Work-in-Progress: Real-Time Deep Neural Inference on Resource-Constrained Edge Devices

Md Tasnim Farhan Fatin and Monowar Hasan
School of Electrical Engineering and Computer Science, Washington State University, USA
Email: m.fatin@wsu.edu, monowar.hasan@wsu.edu

***Abstract*—Deep neural networks (DNNs) are now central to perception and control in time-critical cyber-physical systems, yet many mid- and low-tier edge platforms (e.g., single-board computers without GPUs and most microcontrollers) cannot host full models or meet real-time constraints. Cooperative inference by offloading workload across nearby devices improves feasibility, but most existing schemes do not analyze end-to-end timing and are thus not suitable for real-time applications. This Work-in-Progress paper studies deadline-aware distributed DNN inference with an asynchronous execution semantics: a device proceeds as soon as its required partial outputs are available, without global synchronization. We develop a time-aware optimization model that minimizes response time subject to deadline constraints. We also outline our ongoing research efforts and future extensions of the proposed work.**

***Index Terms*—Distributed Inference, Real-Time Systems, DNN.**

I. Introduction

The proliferation of Internet-of-Things (IoT) and edge devices has enabled new classes of cyber-physical applications in domains such as robotics, digital agriculture, industrial automation, health care, and autonomous vehicles. Many of these systems rely on deep neural networks (DNNs) for perception and control tasks such as object detection, anomaly recognition, and scene understanding [1], [2]. However, executing modern DNN models on embedded platforms is challenging because these devices operate under severe compute, memory, and energy constraints. For instance, a Raspberry Pi 4 [3] or NVIDIA Jetson Nano [4] often requires several seconds to process a single image using VGG-16 [5], while microcontrollers such as STM32 [6] or ESP32 [7] cannot even host the complete model due to limited memory capacity. Such delays are unacceptable in real-time and safety-critical applications, where inference results must be produced within strict deadlines to maintain safe system operation. Further, deploying newer learning-enabled workloads is especially difficult on legacy or previously installed systems where hardware upgrades are not possible.

Cloud offloading [8], [9] can mitigate local resource limitations but introduces variable latency and security concerns, making it unsuitable for time-sensitive tasks. As an alternative, distributed or collaborative inference allows multiple nearby edge devices to share the workload of executing a DNN [10]. In this approach, layers are partitioned among devices so that each performs part of the computation. Existing frameworks such as DeepThings [11], CoEdge [12], and EdgeFlow [13] have investigated distributed inference using layer-wise or spatial partitioning strategies. However, most adopt "synchronous" execution, where every device must finish its assigned portion of a layer before the next layer begins. This global synchronization causes idle times and increases latency—especially in heterogeneous systems where device speeds and communication bandwidths vary significantly—and, more importantly, these approaches overlook the real-time aspect of inference, where task deadlines must be explicitly analyzed and guaranteed.

In contrast to prior work, we study *asynchronous* distributed inference for resource-constrained real-time edge systems, where each device can begin its computation for the next layer as soon as the required inputs are available, without waiting for all other nodes to finish. We make the following contributions:

- We propose a timing-aware model for distributed DNN inference on low-power embedded platforms such as microcontrollers, addressing an area largely unexplored from a real-time perspective.
- We formulate an optimization problem to determine layer partitioning that minimizes end-to-end response time while respecting real-time deadlines.
- We show that exploiting asynchronous execution—where devices proceed as soon as dependent data are ready—can significantly reduce inference delay compared to synchronous methods.

Preliminary formulations show that allowing asynchronous execution can substantially reduce inference delays compared to synchronous approach considered in existing (time-unaware) work [13]. Our initial work-in-progress lays the foundation for deadline-aware cooperative inference in large-scale, heterogeneous edge environments such as distributed robotic swarms and sensor networks.

II. Problem Overview

Consider a case where we want to execute a time-critical DNN inference in resource-constrained edge devices. An edge device is not capable of executing the entire DNN model. Hence, the DNN inference task Γ is *partitioned* and distributed across N devices denoted by the set $\Pi = \{\pi_1, \pi_2, \cdots, \pi_N\}$. We assume that the DNN task has a temporal constraint, i.e.,

This research is partly supported by the U.S. National Science Foundation Award 2345653. Any findings, opinions, recommendations, or conclusions expressed in the paper are those of the authors and do not necessarily reflect the sponsor's views.

