Optimizing Task Allocation for DNN Inference on Edge Devices

Mark Kotys, Yijie Zhang, and Chase Q. Wu
Department of Data Science
New Jersey Institute of Technology
Newark, NJ 07102, USA
Email: {mbk4, yz829, chase.wu}@njit.edu

Aiqin Hou
School of Information Science and Technology
Northwest University
Xi'an, Shaanxi 710127, China
Email: houaiqin@nwu.edu.cn

***Abstract*—As artificial intelligence continues to evolve, the need to deploy models at the network edge becomes increasingly critical. A key challenge in such environments is the effective allocation of tasks across devices. To tackle this challenge, we formulate the execution of DNN inference tasks on edge devices as an optimization problem and design a one-to-one task allocation scheme that optimizes the total execution time for a given set of tasks. Our method is both device- and task-aware, employing a greedy algorithm to create task-device mappings. We demonstrate the viability of this approach through comparisons with random allocation and nearest-device strategies, showing that our scheme consistently outperforms these alternatives.**

***Index Terms*—Edge Intelligence, Cloud Computing, Task Assignment**

I. Introduction

The proliferation of high-performance mobile devices and the Internet of Things (IoT) has given rise to transformative applications such as Smart Healthcare, Smart Cities, and Smart Transportation [1]. These interconnected systems generate massive amounts of data, fueling advancements in both big data and artificial intelligence. As data volumes surge, the computational demands of deep neural networks (DNNs) and other models grow in parallel, necessitating more sophisticated approaches to data processing and analysis.

Traditionally, inference requests from devices at the network edge are routed to cloud datacenters for processing, with results sent back through the network. This approach is inefficient, particularly for large data, such as videos or images, leading to significant latency penalties [2]. Moreover, when a large number of devices simultaneously send requests, inference delays can become pronounced, which is unacceptable in latency-sensitive networks such as automotive systems [3]. To address these challenges, a new paradigm, edge intelligence, has emerged, shifting both training and inference computations to the network edge [4].

Edge intelligence operates through a hierarchical, three-tier structure: local, edge, and cloud. Each tier reflects the computational capacity of its devices, with the cloud offering significantly more power than local devices. However, not all tasks require the cloud's resources. This tiered structure enables efficient task distribution: large or high-accuracy tasks are processed in the cloud, while smaller tasks are managed by local and edge devices using lighter models.

However, this structure has limitations. Large models, such as LLMs, often require more memory than local devices can provide. Additionally, running tasks on less powerful machines can significantly increase inference time. For example, the work in [5] highlights a study where processing a single augmented reality frame takes 600 milliseconds, far too slow for practical use.

Given the hierarchical structure of edge intelligence and the constraints posed by device memory and computational power, efficiently distributing tasks across the network becomes critical. Large models cannot always be deployed on local devices, and delays caused by insufficient processing capacity can severely impact real-time applications such as augmented reality. A key challenge in such environments is the effective allocation of tasks across devices. To tackle this challenge, we formalize the execution of DNN inference tasks on edge devices and design an efficient task allocation scheme to optimize the total execution time for a given set of tasks.

The rest of this paper is organized as follows. Section II reviews relevant literature. Section III formalizes the optimization problem concerning the execution time of DNN inference tasks on edge devices. Section IV presents the design of a device- and task-aware, one-to-one task allocation scheme. Section V assesses the performance of the proposed algorithm through experimental evaluation. Finally, Section VI summarizes our findings and outlines directions for future research.

II. Related Work

Significant efforts have been dedicated to optimizing artificial intelligence performance on edge networks. These efforts can be broadly categorized into two areas, which we will explore further in this section.

Model Optimization: The first significant area of focus is the development of methods to optimize the

979-8-3315-2096-0/25/$31.00 ©2025 IEEE

model itself. Two common approaches are weight pruning and quantization [6], [7]. Weight pruning can be further categorized into structured and unstructured methods [8], with the key distinction being that unstructured pruning reduces the dimensions of the pruned matrix, while structured pruning eliminates the least important weights. Quantization, on the other hand, minimizes model size by decreasing the number of bytes used to represent weights. Both approaches aim to reduce model size while preserving as much accuracy as possible, resulting in lighter models suitable for edge devices. Moreover, research has demonstrated that these optimization processes can adapt to varying computational conditions [9]. Consequently, this area has proven essential for enabling the deployment of models on edge devices.

Multi-Layer Utilization: This line of research addresses the network aspect of edge intelligence. Several studies proposed federated learning for model training. For instance, the work in [10] investigates a vehicle edge computing scenario where quantized gradients are transmitted to minimize data transfer over the network. Additionally, a recent approach in [11] involves executing part of the task at the local layer. If the local device lacks sufficient capacity, the remaining computations are offloaded to the edge layer. This way, the inference may be completed and relayed back to the local layer, or it may be forwarded to the cloud for further processing.

