Intra-DP: A High Performance Collaborative Inference System for Mobile Edge Computing

Submission #676 to ACM MobiCom 2025

Abstract

Deep Neural Networks (DNNs) have revolutionized mobile applications in various domains. However, their deployment on resource-constrained mobile devices presents significant challenges in achieving real-time performance while managing limited computational resources and battery life. While Mobile Edge Computing (MEC) offers collaborative inference with GPU servers as a promising solution, existing approaches primarily rely on layer-wise model partitioning, which faces significant transmission bottlenecks caused by the sequential execution of DNN operations. To address this challenge, we present Intra-DP, a high-performance collaborative inference system optimized for DNN inference on MEC. Intra-DP employs a novel parallel computing technique based on local operators (i.e., operators whose minimum unit input is not the entire input tensor, such as the convolution kernel). By cutting their operations into several independent sub-operations and overlapping the computation and transmission of different sub-operations through parallel execution, Intra-DP significantly mitigates transmission bottlenecks in MEC, achieving fast and energy-efficient inference. The evaluation demonstrates that Intra-DP reduces inference time and per-inference energy consumption by up to 50% and 75%, respectively, compared to state-of-the-art baselines, without sacrificing accuracy.

1 Introduction

Deep Neural Networks (DNNs) have powered a wide range of mobile applications, from intelligent wearable sensors [13, 59] and autonomous vehicles [5, 69] to smart manufacturing systems [85, 97]. These applications are built upon fundamental advances in object detection [35, 49, 54], robotic control [41, 78, 89], and environmental perception [11, 42, 82], all of which demand fast inference for swift responses. Deploying DNN models on real-world mobile devices (e.g., smartphones and IoT devices) presents two major challenges: the need for fast inference and the limitation of computing resources, including computational power and battery life.

Recent studies [2, 8, 29, 33, 56, 87, 95] have proposed mobile edge computing (MEC) as a promising solution for fast and energy-efficient inference by leveraging GPU servers at the edge of the radio access network. Conventional MEC approaches (illustrated in Fig. 1) are categorized into device-only, server-only, and collaborative inference [14, 43, 72]. Device-only inference is constrained by limited on-device computing resources (Sec. 2.1), while server-only inference suffers from bandwidth limitations (Sec. 2.2). Collaborative inference achieves a better balance between computation and communication through layer-wise model partitioning (layer partitioning), which distributes DNN layers across mobile devices and GPU servers. Since some intermediate layers produce significantly smaller activations than raw input data [29], this approach reduces transmission overhead and further optimizing resource utilization (Sec.2.3).

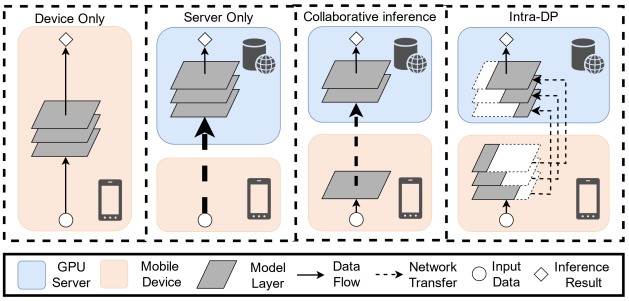


Figure 1: Comparison between conventional Mobile Edge Computing approaches and Intra-DP. Inference results computed on the GPU server must be transmitted back to the mobile device for application utilization.

However, despite its advantages, the sequential nature of layer partitioning introduces significant transmission delays, making it a major bottleneck for collaborative inference in MEC environments. Existing methods enforce strictly sequential execution, consisting of early-layer computation on mobile devices, intermediate result transmission, and final inference on GPU servers. Limited network bandwidth capacity (due to MEC network limitations and practical instability, Sec. 2.2) prolongs transmission (about half of inference time in our experiments), causing significant idle periods on mobile devices. This not only increases overall inference time but also results in significant energy waste (around 40%). Although pipeline execution [15, 22, 23] attempts to alleviate transmission overheads by overlapping computation and communication across multiple requests, its primary benefit lies in improving throughput rather than reducing the inference time of individual requests, which is a crucial requirement for real-time mobile applications.

To address these limitations, a parallel execution strategy that overlaps computation and communication within a single inference request offers a promising alternative. Unlike traditional sequential execution, this approach allows computation and data transmission to proceed concurrently, significantly reducing idle time on mobile devices. This not

only speeds up inference but also improves energy efficiency, as idle-period energy consumption is primarily driven by core components such as the CPU, GPU, and memory modules [37] (95% of total energy in our experiment), while wireless network interface cards contribute only 1.5%.

However, implementing this parallel computing technique poses three key challenges. ① Traditional parallel computing methods (data, tensor, and pipeline parallelism) work well in data centers but struggle in MEC due to batch size limits, high synchronization costs, and transmission delays (Sec. 2.4), necessitating a finer-grained scheduling unit. ② Fine-grained parallelism is vulnerable to consistency issues, where the failure or error of any single unit (e.g., receiving incorrect input) can compromise the entire inference result. ③ Minimizing latency and energy consumption requires a new scheduling paradigm to balance computation and transmission, but finer-grained scheduling increases the search space from $O(n)$ (layer-wise) to $O(n^2)$, complicating optimization.

To address these challenges, we propose Intra-DP (Intra-Data Parallel), a high-performance collaborative inference system optimized for MEC environments. ① Intra-DP introduces a novel parallel computing technique that defines scheduling units based on local operators, where the minimum input is a subset of the full tensor (e.g., individual elements for ReLU [16], tensor blocks for convolution [56]). Their operations can be decomposed into independent sub-operations, enabling those in subsequent layers to start execution as soon as their required inputs are ready, without waiting for all sub-operations in the current layer to complete. This enables finer-grained parallel scheduling, achieving computation-communication overlap within a single inference request. ② To ensure result correctness and completeness, Local Operation Parallelism (LOP) manages data dependencies between operations, where one operation's output serves as another's input, ensuring each operator receives the correct input and propagates the correct output during parallel execution. ③ To achieve fast and energy-efficient inference, Local Operation Scheduling Strategy (LOSS) formulates scheduling as a constrained optimization problem, and optimally distributes local operations between mobile devices and GPU servers, using a solution derived from differential evolution [68].

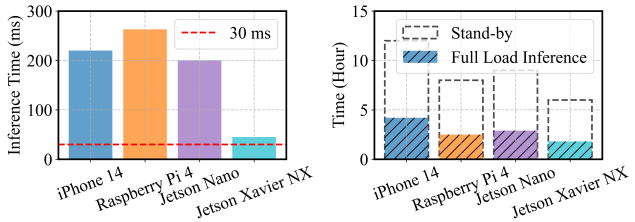
In summary, this paper makes the following contributions:

- We propose Intra-DP, a novel collaborative inference system that enables fine-grained parallel scheduling based on local operators, specifically optimized for MEC environments. The code is released on https://github.com/mobicom25paper676/intraDP.
- We develop a comprehensive parallel computing technique that guarantees inference correctness based on local operations while enabling efficient computation-transmission overlapping.
- We design an innovative scheduling algorithm leveraging LOP to optimize both inference speed and energy efficiency.
- We implement an adaptive control mechanism integrated with LOSS to dynamically handle real-time network bandwidth fluctuations in MEC environments.
- We empirically evaluate Intra-DP through extensive experiments, demonstrating its superiority over state-of-the-art baselines in MEC environments in terms of inference latency and energy efficiency. Specifically, Intra-DP reduces inference time by up to 50% and per-inference energy consumption by up to 75% compared to baselines without sacrificing accuracy.

2 Background and Motivation

2.1 Device-only Inference

Device-only inference, which deploys DNN models directly on mobile devices, is severely constrained by the limited computational power of processors and battery capacity. As shown in Fig. 2a, the inference latency on these devices exceeds the 30ms threshold required for smooth video fluency [25, 26] (indicated by the red dotted line). Moreover, Fig. 2b demonstrates that frequent model inference significantly reduces device standby time to just 20–40% of its normal duration, severely affecting user experience and overall device practicality.



(a) Inference Time **(b) Battery Life**

Figure 2: The performance of VGG-16 under device-only inference across different mobile devices [7, 10, 55, 58, 62, 64].

2.2 Server-only Inference

Server-only inference, which deploys DNN models directly on a GPU server, is susceptible to long tail delays and requires substantial bandwidth for data transmission. As shown in Fig. 3, under the same testbed settings as Sec. 5 (a robot equipped with a Jetson Xavier NX [62] and a GPU server with an NVIDIA GeForce GTX 3080), inference time increases sharply as bandwidth decreases. However, in real-world scenarios, mobile devices primarily rely on wireless networks, which offer high mobility but limited bandwidth capacity compared to data center networks equipped with high-speed technologies such as InfiniBand [61] or PCIe [40] (40–500 Gbps).