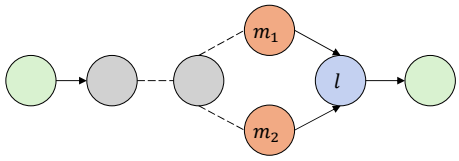

Fig. 1. Representation of DNN as a DAG where each node is a neural network layer. Green nodes represent the input and output of the model. Dotted lines and gray nodes represent multiple intermediate layers. Layers typically have data dependency. For instance, layer l requires the results from the two previous layers, m_1 and m_2.

it must complete within a designer-given timing bound D (called inference *deadline*). Our objective here is to derive *end-to-end response times* for this partitioned DNN execution and determine whether distributed inference can meet the deadline constraints.

The DNN workload is represented by a directed acyclic graph (DAG), where each node in the graph is a layer of the DNN, as shown in Fig. 1. We represent the layers of the DNN task Γ by L layers denoted by the set $\mathcal{L}$ and the connections between layers by $\mathcal{E}$. For a layer $l \in \mathcal{L}$, the set of layers that are required to calculate layer l is $\mathcal{M}(l) = m : (m, l) \in \mathcal{E}$ where $\mathcal{M}(l)$ is the predecessor set of l. Let $\mathcal{P}_{l,i}(m) \subseteq \Pi$ be the set of required senders for layer l, device π_i from layer m. Besides, we define the dependency set based on the predecessor set and required senders as $\mathcal{R}_{l,i} = \{(m, j) : m \in \mathcal{M}(l), j \in \mathcal{P}_{l,i}(m)\}$. This dependency set imposes an execution order constraint for distributed DNN inference; that is $\forall (m, j) \in \mathcal{R}_{l,i}$, device π_i cannot begin computing its assigned portion of layer l until device π_j has completed both computation and transmission of the required portion of layer m.

As the inference process is computationally intensive, it is partitioned both spatially and temporally; that is, the computation proceeds layer by layer, with each layer distributed across N devices. We want to determine the partitions for each layer to assign to devices so that each device can perform small, layer-wise computations across the entire DNN inference chain. As we distribute the DNN inference task (e.g., layers) across N devices, the partitions running on a device may differ from one layer to the next. The data exchange between devices can be described as a bipartite graph where edges indicate which outputs from one device must be sent to which device for the next layer. During execution, once a device π_i has finished computing the parts of a layer l that are required by another device π_j, π_i transmits the output to π_j. As soon as π_j receives the necessary components of the layer it is responsible for processing, it begins computing without waiting for all other devices to finish. The DNN tasks may also require modification to ensure that each device has sufficient context for correct computation. For example, during convolution operations, intermediate layers may need to store overlapping rows from neighboring devices.

We assume a convolutional neural network, where the output of each layer is two-dimensional (height × width). For each layer l, the two-dimensional output feature row denoted by $\{1, \cdots, H_l\}$. A device π_j executes a portion $[x^l_{i-1}+1, x^l_i]$ of a layer l 's output feature such that $1 \le x^l_1 \le \cdots \le x^l_N \le H_l$. We assume that the execution time of a layer on a device is proportional to the size of its output. The execution time is linear with the size of the input and output for each layer and shows different linear trends across different layers [14], [15]. For worst-case real-time analysis, the worst-case execution time (WCET) on device π_i for layer l with chunk (partition) size $\Delta^l_i = x^l_i - x^l_{i-1}$ (and $x^l_0 = 0$) is given by $C_{l,i} = a_{l,i}\Delta^l_i + b_{l,i}$, where $a_{l,i}$ and $b_{l,i}$ are weighted constants for calculating the execution time. These coefficients depend on both the layer's computation and the device's processing power.

