-E., Mo, X., Zumar, C., Stoica, I., Gonzalez, J., and Tumanov, A. Inferline: latency-aware provisioning and scaling for prediction serving pipelines. In *Proceedings of the 11th ACM Symposium on Cloud Computing* (2020), pp. 477–491.

[6] Défossez, A., Adi, Y., and Synnaeve, G. Differentiable model compression via pseudo quantization noise. *arXiv preprint arXiv:2104.09987* (2021).

[7] Ding, M., Wang, P., López-Pérez, D., Mao, G., and Lin, Z. Performance impact of los and nlos transmissions in dense cellular networks. *IEEE Transactions on Wireless Communications 15*, 3 (2015), 2365–2380.

[8] Elgazzar, K., Martin, P., and Hassanein, H. S. Cloud-assisted computation offloading to support mobile services. *IEEE Transactions on Cloud Computing 4*, 3 (2014), 279–292.

[9] Fang, Z., Yu, T., Mengshoel, O. J., and Gupta, R. K. 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* (2017), pp. 2067–2070.

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

[11] Gong, C., Chen, Y., Lu, Y., Li, T., Hao, C., and Chen, D. Vecq: Minimal loss dnn model compression with vectorized weight quantization. *IEEE Transactions on Computers 70*, 5 (2020), 696–710.

[12] Gou, J., Yu, B., Maybank, S. J., and Tao, D. Knowledge distillation: A survey. *International Journal of Computer Vision 129*, 6 (2021), 1789–1819.

[13] Honkote, V., Kurian, D., Muthukumar, S., Ghosh, D., Yada, S., Jain, K., Jackson, B., Klotchkov, I., Nimmagadda, M. R., Dattawadkar, S., 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)* (2019), IEEE, pp. 48–50.

[14] Hu, C., Bao, W., Wang, D., and Liu, F. Dynamic adaptive dnn surgery for inference acceleration on the edge. In *IEEE INFOCOM 2019-IEEE Conference on Computer Communications* (2019), IEEE, pp. 1423–1431.

[15] Hu, Y., Imes, C., Zhao, X., Kundu, S., Beerel, P. A., Crago, S. P., and Walters, J. P. Pipeedge: Pipeline parallelism for large-scale model inference on heterogeneous edge devices. In *2022 25th Euromicro Conference on Digital System Design (DSD)* (2022), IEEE, pp. 298–307.

[16] JOSEPH, K. J., KHAN, S., KHAN, F. S., AND BALASUBRAMANIAN, V. N. Towards open world object detection. In *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition (CVPR)* (June 2021), pp. 5830–5840.

[17] LI, A., SONG, S. L., CHEN, J., LI, J., LIU, X., TALLENT, N. R., AND BARKER, K. J. Evaluating modern gpu interconnect: Pcie, nvlink, nv-sli, nvswitch and gpudirect. *IEEE Transactions on Parallel and Distributed Systems 31*, 1 (2019), 94–110.

[18] LI, Q., GAMA, F., RIBEIRO, A., AND PROROK, A. Graph neural networks for decentralized multi-robot path planning. In *2020 IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS)* (2020), IEEE, pp. 11785–11792.

[19] LI, Y., YU, Z., CHOY, C., XIAO, C., ALVAREZ, J. M., FIDLER, S., FENG, C., AND ANANDKUMAR, A. Voxformer: Sparse voxel transformer for camera-based 3d semantic scene completion. In *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition* (2023), pp. 9087–9098.

[20] LIN, T., KONG, L., STICH, S. U., AND JAGGI, M. Ensemble distillation for robust model fusion in federated learning. *Advances in Neural Information Processing Systems 33* (2020), 2351–2363.

[21] LIU, R., AND CHOI, N. 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.

[22] LIU, S., LI, X., LU, H., AND HE, Y. Multi-object tracking meets moving uav. In *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition (CVPR)* (June 2022), pp. 8876–8885.

[23] MASIUKIEWICZ, A. 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.

[24] MCNALLY, W., VATS, K., WONG, A., AND MCPHEE, J. Rethinking keypoint representations: Modeling keypoints and poses as objects for multi-person human pose estimation. In *European Conference on Computer Vision* (2022), Springer, pp. 37–54.