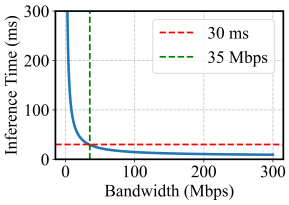


Figure 3: The inference time of VGG-16 under server-only inference across different network bandwidth.

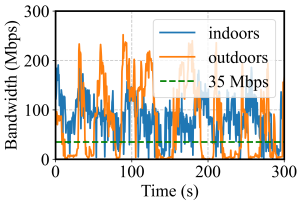


Figure 4: The wireless transmission instability of TCP between our robot and the base station in MEC networks.

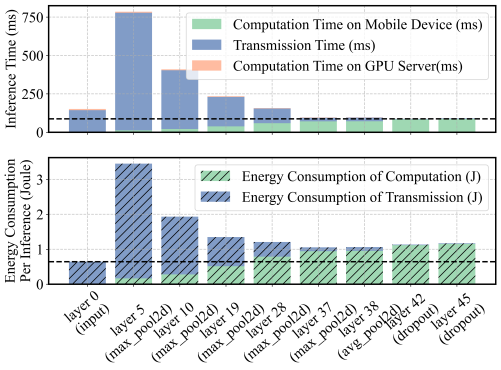


Figure 5: The performance of different layer partitioning strategies for VGG-19 under 35 Mbps in our experiments. The X-axis represents various partitioning points, where 'layer i' denotes that all layers up to and including the i_{th} layer are executed on the robot, while the remaining layers are offloaded to the GPU server.

The bandwidth capacity of wireless networks is inherently constrained by both theoretical limits and practical challenges in MEC. While Wi-Fi 6 can achieve a peak bandwidth of 1.2 Gbps per stream [48], mobile devices lack the necessary hardware to fully leverage this capacity [88]. Furthermore, the actual available bandwidth varies significantly due to factors such as device mobility [53, 63], signal obstruction [18, 71], and channel contention [3, 70], further limiting effective bandwidth.

To examine wireless instability in MEC scenarios, we conducted a robot surveillance experiment where four-wheeled robots navigated through a lab (indoors) and a campus garden (outdoors) at speeds of 5–40 cm/s. Using iperf [1], we measured real-time wireless bandwidth capacity between the robot and a base station over TCP [76] at 0.1-second intervals for five minutes. As shown in Fig. 4, the average bandwidth was 93 Mbps indoors and 73 Mbps outdoors, with outdoor measurements exhibiting higher fluctuations and occasional near-zero drops due to obstacles and reduced signal reflections, indicating that server-only inference is difficult to sustain under realistic wireless network conditions.

2.3 Existing Collaborative Inference

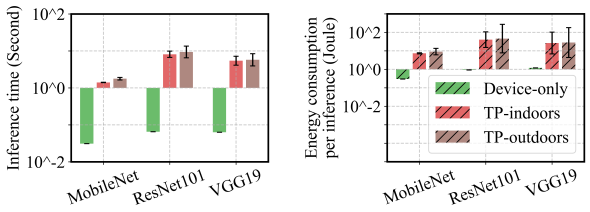
Existing collaborative inference methods [8, 14, 29, 33, 43, 56, 87, 95] primarily focus on layer partitioning to balance inference time and energy consumption, as shown in Fig.5. "Layer 0" represents server-only inference, where all model layers run on a GPU server, and transmission time varies with network bandwidth, even under the same partitioning strategy. These methods can be categorized by their optimization objectives: accelerating DNN inference[29, 36, 43, 56, 87] or minimizing energy consumption under deadline constraints [14, 44, 81]. The widespread adoption of layer partitioning in mobile applications has drawn significant research attention [8, 29, 33, 56, 87, 95], as optimal strategies depend on multiple factors, including model architecture, hardware capabilities (e.g., GPU servers and mobile devices), network conditions, and application-specific trade-offs between inference speed and energy efficiency. However, all existing methods [8, 14, 29, 33, 36, 43, 44, 56, 81, 87, 95] are inherently constrained by transmission delays due to the sequential nature of layer partitioning, a limitation that can be effectively addressed by Intra-DP.

2.4 Related Parallel Computing Techniques

Parallel computing techniques have been extensively explored and proven effective in modern data centers [31, 83, 98]. However, they are unsuitable for MEC due to fundamental differences in computational constraints and latency requirements. The three primary parallelism strategies in data centers are data parallelism (DP), which replicates the model across devices and processes mini-batches in parallel; tensor parallelism (TP), which partitions computations within a layer across multiple devices; and pipeline parallelism (PP), which distributes different layers of DNN model across devices (layer partitioning) and executes them in a pipelined manner (pipeline execution) to reduce idle time.

DP's efficiency is fundamentally constrained by batch size requirements [57]. In data centers, large batch sizes (e.g., 16 images) are split into mini-batches (e.g., 2 images each), maintaining computational efficiency. However, MEC requires real-time inference with minimal latency, typically processing a single input at a time (e.g., 1 image). Further splitting such small inputs into sub-mini-batches (e.g., 1/4 of an image) is impractical, making DP inefficient. Some methods [93, 96] split input images across an IoT device group based on computational power, but in MEC environments with GPU servers, this approach loses effectiveness. The significant computational disparity between mobile devices and GPU servers leads to a near-complete reliance on server-side execution, essentially degrading into server-only inference.

TP enables parallel computation within a single layer across devices but incurs substantial communication overhead due to the required all-reduce synchronization [98]. We evaluated DINA [56], a state-of-the-art TP method, on our testbed (Sec.5). As shown in Fig.6, the excessive all-reduce



(a) Inference Time (b) Energy Consumption

Figure 6: The performance of TP for different models.

overhead results in 45.2× to 143.9× longer inference time and 28.5× to 62.7× higher energy consumption per inference than device-only inference. While there have been efforts to reduce the communication overhead of TP in data centers, such as distributing each layer along both the spatial and temporal dimensions [77], these methods remain ineffective in MEC due to the persistent all-reduce requirement. In contrast, Intra-DP eliminates this communication for local operators, further reducing overhead.

PP employs pipeline execution to enhance throughput by overlapping computation and communication across multiple requests. However, this approach does not reduce the completion time of a single inference [15, 22, 23]. SPINN [39] attempts to optimize transmission efficiency through an early-exit policy combined with layer partitioning. While this reduces the number of transmissions by trading off accuracy, it introduces unpredictable and incorrect inference results, and requires careful expert tuning of parameters. Additionally, transmission delays persist in SPINN, an issue that can be effectively addressed by Intra-DP without sacrificing accuracy.

In conclusion, conventional parallel computing techniques, despite their success in data centers, are unsuitable for MEC. DP is constrained by batch size limitations, TP suffers from excessive synchronization overhead, and PP's pipeline execution focuses on throughput rather than reducing inference latency. Consequently, existing collaborative inference in MEC relies on layer partitioning, which enforces sequential execution and introduces transmission delays, making them a major bottleneck for low-latency inference.

3 System Overview

In this section, we introduce Intra-DP, a high-performance collaborative inference system optimized for MEC environments. In this setting, resource-constrained mobile devices (e.g., robots, drones, smartphones) collaborate with GPU servers through wireless networks (e.g., Wi-Fi 6/5G) to achieve real-time and energy-efficient inference on local inputs.

3.1 Key Insight

Existing methods are limited to layer partitioning, which enforces serial execution and limits parallelism. Each DNN layer comprises one or more operators (e.g., convolution [56], softmax [50]), which must be executed sequentially to ensure

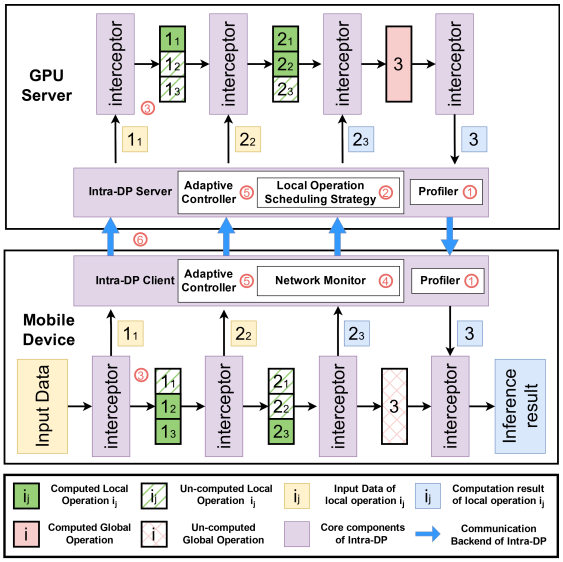


Figure 7: Overview and inference workflow of Intra-DP. Local operation i_j represents the j-th local operation within the i-th operator of the model.

computational correctness. As a result, current approaches can only partition these serial execution sequences at the layer level, restricting opportunities for parallel execution.