Our work addresses the second area, focusing on minimizing the total time required to execute a specified number of tasks on pretrained models. We enhance the efficiency of each layer by developing a task allocation scheme that employs a greedy mapping between tasks and devices. In this approach, we allocate the most computationally intensive tasks to devices based on their performance in a one-to-one mapping. We evaluate our strategy against two benchmarks: a random chance-based method and a closest-device method, comparing their results to evaluate the performance.

III. Problem Formulation

This section constructs the cost models and provides a formal definition of the problem under study.

A. Cost Models

In this paper, we suppose that bandwidth and devices remain stable throughout the input transmission, computation, and output transmission for all tasks. This is valid in many edge computing environments, which are often static; thus, connectivity remains unchanged based on device location, and devices do not unexpectedly disconnect from the network, which would typically pose a challenge [12]. Additionally, we suppose that each task has similar inputs, as all computations are conducted using the same DNN model.

To avoid scheduling issues, which are beyond the scope of this paper, we assume prior knowledge of all tasks to be executed. Additionally, each device is limited to accepting one task at a time to prevent task queuing on any given device.

We define each task as a distinct request from a user within the environment. While each task follows the same quantitative format, its size may vary. Consequently, two tasks may not have identical inference times on the same device, even when utilizing the same DNN model. However, the output dimensions for each task remain consistent.

We consider an environment characterized by a global view and structured into three layers: the local layer ℓ, the edge layer e, and the cloud layer c. Each layer comprises heterogeneous devices represented by $D_h = [1, 2, \ldots, i, \ldots, N]$, where $h \in \{\ell, e, c\}$, and i denotes the index of each device. In this framework, devices within a layer can communicate with one another, while each layer can also interact with other layers.

We define $M = [m_1, m_2, \ldots, m_J]$, where m_j represents a task of index j, and $X = [x_1, x_2, \ldots, x_N]$, where x_i denotes the task assigned to device i, measured in bytes. For simplicity, we use x_i to denote the variable defined as $x = m_j$, meaning $x_i = m_{ji}$. To estimate the computation time of a task on a device, we adopt a method similar to that proposed in [13], characterizing each device by two parameters: λ_i and f_i. The variable λ_i is measured in cycles per kilobyte (KB) of data for the i-th device and is determined by initially processing a small data chunk on the DNN hosted on that device. The variable f_i denotes the number of cycles per second that the device can execute computations; this value is known for each device, given our global view of the environment. Additionally, we define σ_h as the bandwidth between the device where the task is generated and the corresponding layer. Lastly, we define $O = [o_1, o_2, \ldots, o_J]$ as the output produced by task j, measured in bytes.

We compute the total time cost for executing a single task of x_i bytes on device h as follows:

$$TC_i^h = \frac{x_i}{\sigma_h} + \frac{\lambda_i x_i}{f_i} + \frac{o}{\sigma_h}. \tag{1}$$

Since we are considering a single task in this case, we omit the subscript j. The equation can be divided into three distinct components. The first term calculates the time required to transfer the task over the network; if the device resides within the same layer as h, we assume that the value of this term is zero and hence ignored. The second term estimates the time for the i-th device to complete the task, referred to as CT_i for computation time. The third term represents the time taken to transfer the output back to the originating device, which is also ignored in the case of intra-layer communication. The first and third terms collectively constitute the data transfer time DT.

Given a set M of tasks, we compute the total time as:

$$T = \sum_{j \in M} min(TC_j^\ell, TC_j^e, TC_j^c), \tag{2}$$

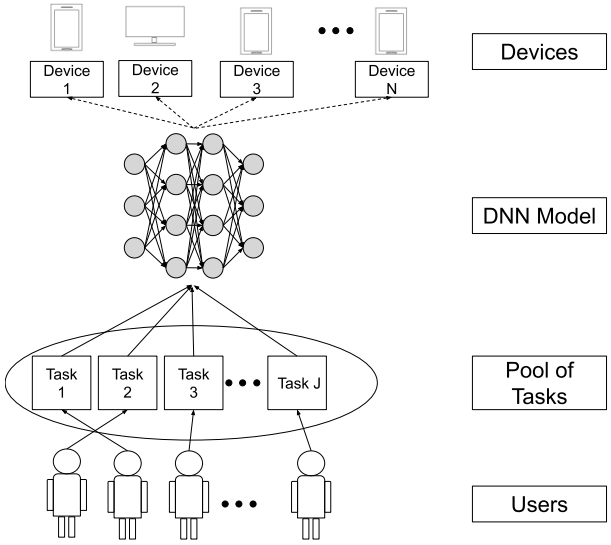

Fig. 1: DNN inference task allocation problem on edge devices.

which calculates the summation of the minimal time it takes to run each task on local, edge, or cloud devices.