[25] NARAYANAN, D., SHOEYBI, M., CASPER, J., LEGRESLEY, P., PATWARY, M., KORTHIKANTI, V., VAINBRAND, D., KASHINKUNTI, P., BERNAUER, J., CATANZARO, B., 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* (2021), pp. 1–15.

[26] NOORMOHAMMADPOUR, M., AND RAGHAVENDRA, C. S. Datacenter traffic control: Understanding techniques and tradeoffs. *IEEE Communications Surveys & Tutorials 20*, 2 (2017), 1492–1525.

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

[28] OHKAWA, T., YAMASHINA, K., KIMURA, H., OOTSU, K., AND YOKOTA, T. Fpga components for integrating fpgas into robot systems. *IEICE TRANSACTIONS on Information and Systems 101*, 2 (2018), 363–375.

[29] PEI, Y., MUTKA, M. W., AND XI, N. Connectivity and bandwidth-aware real-time exploration in mobile robot networks. *Wireless Communications and Mobile Computing 13*, 9 (2013), 847–863.

[30] PYTORCH. pytroch. https://pytorch.org/, 2024.

[31] REN, Y., TUNG, C.-W., CHEN, J.-C., AND LI, F. Y. Proportional and preemption-enabled traffic offloading for ip flow mobility: Algorithms and performance evaluation. *IEEE Transactions on Vehicular Technology 67*, 12 (2018), 12095–12108.

[32] SARKAR, N. I., AND MUSSA, O. The effect of people movement on wi-fi link throughput in indoor propagation environments. In *IEEE 2013 Tencon-Spring* (2013), IEEE, pp. 562–566.

[33] SIMONYAN, K., AND ZISSERMAN, A. Very deep convolutional networks for large-scale image recognition, 2015.

[34] WANG, H., POTLURI, S., LUO, M., SINGH, A. K., SUR, S., AND PANDA, D. K. Mvapich2-gpu: optimized gpu to gpu communication for infiniband clusters. *Computer Science-Research and Development 26*, 3 (2011), 257–266.

[35] WANG, J., SUN, Z., GUAN, X., SHEN, T., ZHANG, Z., DUAN, T., HUANG, D., ZHAO, S., AND CUI, H. 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).

[36] WANG, L., AND YOON, K.-J. Knowledge distillation and student-teacher learning for visual intelligence: A review and new outlooks. *IEEE transactions on pattern analysis and machine intelligence 44*, 6 (2021), 3048–3068.

[37] WU, T., HE, S., LIU, J., SUN, S., LIU, K., HAN, Q.-L., AND TANG, Y. A brief overview of chatgpt: The history, status quo and potential future development. *IEEE/CAA Journal of Automatica Sinica 10*, 5 (2023), 1122–1136.

[38] XIA, Z., LIU, Y., LI, X., ZHU, X., MA, Y., LI, Y., HOU, Y., AND QIAO, Y. Scpnet: Semantic scene completion on point cloud. In *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition* (2023), pp. 17642–17651.

[39] XIANG, Y., AND KIM, H. Pipelined data-parallel cpu/gpu scheduling for multi-dnn real-time inference. In *2019 IEEE Real-Time Systems Symposium (RTSS)* (2019), IEEE, pp. 392–405.

[40] YANG, X., LIN, H., LI, Z., QIAN, F., LI, X., HE, Z., WU, X., WANG, X., LIU, Y., LIAO, Z., ET AL. Mobile access bandwidth in practice: Measurement, analysis, and implications. In *Proceedings of the ACM SIGCOMM 2022 Conference* (2022), pp. 114–128.

[41] YANG, Y., JUNTAO, L., AND LINGLING, P. Multi-robot path planning based on a deep reinforcement learning dqn algorithm. *CAAI Transactions on Intelligence Technology 5*, 3 (2020), 177–183.

[42] ZHUANG, Y., ZHAO, H., ZHENG, L., LI, Z., XING, E., HO, Q., GONZALEZ, J., STOICA, I., AND ZHANG, H. On optimizing the communication of model parallelism. *Proceedings of Machine Learning and Systems 5* (2023).