Our key insight is that, although operators are the smallest computation units at the system level, their computations (operation) can still be decomposed into several independent sub-operations based on their computational properties. Specifically, local operators, whose computations do not require access to the entire input tensor, enable finer-grained parallel execution. While existing methods treat each operator's computation as an indivisible operation (e.g., a full convolution for a kernel), Intra-DP leverages this by treating each individual computation within a local operator, based on its minimal input unit, as an independent local operation. Since these local operations are independent, those in subsequent layers can be computed as soon as their required inputs are ready, without waiting for all operations in the current layer to finish. This independence enables a new parallelization opportunity within serial execution sequences, breaking the strict sequential execution constraint.

3.2 Overall Workflow

As illustrated in Fig. 7, Intra-DP consists of three integrated stages: Profiling, Scheduling Optimization, and Runtime.

The first stage, Profiling (①), is an offline process that collects essential runtime traces for Scheduling Optimization. It runs only once and captures key execution characteristics, including: i) operator types and input/output sizes; ii) execution times of operations on the mobile device and GPU server; iii) data dependencies between operations (i.e., the output of one operation serves as the input for another).

Using this profiling data, the system enters the Scheduling Optimization stage, where it applies Local Operation Scheduling Strategy (Sec. 4.3, ②) to achieve low-latency, energy-efficient inference. This strategy determines the optimal distribution of local operations, balancing computational workload and transmission synchronization. To mitigate the impact of network fluctuations, Intra-DP precomputes optimal scheduling plans for varying network bandwidth conditions in 1 MB intervals within the typical MEC bandwidth range (0~50 MB), eliminating runtime optimization overhead.

During Runtime, Intra-DP utilizes Operator Interceptors (Sec. 4.1, ③) to manage input/output tensor partitioning and reconstruction for each operator, enabling partial tensor processing based on local operations. To ensure reliability in MEC environments, the system continuously monitors network conditions utilizing established tools [90] (④) and Adaptive Controllers (Sec. 4.4, ⑤) dynamically adjust scheduling strategies based on real-time bandwidth fluctuations, ensuring stable performance. Then, Local Operation Parallelism (Sec. 4.2, ⑥) enables parallel execution between the client and server while preserving data dependencies for correct inference. To support seamless transitions between different schedules without delays, Intra-DP maintains a model replica on the GPU server (Fig. 7), allowing instant adaptation to network variations.

Despite these optimizations, Intra-DP introduces negligible overhead beyond the original inference process. The only additional cost stems from tensor splitting and combining. However, the impact of tensor splitting is effectively hidden due to concurrent data transmission and computation. While tensor combining introduces potential waiting time due to data dependencies, LOSS minimizes this impact by formulating the waiting overhead as a nonlinear optimization problem and solves it efficiently. This careful design ensures that the system of Intra-DP maintains high execution efficiency with minimal extra cost.

4 Detailed Design

4.1 Local Operators and Global Operators

Since the performance gains of Intra-DP rely on the parallel execution of local operations, it is crucial to determine the minimal input unit required for each operator based on its computational characteristics. To achieve this, Intra-DP employs Operator Interceptors (Fig. 7 ③) to handle input/output tensor partitioning and reconstruction, enabling partial tensor processing, and classify operators into local and global categories: local operators process only a subset of the input tensor, whereas global operators require the entire tensor. We identify three common types of local operators in models commonly used on mobile devices (mobile models):

- Element-wise local operators operate on individual tensor elements and are widely used in activation functions such as ReLU [16], Sigmoid [91], and SiLU [34]. However, activation functions like softmax [50] require access to the entire input tensor and are therefore global operators.
- Block-wise local operators process tensor blocks at corresponding positions, commonly found in convolution-related layers such as convolution [56] and max pooling [74]. The block size is determined by the operator's parameters (e.g., kernel size, padding, dilation) [66].
- Row-wise local operators operate on individual rows of the input tensor and are widely used in matrix operations such as addition [92] and multiplication [21]. These operations split the input matrix by rows, ensuring that each device processes an independent subset while sharing the same layer parameter matrix. According to matrix calculation principles as following, the computation of row a_1 results in $(c_{11} \cdots c_{1n})$, which is itself a row and can be directly processed by subsequent matrix operators in the same manner, eliminating the need for all-reduce communication.

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

  Unlike TP, which partitions the layer parameter matrix and replicates the full input matrix across multiple devices, row-wise local operators allow independent row processing across devices while maintaining a shared parameter matrix, avoiding all-reduce communication. Moreover, LOP treats operators with layer parameter matrices containing only one row as global operators.

Models with many local operators, prevalent in mobile models such as CNNs for computer vision [54] and point cloud processing [78], can benefit from Intra-DP through parallel execution. Table 1 quantifies their distribution across several mobile models based on the above definition.

Model	Local / Global	Model	Local / Global
DenseNet [32]	428 / 3	RegNet [86]	231 / 3
ResNet [75]	341 / 3	VGGNet [73]	73 / 5
ResNeXt [84]	342 / 3	ConvNeXt [80]	340 / 6

Table 1: Number of local/global operators in mobile models.

While local operators dominate CNN-based models, transformer-based models rely on self-attention mechanisms, which use softmax [50], a typical global operator, to compute attention weights over entire input sequences [30]. Since each transformer layer contains at least one global operator, the performance gains of Intra-DP over existing collaborative inference methods are limited, as its advantages primarily come

from parallelizing local operations, which is constrained by the presence of global operators. However, Intra-DP still outperforms existing methods by enabling parallel execution of local operations between consecutive global operators and seamlessly falling back to conventional layer partitioning when handling models without local operators. Although transformer-based models are rarely deployed directly on mobile devices, their increasing adoption motivates us to further enhance Intra-DP. As future work, we will support more types of local operators and develop lightweight synchronization techniques for global operators, such as synchronizing only the sum results in softmax [50] instead of transferring the entire input tensor, to reduce synchronization overhead and enhance Intra-DP's applicability.

4.2 Local Operation Parallelism

In this section, we demonstrate how Intra-DP ensures inference correctness based on local operators and how LOP achieves low latency and energy efficiency through the parallel execution of local operations (Fig. 7 ⑥).

LOP guarantees inference correctness by comprehensively managing dependencies between local and global operators. It tracks data dependencies to ensure that the output of one operation is correctly used as the input for subsequent operations, considering a dependency established if any part of the output serves as input. For local operators, it preserves computational correctness by enforcing the appropriate execution sequence and ensuring that each operation receives the correct input (either a raw input data or the output from a preceding operation) based on the DNN model structure and operator computation logic. For global operators, synchronization barriers are introduced to guarantee that all required input tensors are fully assembled before execution. By maintaining proper data flow throughout the inference process, this approach guarantees correctness across both local and global operators.

Fig. 8 compares the workflow of LOP with conventional TP and PP (layer partitioning). To reduce transmission overhead, LOP employs two key optimizations. First, it overlaps computation and transmission for different local operations to maximize execution efficiency (e.g., transmitting the input for '$LO1_1$' while computing '$LO1_2$', '$LO1_3$', and '$LO2_3$' on the robot). Second, it exploits data locality by prioritizing execution on devices that already contain the required input (e.g., '$LO2_1$' directly using the output of '$LO1_1$' on the GPU server). Unlike TP, which suffers from high communication costs due to frequent all-reduce synchronization, LOP limits synchronization to global operators only, eliminating unnecessary data transfers. Although LOP incurs finer-grained communication and a higher volume than PP, its early overlapping strategy effectively reduces overall transmission time, outperforming PP's sequential execution for individual inference

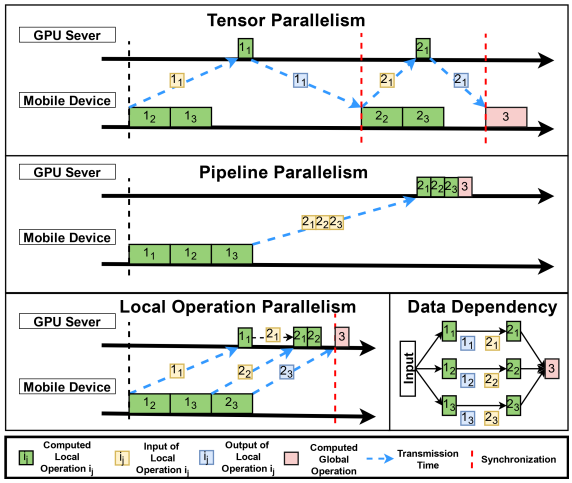


Figure 8: Workflow of TP, PP and LOP. In the three cases above, each local operator executes three operations with identical computation times on both the mobile device and the GPU server, along with the corresponding transmission time, while the lower right corner illustrates the data dependencies between operations.

requests. By efficiently overlapping computation and communication, Intra-DP minimizes idle time on mobile devices, accelerating inference and enhancing energy efficiency, as idle-period energy consumption is primarily dominated by core components such as the CPU, GPU, and memory modules rather than the wireless network interface cards.