To perform computations for layer l, suppose device π_i needs information of the preceding layer m from device π_j. As an illustration, let π_1 is executing portion $[1, x^l_1]$ of output feature, as shown in Fig. 2. To perform this operation, π_1 needs information covering the range $[s^l_1, e^l_1]$ from the previous layer m (marked blue). The required data is divided among devices π_1 and π_2. For instance, in Fig. 2, $\mathcal{P}_{l,1}(m) = \{\pi_1, \pi_2\}$. As the execution of DNN is modeled as a DAG, a layer l can depend on multiple layers, represented as the precedence set of l or $\mathcal{M}(l)$ (e.g., in Fig. 2, $\mathcal{R}_{l,1} = \{(m, 1), (m, 2)\}$). Hence, π_2 needs to send the overlapped region $p^{l\leftarrow m}_{1\leftarrow 2}$. Let the bandwidth to send information from device j to device i be $B_{i,j}$. The transfer time (over the communication channel) for sending the region $p^{l\leftarrow m}_{i\leftarrow j}$ of layer m for computing layer l from device j to device i is then given by: $X^{l\leftarrow m}_{i\leftarrow j} = p^{l\leftarrow m}_{i\leftarrow j}/B_{i,j}$. If π_i computes portion of layer l from $x^l_{i-1} + 1$ to x^l_i, which requires information of previous layer from s^l_i to e^l_i, then $p^{l\leftarrow m}_{i\leftarrow j} = \max\left(0,\ \min\left(e^l_i, x^m_j\right) - \max\left(s^l_i, x^m_{j-1}\right)\right)$. Let $T_{l,i}$ be the completion time of the portion of layer l assigned to device π_i, which can be formally defined as follows:

$$T_{l,i} = C_{l,i}(\Delta^l_i) + \max_{(m,j)\in\mathcal{R}_{l,i}} (T_{m,j} + X^{l\leftarrow m}_{i\leftarrow j}) \tag{1}$$

The first term in Eq. (1), $C_{l,i}$ represents the execution time of a device for layer l for a chunk size Δ^l_i. The second term denotes the starting time of the execution. The arguments in the max operator capture the arrival times of all required predecessor computations (i.e., the completion time $T_{l,i}$ must consider prior dependent layers).

A. An Illustrative Example

Let us consider two approaches. For the first case, every device must complete its partitioned layer before any other device can begin the next. That is, if m is a preceding layer of l, all devices must complete layer m before any device can start processing layer l. We refer to this as "synchronous" execution.[1] For the latter approach, we do not have such a requirement, and a device can start computation for layer l as soon as it has received the required input from the previous layer m. We call this "asynchronous" execution.

[1]Prior work [13] introduces a synchronous execution approach but does not design it for real-time inference applications.

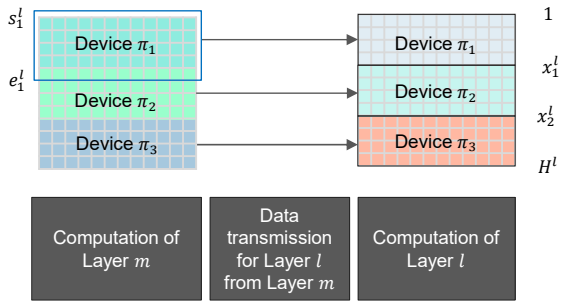

Fig. 2. An illustration of data transmission between devices. Let s_i^l and e_i^l indicate the starting and ending position of required information for computing layer l on π_i (left). The parameters x_1^l and x_2^l indicate the partition of the layer l, and H^l is the height of the layer output (right).

TABLE I
EXAMPLE WORKLOAD.

Layer (l)	Computation Time ($C_{l,i}$)			
	π_1	π_2	π_3	π_4
l_1	11	10	8	9
l_2	9	7	9	10
l_3	9	8	7	9

We now explain the distributed inference problem with a simple toy example. Consider a DNN inference task with 3 convolution layers (l_1, l_2 and l_3) distributed across 4 resource-constrained devices (π_1, π_2, π_3 and π_4). A portion of each layer is executed in parallel on all devices. Each device requires information from its neighboring devices (e.g., neighboring devices of π_2 are π_1 and π_3). For layer 2, we can write the dependency set $\mathcal{R}_{l,i}$ as: $\mathcal{R}_{2,\pi_1} = \{(1,\pi_1),(1,\pi_2)\}$, $\mathcal{R}_{2,\pi_2} = \{(1,\pi_1),(1,\pi_2),(1,\pi_3)\}$, and $\mathcal{R}_{2,\pi_3} = \{(1,\pi_2),(1,\pi_3)\}$. Table I shows the execution time of a layer l on device π_i. Further, let us assume the data transmission delay $X_{i\leftarrow j}^{l\leftarrow m} = 2$ ms when $i \neq j$ and $X_{i\leftarrow j}^{l\leftarrow m} = 0$ when $i = j$.