4.3 Local Operation Scheduling Strategy

In this section, we detail how Intra-DP optimizes computation and transmission scheduling for local operations, achieving fast and energy-efficient inference (Fig. 7 ②).

LOSS formulates local operation scheduling as a constrained optimization problem, efficiently solved via differential evolution [68]. To streamline heterogeneous operation integration, it unifies global and local operations by treating global operators as special cases that aggregate all outputs from preceding local operations. The key performance boost stems from computation-communication overlapping, where LOSS leverages data locality by prioritizing execution on devices that already contain the required input. To further optimize scenarios where multiple operations on mobile devices and GPU servers simultaneously depend on the same output, especially in block-wise operations, LOSS introduces local operation replication, selectively duplicating operations across devices. This strategy incurs minimal computational redundancy but significantly reduces data transmission costs, particularly benefiting distributed operations that share input dependencies.

4.3.1 DNN Execution Model A deep neural network (DNN) is represented as a directed acyclic graph $G = (V, E)$, where vertices V denote layers (operators) and edges E indicate data dependencies. Two virtual vertices, *input* and *output*,

represent the model's initial input and final inference result. The execution spans two instances: one on a resource-constrained mobile device (M) and the other on a GPU server (R).

4.3.2 Variables and Functions We treat each calculation of an operator based on its minimum input unit as an operation, and define $OP(v)$ as the set of operations for vertex v, where global operators always have $|OP(v)| = 1$. Operation allocations to M and R are represented by $x_M(v), x_R(v) \subseteq OP(v)$, with overlaps ($x_M(v) \cap x_R(v) \neq \emptyset$) occurring when local operations are replicated, especially for block-wise operators.

Execution starts at times $s_M(v), s_R(v) \geq 0$, and computation durations on each device are estimated as $C_M(v)$ and $C_R(v)$. If no operation is performed on a device, its computation time is zero. Data transmission times include $T_{\mathrm{MR}}(u, v)$ for sending data from M to R and $T_{\mathrm{RM}}(u, v)$ for the reverse direction, both determined by network bandwidth and dependent on the volume of input/output data associated with the respective operations.

4.3.3 Objective Function The goal is to minimize end-to-end inference latency while ensuring that the final results are available on the mobile device, thereby reducing its idle time:

$$\min T = s_M(output), \tag{2}$$

4.3.4 Constraints **Data Partitioning Constraints.** To ensure correctness, all operations must be executed: $x_M(v) \cup x_R(v) = OP(v), \forall v \in V$.

Location Constraints. Input data is initially available on M, ensuring $x_M(input) = input$ and $x_R(input) = \emptyset$. Similarly, the final results must reside on M, so $x_M(output) = output$ and $x_R(output) = \emptyset$.

Data Dependency Constraints. Operations start only after receiving all required inputs. For each v with parent connections $e = (u, v)$, execution on M follows:

$$s_M(v) = \begin{cases} s_M(u) + C_M(u), \\ \quad \text{if } x_M(v) - child(x_M(u)) = \emptyset, \\ \max\left(s_R(u) + C_R(u) + T_{\mathrm{RM}}(u, v), \right. \\ \quad \left. s_M(u) + C_M(u)\right), \\ \quad \text{if } x_M(v) - child(x_M(u)) \neq \emptyset. \end{cases} \tag{3}$$

Here, $\mathrm{child}(x(u))$ denotes the set of operations consuming outputs of $x(u)$ according to their data dependencies, and the transmission volume in $T_{\mathrm{MR}}(u, v)$ is given by $x_M(v) - \mathrm{child}(x_M(u))$. When $x_M(v) - \mathrm{child}(x_M(u)) = \emptyset$, all required inputs for $x_M(v)$ are produced by operations on M; otherwise, $x_M(v)$ also depends on outputs from operations executed on R. Similar constraints apply to R.

Computational Efficiency Constraints. To reduce excessive kernel launch overhead from input fragmentation while LOSS processes minimal input units, Intra-DP adopts operation batching, which consolidates multiple operations into a single system-level kernel launch, to avoid redundant launches. By managing the partitioning and reconstruction of input/output tensors for each operator, this approach preserves fine-grained control over local operations while minimizing redundant executions. To formalize this, for all operators $v \in V$: $x_M(v) = \{i \mid i \in [\min(x_M(v)), \max(x_M(v))]\}$, $x_R(v) = \{i \mid i \in [\min(x_R(v)), \max(x_R(v))]\}$.

Transmission Efficiency Constraints. Building on the insight from existing layer partitioning methods that certain intermediate DNN layers produce significantly smaller outputs than raw input data [29], we impose the following constraints to minimize transmission overhead by eliminating redundant data transfers whenever possible. This also substantially reduces the search space, significantly accelerating the solving process of LOSS. For layers u whose outputs exceed the raw input size ($u \in \Pi$), with all $e = (u, v) \in E$: $x_M(v) - \mathrm{child}(x_M(u)) = \emptyset$, and $x_R(v) - \mathrm{child}(x_R(u)) = \emptyset$.

4.3.5 Solution Algorithm As the search space increases from $O(n)$ (layers) to $O(n^2)$ (operations), LOSS employs differential evolution [68], leveraging its strong global search capability and parallelism to efficiently solve the nonlinear, non-convex scheduling problem. Given fixed hardware, bandwidth emerges as the dominant variable affecting local operation partitioning. This allows the problem to be addressed within a constrained search space, eliminating the exploration overhead of reinforcement learning [38].

While solving takes tens of minutes per model, it is performed offline within the Scheduling Optimization stage and does not impact runtime inference. Scheduling plans only need to be updated when hardware configurations change, such as upgrades to the mobile device's computing accelerator or GPU server. Although Intra-DP's system performance is directly influenced by solution quality, and achieving global optimality in non-convex problems remains challenging, future research will focus on refining algorithms to enhance solution quality, ultimately improving scheduling efficiency and overall system performance.

4.4 Adaptive Control Mechanism

To ensure efficient inference under varying network conditions, we assume stable bandwidth during each request, as model inference time (tens to hundreds of milliseconds) is finer-grained than wireless bandwidth fluctuations (a few seconds [65]). The adaptive control mechanism of Intra-DP runs on both the mobile device client and GPU server (Fig. 7 ⑤), with detailed algorithms in Alg.1 and Alg.2. During runtime, Intra-DP estimates real-time bandwidth (line 1 in both algorithms) and selects a suitable scheduling plan. It then computes the designated local operations (line 5 in both algorithms), producing outputs that represent the computation results of these local operations or an empty set if no local computation is performed. Leveraging operation

Algorithm 1: Intra-DP client at runtime stage

Input: Data input for inference *input*; DNN model *model*
Output: The inference result *ret*
Data: $Input(i)$,$Output(i)$: input and output of layer i; x_M^b, x_R^b: schedule plan under the b bandwidth.

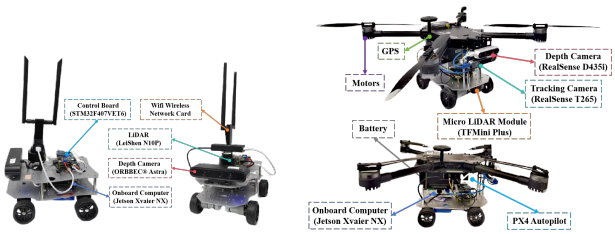
1 $b = TestBandwidth()$
2 $Input(0) = input$
3 **foreach** *layer u in model* **do**
4 **if** $Input(u) \neq \emptyset$ *and* $x_M^b(u) \neq \emptyset$ **then**
5 $output(u) = compute(Input(u), x_M^b(u))$
6 **end**
7 **foreach** *edge* $e = (u, v) \in E$ **do**
8 **if** $x_M^b(v) - child(x_M^b(u)) \neq \emptyset$ **then**
9 $Input(v) = combine(Ouput(u), ReceiveFromServer())$
10 **end**
11 **if** $x_R^b(v) - child(x_R^b(u)) \neq \emptyset$ **then**
12 $SendToServer(Ouput(u), x_R^b(v) - child(x_R^b(u)))$
13 **end**
14 **end**
15 **end**
16 $ret = Output(t)$
17 **return** ret

Algorithm 2: Intra-DP server at runtime stage

1 $b = ReceiveFromClient()$
2 $Input(0) = \emptyset$
3 **foreach** *layer u in model* **do**
4 **if** $Input(u) \neq \emptyset$ *and* $x_R^b(u) \neq \emptyset$ **then**
5 $output(u) = compute(Input(u), x_R^b(u))$
6 **end**
7 **foreach** *edge* $e = (u, v) \in E$ **do**
8 **if** $x_M^b(v) - child(x_M^b(u)) \neq \emptyset$ **then**
9 $SendToClient(Ouput(u), x_R^b(v) - child(x_R^b(u)))$
10 **end**
11 **if** $x_R^b(v) - child(x_R^b(u)) \neq \emptyset$ **then**
12 $Input(v) = combine(Ouput(u), ReceiveFromClient())$
13 **end**
14 **end**
15 **end**

batching from LOSS's computational efficiency constraints, Intra-DP merges all operations within the same operator ($x_M^b(u)$) into a single kernel launch. This launch executes once the required inputs are ready, prioritizing inference correctness over strict adherence to the scheduled execution time ($s_M(u)$). The associated blocking overhead, primarily from tensor combining, is optimized in LOSS.

If the scheduling plan requires data transmission, the server sends the computed outputs to the client (line 9 in Alg.2), which integrates them with its results for subsequent operators (line 9 in Alg.1). Similarly, if data needs to be sent from client to server, the client transmits the outputs (line 12 in Alg.1), and the server integrates them accordingly (line 12 in Alg.2). Otherwise, the outputs of the current operator directly serve as inputs for subsequent operators. Once all operations are completed, the final inference result remains on the client, ready for upper-layer applications (line 17 in Alg.1).



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

Figure 9: The detailed composition of the robot platforms.

	inference	communication	standby
Energy (Watt)	13.35	4.25	4.04

Table 2: Power draw (Watt) of our robot in different states.

5 Implementation

In this section, we first elaborate on the implementation of Intra-DP, and we then introduce the experiment setup.

5.1 System Implementation

5.1.1 Software We implement Intra-DP using Python and PyTorch [67], integrating it seamlessly into existing deep learning workflows. The implementation hooks into the model's forward method, where the first forward pass profiles the model using PyTorch's default profiler and execution schedule. Based on this profiling data, Intra-DP then intercepts and parallelizes all subsequent forward calls according to the optimized execution plan. With a lightweight integration requiring only three lines of code, Intra-DP ensures ease of deployment in existing applications.

5.1.2 Hardware Testbed We evaluate Intra-DP on two customized robotic platforms: a four-wheeled ground robot (Fig. 9a) and an air-ground hybrid robot (Fig. 9b). Each robot is equipped with a Jetson Xavier NX (8GB) [62], enabling onboard model inference with local computation. The system runs Ubuntu 20.04 with ROS Noetic and uses a dual-band USB network adapter (MediaTek MT76x2U) for wireless communication. Detailed hardware and sensor configurations are shown in Fig. 9. The GPU server is a PC equipped with an Intel i7-7700K CPU and an NVIDIA GeForce RTX 3080 GPU. It connects to the robots via Wi-Fi 6 on an 80MHz channel at 5GHz. Tab. 2 summarizes the robots' on-board energy consumption (excluding motor power) in different states: inference (full GPU utilization, including CPU/GPU power), communication (communication with the GPU server, including wireless network card energy consumption). Each

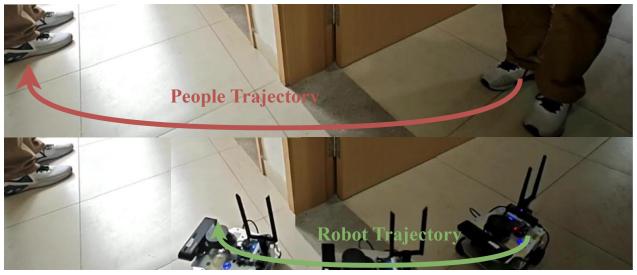


Figure 10: Kapao [54], a real-time people-tracking application on our four-wheeled robot with a CNN-based human keypoint detection model.

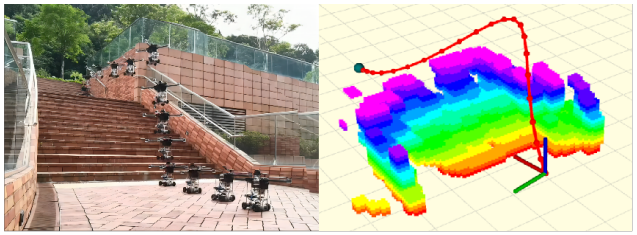


Figure 11: AGRNav [78], an autonomous navigation application on our air-ground robot with a CNN-based 3D semantic scene completion model.

Jetson Xavier NX is powered by a 21.6Wh battery, sustaining up to 1.6 hours of continuous model inference.

5.2 Experiment Setup

5.2.1 Task We evaluated two typical real-world robotic applications on our testbed: Kapao [54] (Fig 10) and AGRNav [78] (Fig 11). We also evaluated several models common to mobile devices with their implementation from Torchvision [52] on a larger scale to further corroborate our observations and findings: VGGNet [73], ConvNeXt [80], ResNet [75], DenseNet [32].

5.2.2 Emulation Environments We evaluated two real-world environments: indoors (robots move in our laboratory with desks and separators interfering with wireless signals) and outdoors (robots move in our campus garden with trees and bushes interfering with wireless signals, resulting in lower bandwidth). The corresponding bandwidths between the robot and the GPU server in indoors and outdoors scenarios are shown in Fig. 4.

5.2.3 Baselines To comprehensively evaluate the performance of Intra-DP, we conducted comparative experiments against several baseline approaches:

- **Device-only inference ("Device-only")**: A conventional setup where the entire model is deployed and executed on the mobile device.
- **Server-only inference ("Server-only")**: A cloud-based approach in which the full model runs on a GPU server. While both Device-only and Server-only represent extreme cases of layer partitioning, we evaluate them separately to highlight the benefits of other baselines in terms of inference latency and energy efficiency.
- **DSCCS** [43]: A state-of-the-art layer partitioning method designed for inference acceleration. To ensure a fair comparison and eliminate the influence of parameter variations, we enhance DSCCS with Intra-DP's adaptive control mechanism, allowing it to dynamically adjust to wireless network fluctuations.
- **SPSO-GA** [14]: An advanced layer partitioning method that optimizes energy consumption while satisfying timing constraints. We configure SPSO-GA with a 1 Hz deadline constraint, corresponding to the minimum frequency required for effective robotic motion control. As with DSCCS, we integrate Intra-DP's adaptive control mechanism into SPSO-GA to allow real-time adaptation to network variations.

For consistency and accurate performance comparison, all systems are implemented without model compression, transmitting raw model activations to maintain accuracy. The Profiling and Scheduling Optimization stages of Intra-DP and all baselines run offline in advance.

6 Performance Evaluation

In this section, we evaluate the performance of Intra-DP from three research aspects: i) comprehensive comparisons with baseline systems to demonstrate the superiority of Intra-DP in terms of inference time and energy consumption for real-world mobile applications; ii) investigating the LOSS of Intra-DP through micro-event analysis; iii) sensitivity Studies to analyze system performance under varying network bandwidth variations, model structures, and equipment computational powers.

6.1 Superiority of Intra-DP

6.1.1 Inference Time Fig. 12 demonstrates the superior performance of Intra-DP compared to four baseline methods across various tasks and environments, achieving significant speedup in inference time (up to 50% for indoors and up to 48% for outdoors). Specifically, Intra-DP outperforms the closest baseline DSCCS by 8% to 26% indoors and 8% to 22% outdoors. These improvements primarily result from LOP, which enables parallel execution and overlaps computation with transmission across local operations within the same inference request, and LOSS, which optimizes scheduling under varying network conditions. Notably, all offloading-based methods, including server-only, two layer partitioning methods, and Intra-DP, exhibit greater standard deviations and longer inference times outdoors due to severe bandwidth fluctuations, as shown in Fig. 4. Although the performance gain is less pronounced on DenseNet, this is analyzed in our sensitivity study in Sec. 6.3.2. Additionally, a micro-event analysis in Sec. 6.2 further explains the efficiency of Intra-DP in inference.