Table II represents the events and completion times for both cases for this example workload. If we perform distributed inference using the synchronous approach, $T_1 = \max(C_{1,i} + X_{i\leftarrow j}^{2\leftarrow 1}) = 11 + 2 = 13$ ms, $T_2 = T_1 + \max(C_{2,i} + X_{i\leftarrow j}^{3\leftarrow 2}) = 13+10+2 = 25$ ms, and $T_3 = T_2 + \max(C_{3,i}) = 25+9 = 34$ ms. Hence, the end-to-end response time for this DNN inference task is 34 ms.

If we take the asynchronous execution approach, for l_1, the devices π_1, π_1, π_1, and π_4 finish execution at 11 ms, 10 ms, 8 ms, and 9 ms, respectively. After taking into account the transmission time, each device can start executing the partition of l_2 at 12 ms, 13 ms, 12 ms, 10 ms, respectively (calculated using $\max(T_{1,j} + X_{i\leftarrow j}^{2\leftarrow 1}), \forall (1,j) \in \mathcal{R}_{2,i}$). The computation of l_2 finishes at 21 ms, 20 ms, 21 ms, 20 ms respectively (since $T_{2,i} = C_{2,i} + \max(T_{1,j} + X_{i\leftarrow j}^{2\leftarrow 1})$). Finally, each device can start executing the partition of the last layer (l_3) at 22 ms, 23 ms, 22 ms, 23 ms, respectively (obtained by $\max(T_{2,j} + X_{i\leftarrow j}^{3\leftarrow 2}), \forall (2,j) \in \mathcal{R}_{3,i}$). The computation of l_3 finishes at 31 ms, 31 ms, 29 ms, 31 ms, respectively (i.e., $(T_{3,i} = C_{3,i} + \max(T_{2,j} + X_{i\leftarrow j}^{3\leftarrow 2}))$. Thus, the maximum execution time for the inference task in this case is 31 ms. Compared to the synchronous method introduced in earlier work [13], an asynchronous execution can decrease end-to-end execution time by 9.7%, which is critical for real-time inference.

TABLE II
RESPONSE TIME FOR SYNCHRONOUS AND ASYNCHRONOUS INFERENCE.

Event	Approach	Layer Completion Time			
		π_1	π_2	π_3	π_4
L_1 Complete	Synchronous	11	10	8	9
	Asynchronous	11	10	8	9
L_2 Start	Synchronous	13	13	13	13
	Asynchronous	12	13	12	10
L_2 Complete	Synchronous	22	20	22	23
	Asynchronous	21	20	21	20
L_3 Start	Synchronous	25	25	25	25
	Asynchronous	22	23	22	23
L_3 Complete	Synchronous	34	33	32	33
	Asynchronous	31	31	29	31

III. TIME-AWARE DISTRIBUTED INFERENCE

We aim to determine the optimal partition points that minimize the response time of the inference task while ensuring that the timing constraints are satisfied. For instance, consider a case where two dependent layers m and l are distributed among three devices (π_1, π_2, and π_3) and $C_{l,1}$ is the execution time of a portion of layer l, as shown in Fig. 3. Device π_1 needs information from Device π_2 and π_3. We cannot start executing layer l unless π_1 finishes execution of layer m, and also the required information from π_2 and π_3 is received. As we want to reduce the end-to-end response times, our objective is to find the minimum value of the maximum execution time of the devices used to compute the *final* layer. Mathematically, this can be represented as follows:

$$\min(\max(T_{L,i})), \qquad \forall i = \{\pi_1, .., \pi_N\}. \tag{2}$$

As each layer is distributed among N devices, the combined partition will form the complete output feature, i.e.,

$$\sum_{i=1}^{N} \Delta_i^l = H^l, \quad \forall i \in \{\pi_1, \cdots, \pi_N\}. \tag{3}$$