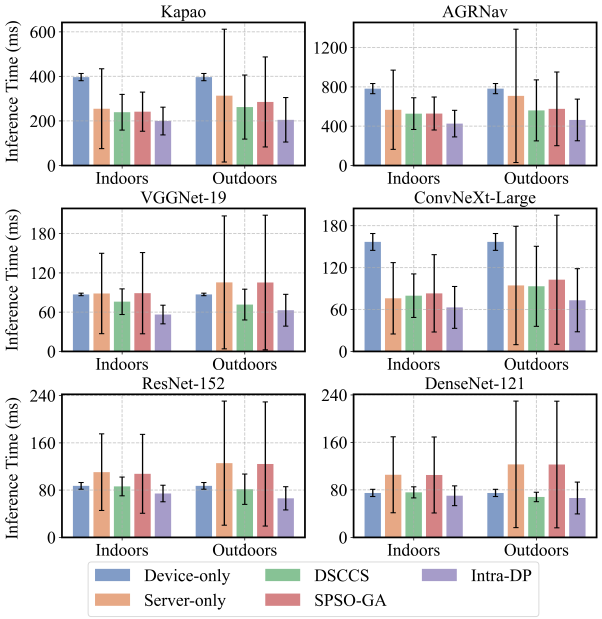


Figure 12: Inference time for different models across various environments and systems.

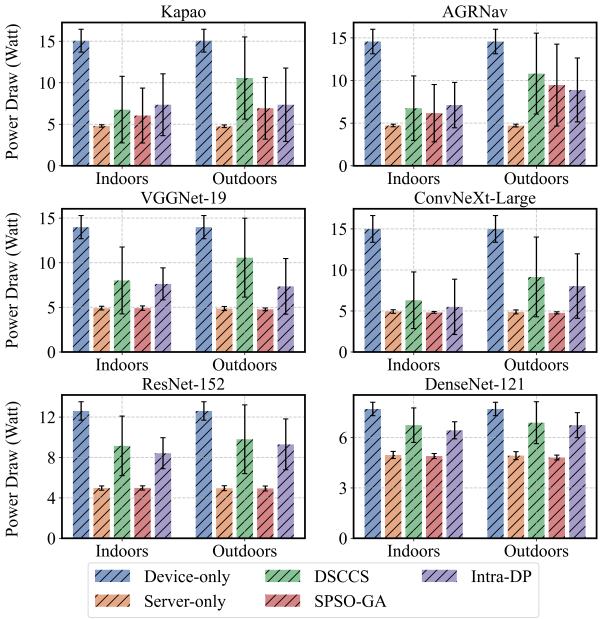


Figure 13: Power draw for different models across various environments and systems.

6.1.2 Energy Consumption Fig. 13 presents the power draw across different methods and scenarios. The device-only approach consumes the most energy due to the intensive computational load on robots, while the server-only approach achieves the lowest energy consumption by offloading all computations to the GPU server. Among the partial-offloading methods, SPSO-GA demonstrates better energy efficiency than DSCCS due to its energy-oriented optimization, whereas Intra-DP exhibits slightly higher energy consumption due to occasional re-computation of local operations. The power draw between Intra-DP and DSCCS remains nearly the same, indicating that the computation-communication overlap in Intra-DP does not introduce significant additional energy costs. Tab. 2 further supports this conclusion, showing that robots continue consuming 95% of their active power even when idle, mainly due to static energy consumption in components such as the CPU, GPU, and memory [37]. In contrast, the wireless network card consumes only 0.21W during transmission compared to 13.35W during computation.

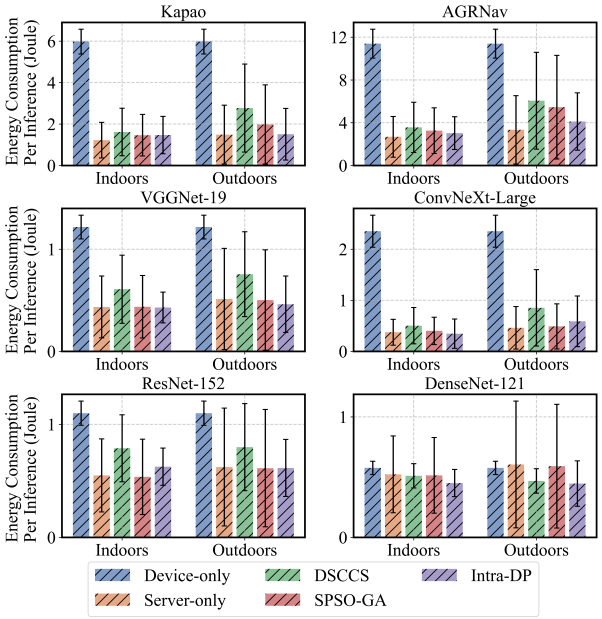


Figure 14: Energy consumption per inference request for different models across various environments and systems.

Fig. 14 presents the energy consumption per inference request across different methods and scenarios. Although Intra-DP exhibits relatively higher power draw in Fig. 13, it achieves significantly lower energy consumption per inference request across all models and environments, reducing energy usage by up to 75% indoors and up to 72% outdoors due to its fast inference time. Notably, compared to the most energy-efficient server-only approach, Intra-DP increases energy consumption per inference request by at most 20% while reducing inference time by up to 42% and greatly reduced long tail delays.

6.2 Micro-Event Analysis

In this section, we presents a detailed investigation of Intra-DP's superior inference speed through micro-event analysis, focusing on the effectiveness of LOSS compared to baseline methods.

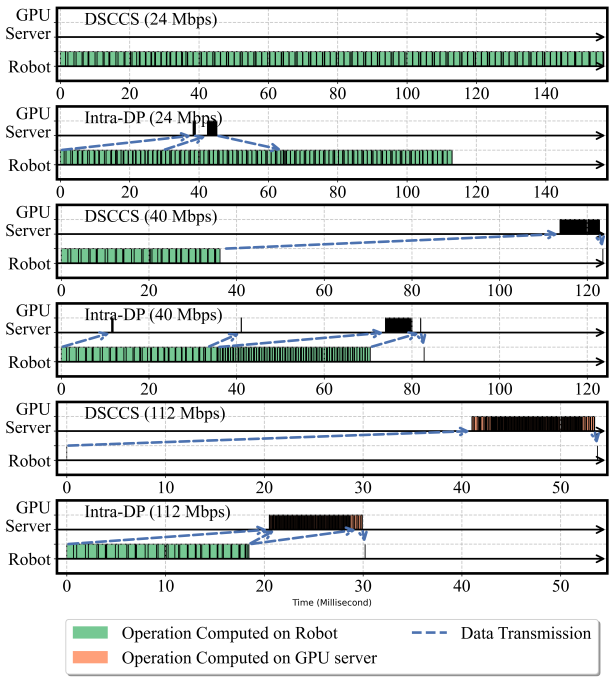


Figure 15: Snapshots of schedule plan of Intra-DP and baseline during runtime under various network bandwidth.

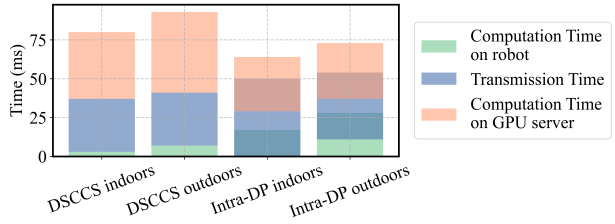


Figure 16: Breakdown of each phase of the inference process.

6.2.1 Detailed Schedule Plan To quantify the performance advantages of Intra-DP, we conducted a comparative analysis with DSCCS by recording real-time scheduling plans for ConvNeXt under varying bandwidth conditions, as shown in Fig. 15. The primary performance gain of Intra-DP stems from the parallel execution of local operations, enabling concurrent computation on the robot, data transmission, and computation on the GPU server. With the integration of Intra-DP's adaptive control mechanism, DSCCS can dynamically select the optimal layer partitioning plan based on real-time network bandwidth. Under low-bandwidth conditions, DSCCS shifts more computation to the device, trading increased energy consumption for reduced tail latency. However, while DSCCS degrades to device-only, Intra-DP maintains its distributed execution capability by leveraging an expanded parallel scheduling space enabled by LOP, requiring less bandwidth for GPU server acceleration. Conversely, in high-bandwidth scenarios, DSCCS transitions to server-only, whereas Intra-DP leverages LOSS to optimize network and GPU server utilization for faster inference.