Due to the presence of max and min operators in Eq. (1) and Eq. (2), it becomes a non-convex optimization problem to derive the values of $T_{l,i}$. Note that,

$$T_{l,i} \geq C_{l,i}(\Delta_i^l) + T_{m,j} + X_{i\leftarrow j}^{l\leftarrow m} \quad \forall i \in \{\pi_1, \cdots, \pi_N\}, \forall l \in \{1, \cdots, L\}, \forall (m,j) \in \mathcal{R}_{l,i}. \tag{4}$$

Let us introduce a new variable $T = \max(T_{L,i})$. Then,

$$T \geq T_{L,i} \quad \forall i \in \{\pi_1, \cdots, \pi_N\} \tag{5}$$

The decision variables in our optimization problem include *(a)* partitioned segments of each layer H^l, i.e., Δ_i^l, $\forall i \in \{\pi_1, .., \pi_N\}$, *(b)* completion time of each device for each assigned portion $T_{l,i}$, $\forall i \in \{\pi_1, .., \pi_N\}, \forall l = \{1, .., L\}$ and *(c)*

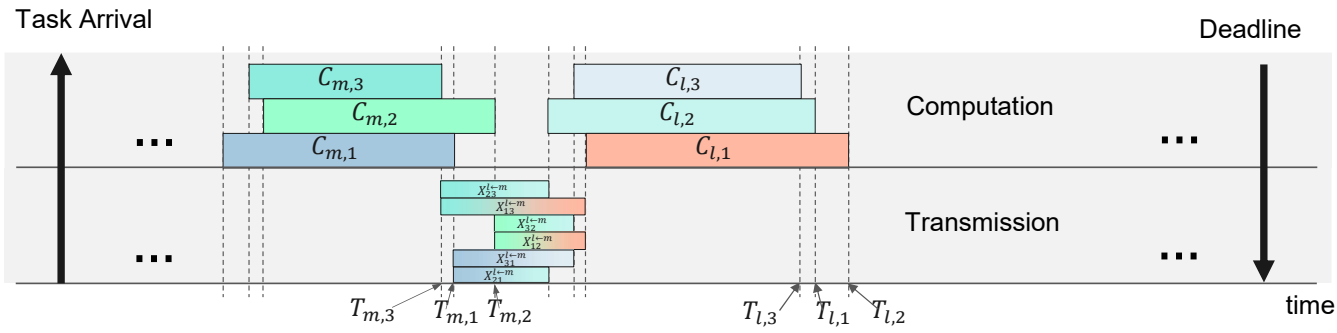

Fig. 3. Computation and communication steps for two dependent layers m and l (where m precedes l) spited across three devices.

the end-to-end execution time, T. Formally, the optimization problem is defined by:

$$\begin{aligned} &\text{minimize} \quad T \\ &\text{subject to: Eq. (3), Eq. (4), and Eq. (5).} \end{aligned} \tag{6}$$

For a DNN inference task with deadline D, the timing constraint is defined by: $T \leq D$, and if that holds, the task meets its real-time requirement. As T and $X_{i\leftarrow j}^{l\leftarrow m}$ are real numbers and Δ_i^l is an integer for $\forall i, l$, the optimization problem in Eq. (6) is a mixed-integer linear programming problem [16], which can be solved using an off-the-shelf solver such as GLPK [17].

IV. Conclusion and Future Work

Our research explores time-aware distributed deep neural inference for resource-constrained embedded devices. As a preliminary effort, we formulate an optimization problem to determine the end-to-end response time of a DNN inference task partitioned across multiple devices. While the concepts presented here lay the groundwork for deadline-aware distributed inference, further investigation is required to make learning-enabled real-time edge systems feasible on resource-constrained platforms. Our immediate focus is on developing efficient methods to solve the formulated optimization problem for large, complex DNN architectures distributed across numerous edge nodes—such as those in wide-area surveillance systems. In our initial formulation, we consider a single DNN task with all nodes available for inference, which may not hold in realistic settings. Future work will extend this framework to scenarios involving multiple periodic or sporadic inference chains and dynamic node availability. This initial work-in-progress represents a crucial first step toward enabling real-time deep learning inference in distributed, low-resource edge applications.

References

[1] R. Qian, X. Lai, and X. Li, "3D object detection for autonomous driving: A survey," *Pattern Recognition*, vol. 130, p. 108796, 2022.

[2] X. Huang, D. Kroening, W. Ruan, J. Sharp, Y. Sun, E. Thamo, M. Wu, and X. Yi, "A survey of safety and trustworthiness of deep neural networks: Verification, testing, adversarial attack and defence, and interpretability," *Computer Science Review*, vol. 37, p. 100270, 2020.

[3] A. Goel, C. Tung, X. Hu, G. K. Thiruvathukal, J. C. Davis, and Y.-H. Lu, "Efficient computer vision on edge devices with pipeline-parallel hierarchical neural networks," in *2022 27th Asia and South Pacific Design Automation Conference (ASP-DAC)*, pp. 532–537, 2022.

[4] M. Lechner and A. Jantsch, "Blackthorn: Latency estimation framework for cnns on embedded nvidia platforms," *IEEE Access*, vol. 9, pp. 110074–110084, 2021.

[5] S. Liu and W. Deng, "Very deep convolutional neural network based image classification using small training sample size," in *2015 3rd IAPR Asian Conference on Pattern Recognition (ACPR)*, pp. 730–734, 2015.

[6] C. Alippi, S. Disabato, and M. Roveri, "Moving convolutional neural networks to embedded systems: The AlexNet and VGG-16 case," in *2018 17th ACM/IEEE International Conference on Information Processing in Sensor Networks (IPSN)*, pp. 212–223, 2018.

[7] S. S. Saha, S. S. Sandha, and M. Srivastava, "Machine learning for microcontroller-class hardware: A review," *IEEE Sensors Journal*, vol. 22, no. 22, pp. 21362–21390, 2022.

[8] S. Teerapittayanon, B. McDanel, and H. Kung, "Distributed deep neural networks over the cloud, the edge and end devices," in *2017 IEEE 37th International Conference on Distributed Computing Systems (ICDCS)*, pp. 328–339, 2017.

[9] S. S. Ogden and T. Guo, "MODI: mobile deep inference made efficient by edge computing," in *USENIX Workshop on Hot Topics in Edge Computing, HotEdge 2018, Boston, MA, July 10, 2018* (I. Ahmad and S. Sundararaman, eds.), USENIX Association, 2018.

[10] L. Zhou, M. H. Samavatian, A. Bacha, S. Majumdar, and R. Teodorescu, "Adaptive parallel execution of deep neural networks on heterogeneous edge devices," in *Proceedings of the 4th ACM/IEEE Symposium on Edge Computing*, SEC '19, (New York, NY, USA), p. 195–208, Association for Computing Machinery, 2019.

[11] Z. Zhao, K. M. Barijough, and A. Gerstlauer, "DeepThings: Distributed adaptive deep learning inference on resource-constrained iot edge clusters," *IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems*, vol. 37, no. 11, pp. 2348–2359, 2018.

[12] L. Zeng, X. Chen, Z. Zhou, L. Yang, and J. Zhang, "CoEdge: Cooperative DNN inference with adaptive workload partitioning over heterogeneous edge devices," *IEEE/ACM Trans. Netw.*, vol. 29, no. 2, pp. 595–608, 2021.

[13] C. Hu and B. Li, "Distributed inference with deep learning models across heterogeneous edge devices," in *IEEE INFOCOM 2022 - IEEE Conference on Computer Communications*, pp. 330–339, 2022.

[14] Y. Kang, J. Hauswald, C. Gao, A. Rovinski, T. Mudge, J. Mars, and L. Tang, "Neurosurgeon: Collaborative intelligence between the cloud and mobile edge," *ACM SIGARCH Computer Architecture News*, vol. 45, no. 1, pp. 615–629, 2017.

[15] Y. Li, Y. Sun, and A. Jog, "Path forward beyond simulators: Fast and accurate gpu execution time prediction for dnn workloads," in *Proceedings of the 56th Annual IEEE/ACM International Symposium on Microarchitecture*, pp. 380–394, 2023.

[16] C. Artigues, O. Koné, P. Lopez, and M. Mongeau, *Mixed-Integer Linear Programming Formulations*, pp. 17–41. Cham: Springer International Publishing, 2015.

[17] "GLPK (GNU Linear Programming Kit)." https://www.gnu.org/software/glpk/. Accessed: 2025-10-08.