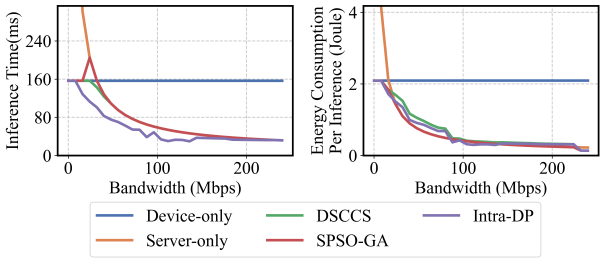


Figure 17: Performance comparison of Intra-DP and baselines under different network bandwidth conditions.

6.2.2 Breakdown Next, we provide a detailed breakdown of each phase of the inference process throughout the ConvNeXt experiment, as illustrated in Fig. 16. In DSCCS, transmission time accounts for up to 44.2% of the total inference time, underscoring the significant transmission bottlenecks encountered by existing layer partitioning methods, even when employing state-of-the-art strategies. Unlike layer partitioning methods, which execute each phase sequentially, Intra-DP enables the transmission phase, the robot computation phase, and the GPU server computation phase to run in parallel, resulting in faster inference times. Although Intra-DP introduces additional communication overhead due to the fine-grained transmission of local operators, it effectively reduces overall completion time through computation-communication overlap and early data transfer initiation, enabling it to outperform traditional layer partitioning methods, as demonstrated in Fig. 15.

6.3 Sensitivity Studies

6.3.1 Network Bandwidth To systematically evaluate Intra-DP's performance under varying bandwidth conditions in MEC environments, we leveraged tc [9] over wired Ethernet to create controlled test scenarios, enabling comprehensive comparisons with baseline methods. As shown in Fig. 17, Intra-DP maintains superior performance across a wider range of network conditions, thanks to the expanded scheduling space enabled by LOP. When bandwidth is extremely low (near 0 Mbps), Intra-DP also degrades to device-only; however, its threshold bandwidth for degradation is significantly lower than that of DSCCS. Similarly, under extremely high bandwidth conditions, Intra-DP degrades to server-only, but its degradation threshold remains much higher than DSCCS. This highlights Intra-DP's extensive parallel optimization space, enabling more efficient execution across diverse network conditions. Notably, the bandwidth thresholds at which layer partitioning methods and Intra-DP transition between device-only and server-only depend on model architecture and device computational power. Furthermore, the performance gain of Intra-DP in Fig. 12 and Fig. 14 is smaller than in Fig. 17 due to inaccurate bandwidth estimation during runtime.

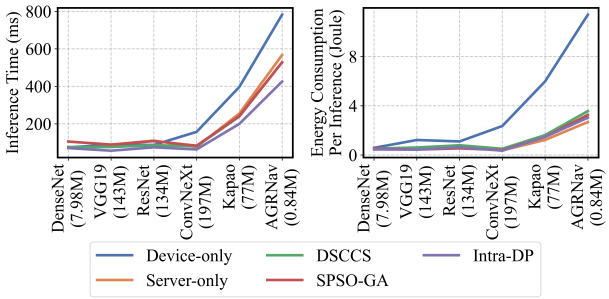


Figure 18: Performance comparison of Intra-DP and baselines with varying model structures.

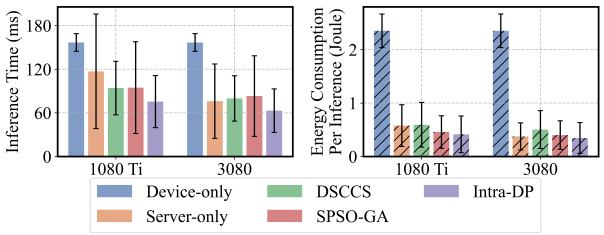


Figure 19: Performance comparison of Intra-DP and baselines on GPU servers with varying computational power.

6.3.2 Model structure As shown in Fig. 18, offloading methods (except device-only) achieved greater performance gains for models with higher computational demands but performed worse than device-only on DenseNet. This indicates that offloading is more beneficial for computationally intensive models, while models with lower computational demands may suffer from communication overhead, especially in bandwidth-constrained MEC systems. Furthermore, although Intra-DP consistently outperformed other baselines, its performance gains over them were relatively smaller for models with fewer parameters. This observation can be attributed to the fact that Intra-DP's performance improvements are primarily achieved through the parallel execution of local operations, making its effectiveness dependent on the model's size and structure. When a model has fewer parameters, the number of local operations available for parallel execution is reduced, thereby limiting Intra-DP's optimization potential for enhancing performance.

6.3.3 Computational Power To assess Intra-DP's performance under varying computational power disparities between the robot and the GPU server, we further evaluate Intra-DP using a GPU server equipped with an NVIDIA GTX 1080 Ti. As shown in Fig. 19, while offloading methods benefit from a greater power differential, Intra-DP achieves the most significant gains by optimizing GPU server utilization through parallel execution of local operations. It is also important to note that the mobile device used in our experiments has substantially higher computational power than typical mobile devices, such as smartphones, as illustrated in Fig. 2a.

7 Related work and discussion

Limited bandwidth. Due to hardware limitations, we evaluated Intra-DP's performance only on commonly available Wi-Fi networks in robotic environments, rather than a broader set of wireless technologies such as 6G [12] or WiMAX [4]. While different wireless networks vary in throughput and range due to distinct transmission protocols, they all suffer from weak connections caused by intermittent jitter and packet loss [47], leading to bandwidth fluctuations in practice (Fig. 4). In these conditions, Intra-DP proves beneficial. When GPU servers are deployed in commercial cloud environments, network congestion and routing inefficiencies further restrict available bandwidth [60], making Intra-DP even more advantageous. Furthermore, as models scale and GPU computational power increases, Intra-DP's advantage continues to grow, further highlighting its relevance.

Inference Request Scheduling. It schedules multiple DNN inference requests to optimize overall latency and energy consumption while leveraging existing collaborative inference methods for individual executions. Various decision algorithms, such as request batching [94], urgency-based prioritization [45, 51], and deep reinforcement learning-based control [6, 19, 20], are employed to determine execution locations and timing. Intra-DP enhances these approaches by introducing a higher-performance collaborative inference method for MEC, and these scheduling strategies can be seamlessly integrated into Intra-DP.

Model Compression. Quantization and model distillation are two most common model compression techniques for mobile devices. Quantization [17, 24, 27] reduces precision of model weights and activations to lower computation costs, while model distillation [28, 46, 79] trains a smaller model to mimic a larger one with fewer resources. Unlike these methods, which trade accuracy for efficiency, collaborative inference accelerates computation by intelligently distributing workloads without compromising accuracy. Building on the significant performance gains already achieved by Intra-DP, our future work will explore how sacrificing some accuracy through techniques such as quantization or early-exit policies [39] can further enhance its efficiency.

8 Conclusion

In this paper, we have proposed Intra-DP, a high-performance collaborative inference system optimized for MEC networks. By leveraging LOP for fine-grained parallel execution and LOSS for optimized scheduling, Intra-DP dramatically reduces transmission overhead in MEC environments through computation-communication overlap across local operations. We envision that the fast and energy-efficient inference of Intra-DP will enable diverse tasks on real-world mobile devices, driving broader adoption of advanced machine learning in mobile applications.

References

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

[2] Nasir Abbas, Yan Zhang, Amir Taherkordi, and Tor Skeie. 2017. Mobile edge computing: A survey. *IEEE Internet of Things Journal* 5, 1 (2017), 450–465.

[3] 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.

[4] Syed A. Ahson and Mohammad Ilyas. 2008. *WiMAX: Standards and Security*. CRC Press. doi:10.1201/9781315219912

[5] Suzan Almutairi and Ahmed Barnawi. 2023. Securing DNN for smart vehicles: An overview of adversarial attacks, defenses, and frameworks. *Journal of Engineering and Applied Science* 70, 1 (2023), 16.

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

[7] AnandTech. 2023. AnandTech. https://www.anandtech.com/show/17587/iphone-14-battery-life-test/4.

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

[9] Joseph D Beshay, Andrea Francini, and Ravi Prakash. 2015. On the fidelity of single-machine network emulation in linux. In *2015 IEEE 23rd International Symposium on Modeling, Analysis, and Simulation of Computer and Telecommunication Systems*. IEEE, 19–22.

[10] Jonas Beuchert and Alex Rogers. 2021. SnapperGPS: Algorithms for Energy-Efficient Low-Cost Location Estimation Using GNSS Signal Snapshots. In *Proceedings of the 19th ACM Conference on Embedded Networked Sensor Systems* (Coimbra, Portugal) *(SenSys '21)*. Association for Computing Machinery, New York, NY, USA, 165–177. doi:10.1145/3485730.3485931

[11] 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.

[12] Robin Chataut, Mary Nankya, and Robert Akl. 2024. 6G networks and the AI revolution—Exploring technologies, applications, and emerging challenges. *Sensors* 24, 6 (2024), 1888.

[13] Nidhi Chawla and Surjeet Dalal. 2021. Edge AI with Wearable IoT: A Review on Leveraging Edge Intelligence in Wearables for Smart Healthcare. *Green Internet of Things for Smart Cities* (2021), 205–231.

[14] 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.

[15] 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.

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

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

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

[19] 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.

[20] 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.

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

[22] Luyao Gao, Jianchun Liu, Hongli Xu, Sun Xu, Qianpiao Ma, and Liusheng Huang. 2024. Accelerating End-Cloud Collaborative Inference via Near Bubble-free Pipeline Optimization. *arXiv preprint arXiv:2501.12388* (2024).

[23] Himanshu Gauttam, Kiran Kumar Pattanaik, Saumya Bhadauria, Garima Nain, and Putta Bhanu Prakash. 2023. An efficient DNN splitting scheme for edge-AI enabled smart manufacturing. *Journal of Industrial Information Integration* 34 (2023), 100481.

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

[25] Nicolò Ghielmetti, Vladimir Loncar, Maurizio Pierini, Marcel Roed, Sioni Summers, Thea Aarrestad, Christoffer Petersson, Hampus Linander, Jennifer Ngadiuba, Kelvin Lin, et al. 2022. Real-time semantic segmentation on FPGAs for autonomous vehicles with hls4ml. *Machine Learning: Science and Technology* 3, 4 (2022), 045011.

[26] Anurag Ghosh, Srinivasan Iyengar, Stephen Lee, Anuj Rathore, and Venkata N Padmanabhan. 2023. React: Streaming video analytics on the edge with asynchronous cloud support. In *Proceedings of the 8th ACM/IEEE Conference on Internet of Things Design and Implementation*. 222–235.

[27] 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.

[28] 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.

[29] 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.

[30] Chenghao Hu and Baochun Li. 2024. When the Edge Meets Transformers: Distributed Inference with Transformer Models. In *2024 IEEE 44th International Conference on Distributed Computing Systems (ICDCS)*. 82–92. doi:10.1109/ICDCS60910.2024.00017

[31] 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.

[32] Gao Huang, Zhuang Liu, Laurens van der Maaten, and Kilian Q. Weinberger. 2018. Densely Connected Convolutional Networks. arXiv:1608.06993 [cs.CV]

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

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

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

[36] 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.

[37] 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.

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

[39] Stefanos Laskaridis, Stylianos I. Venieris, Mario Almeida, Ilias Leontiadis, and Nicholas D. Lane. 2020. SPINN: synergistic progressive inference of neural networks over device and cloud. In *Proceedings of the 26th Annual International Conference on Mobile Computing and Networking* (London, United Kingdom) *(MobiCom '20)*. Association for Computing Machinery, New York, NY, USA, Article 37, 15 pages. doi:10.1145/3372224.3419194

[40] 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.

[41] 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.

[42] 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.

[43] 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).

[44] 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.

[45] Li Lin, Xiaofei Liao, Hai Jin, and Peng Li. 2019. Computation offloading toward edge computing. *Proc. IEEE* 107, 8 (2019), 1584–1607.

[46] 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.

[47] Yuxiang Lin, Yi Gao, and Wei Dong. 2022. Bandwidth Prediction for 5G Cellular Networks. In *2022 IEEE/ACM 30th International Symposium on Quality of Service (IWQoS)*. 1–10. doi:10.1109/IWQoS54832.2022.9812912

[48] 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.

[49] 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.

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

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

[52] Sébastien Marcel and Yann Rodriguez. 2010. Torchvision the machine-vision package of torch. In *Proceedings of the 18th ACM International Conference on Multimedia* (Firenze, Italy) *(MM '10)*. Association for Computing Machinery, New York, NY, USA, 1485–1488. doi:10.1145/1873951.1874254

[53] 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.

[54] 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.

[55] MLPerf. 2023. MLPerf. https://mlcommons.org/working-groups/benchmarks/mobile/.

[56] 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.

[57] 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.

[58] Zijie Ning, Maarten Vandersteegen, Kristof Van Beeck, Toon Goedemé, and Patrick Vandewalle. 2024. Power Consumption Benchmark for Embedded AI Inference.

[59] Hengchang Nong, Mingzhe Jin, Chengcheng Pan, Hongyu Zhou, Chao Zhang, Xuemei Pan, Yuren Chen, Xiangning Wei, Yeping Lu, Kaixiao Zhao, et al. 2024. Intelligent sensing technologies based on flexible wearable sensors: A review. *IEEE Sensors Journal* (2024).

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

[61] NVIDIA. 2024. InfiniBand Networking Solutions. https://www.nvidia.com/en-us/networking/products/infiniband/.

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

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

[64] Raspberry Pi. 2022. Raspberry Pi. https://www.raspberrypi.com/documentation/computers/configuration.html#power-consumption.

[65] Reza Poorzare and Anna Calveras Augé. 2021. How Sufficient is TCP When Deployed in 5G mmWave Networks Over the Urban Deployment? *IEEE Access* 9 (2021), 36342–36355. doi:10.1109/ACCESS.2021.3063623

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

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

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

[69] Ratheesh Ravindran, Michael J Santora, and Mohsin M Jamali. 2020. Multi-object detection and tracking, based on DNN, for autonomous vehicles: A review. *IEEE Sensors Journal* 21, 5 (2020), 5668–5677.

[70] 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.

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

[72] Nir Shlezinger and Ivan V. Bajić. 2022. Collaborative Inference for AI-Empowered IoT Devices. *IEEE Internet of Things Magazine* 5, 4 (2022), 92–98. doi:10.1109/IOTM.001.2200152

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

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

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

[76] Ye Tian, Kai Xu, and Nirwan Ansari. 2005. TCP in wireless environments: problems and solutions. *IEEE Communications Magazine* 43, 3 (2005), S27–S32.

[77] Haoran Wang, Lei Wang, Haobo Xu, Ying Wang, Yuming Li, and Yinhe Han. 2024. PrimePar: Efficient Spatial-temporal Tensor Partitioning for Large Transformer Model Training. In *Proceedings of the 29th ACM International Conference on Architectural Support for Programming Languages and Operating Systems, Volume 3*. 801–817.

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

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

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

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

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

[83] Yecheng Xiang and Hyoseung Kim. 2019. Pipelined data-parallel CPU/GPU scheduling for multi-DNN real-time inference. In *2019 IEEE Real-Time Systems Symposium (RTSS)*. IEEE, 392–405.

[84] Saining Xie, Ross Girshick, Piotr Dollár, Zhuowen Tu, and Kaiming He. 2017. Aggregated Residual Transformations for Deep Neural Networks. arXiv:1611.05431 [cs.CV] https://arxiv.org/abs/1611.05431

[85] Jiawen Xu, Matthias Kovatsch, Denny Mattern, Filippo Mazza, Marko Harasic, Adrian Paschke, and Sergio Lucia. 2022. A review on AI for smart manufacturing: Deep learning challenges and solutions. *Applied Sciences* 12, 16 (2022), 8239.

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

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

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

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

[90] Jun Yao, Salil S. Kanhere, and Mahbub Hassan. 2008. An empirical study of bandwidth predictability in mobile computing. In *Proceedings of the Third ACM International Workshop on Wireless Network Testbeds, Experimental Evaluation and Characterization* (San Francisco, California, USA) *(WiNTECH '08)*. Association for Computing Machinery, New York, NY, USA, 11–18. doi:10.1145/1410077.1410081

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

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

[93] Liekang Zeng, Xu Chen, Zhi Zhou, Lei Yang, and Junshan Zhang. 2021. CoEdge: Cooperative DNN Inference With Adaptive Workload Partitioning Over Heterogeneous Edge Devices. *IEEE/ACM Transactions on Networking* 29, 2 (2021), 595–608. doi:10.1109/TNET.2020.3042320

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

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

[96] Shuai Zhang, Sheng Zhang, Zhuzhong Qian, Jie Wu, Yibo Jin, and Sanglu Lu. 2021. DeepSlicing: Collaborative and Adaptive CNN Inference With Low Latency. *IEEE Transactions on Parallel and Distributed Systems* 32, 9 (2021), 2175–2187. doi:10.1109/TPDS.2021.3058532

[97] Pai Zheng, Honghui Wang, Zhiqian Sang, Ray Y Zhong, Yongkui Liu, Chao Liu, Khamdi Mubarok, Shiqiang Yu, and Xun Xu. 2018. Smart manufacturing systems for Industry 4.0: Conceptual framework, scenarios, and future perspectives. *Frontiers of Mechanical Engineering* 13 (2018), 137–150.

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