B. Problem Definition

Definition 1: Given a list of N devices and a set M of tasks as the input, we formulate the problem as

$$min(T), \tag{3}$$

where our goal is to minimize the total time it takes to run those tasks on the set of devices.

Using this problem definition, we formulate a task allocation problem as illustrated in Figure 1. In this scenario, a group of users generates a pool of tasks. Each task m_j, as defined in the previous section, represents a request to the DNN. Once the request is initiated, the task is transferred to device x_i, where the inference is performed. This allocation is determined based on the characterization values, λ_i and f_i, which are either calculated or predetermined for each device, as defined in Section III. Upon completion of the computation, the output is transmitted back to the originating device. We utilize this scheme in order to provide a solution for our problem.

The verification for a given solution to this task allocation problem involves calculating the total execution time. Given an allocation of tasks to devices, we compute the execution time for each task and sum them. Since summing and comparing the total time can be done in polynomial time, the problem is in NP. Furthermore, this problem is formulated as an Integer Linear Programming problem, and is NP-complete.

IV. Algorithm Design

In this section, we design a task allocation scheme for the minimization problem defined in the previous section. This scheme employs a one-to-one greedy mapping approach, wherein each device is permitted to accept only one task at a time. Given that the data transfer time is negligible for intra-layer communication and constant for inter-layer communication, the proposed algorithm focuses solely on the essential calculations required to map tasks to devices.

Our algorithm features a two-phase process. In the first phase, we categorize each machine based on its computational capacity. This global view enables us to identify available devices and ensures that the most powerful devices are prioritized. In the second phase, we rank the tasks generated by users according to their input sizes, since larger inputs typically correspond to longer execution times. With both the tasks and devices ranked, we establish a one-to-one mapping between the largest task and the most capable machine. Phase one is detailed in Algorithm 1, while the mapping procedure is presented in Algorithm 2. The following paragraphs provide an in-depth explanation of each algorithm.

We begin by discussing Algorithm 1, which represents the initial stage of our approach, where we compute λ_i for each device. This algorithm requires two inputs: a list D of devices and a small task S used to evaluate the performance of each device. The algorithm generates pairs that consist of the calculated λ_i values and their associated devices. To minimize the total inference time, we prioritize the most computationally capable devices by sorting the resulting array, which the algorithm returns.

Algorithm 1 Task Measurement Device Ranking

Input:
D: A list of devices $[1, 2, 3, \ldots, N]$
S: A small task for evaluation

```
if D is empty then
    return NULL;
arr[N];        ▷ Array to store device-performance pairs
for i = 1 to N do
    d ← D[i];                    ▷ Retrieve device at index i
    Compute λ_i by executing task S on the model;
    Create pair (λ_i, d);
    Insert pair into arr[i];
Sort arr by the first value in each pair;
Output: Return arr.
```

Next, we provide a detailed discussion of Algorithm 2. This program takes as input the array generated by Algorithm 1, along with the set M of tasks and the values for f_i.

We begin by validating the input parameters. For instance, if Algorithm 1 fails, it indicates that the array does not exist, rendering Algorithm 2 incapable of establishing a mapping. Similarly, an empty task list precludes any possibility of mapping. Upon confirming the validity of the input, we proceed to sort the tasks based on their sizes,

Algorithm 2 Priority-Based Dynamic Task Allocation

Input:

A: Array generated by Algorithm 1.
M: Tasks $[m_1, m_2, m_3, \ldots, m_J]$.
$f_i, \forall i \in D$.

```
if A is NULL then
    return NULL;
if M is empty then
    return NULL;
Sort M in descending order;
x ← 0;
for m_j ∈ M do
    Remove the first element from M;
    Assign m_j to A[x];
    x ← x + 1;
```

ensuring that the largest tasks are assigned to the most computationally capable devices. This greedy approach prioritizes optimal decisions based on the remaining devices and tasks. Following the creation of this mapping, we initiate the process of transferring each task to its designated device. Once the data is loaded, we measure the execution time of the program on the specified model, thereby determining the computation time (CT) for each task on its respective device. While this algorithm may not yield the utmost efficiency, it serves as a heuristic to minimize the total computational time for the given tasks.

V. Performance Evaluation

A. Experiment Settings

In this section, we present the implementation details of our task allocation algorithm.

The experimental setup comprises a cluster of six physical machines: three Dell PowerEdge R740 machines, each with 20 CPUs (Intel(R) Xeon(R) Silver 4114 CPU @ 2.20 GHz) and 256 GB of memory; two Dell Precision Rack 7910 machines, each with 6 CPUs (Intel(R) Xeon(R) CPU E5-2620 v3 @ 2.40 GHz) and 16 GB of memory; and one Dell PowerEdge R730 machine with 8 CPUs (Intel(R) Xeon(R) CPU E5-2620 v4 @ 2.10 GHz) and 16 GB of memory. We utilize VMware ESXi to provision virtual machines.

Our algorithm is deployed on a cluster comprising of twenty-five virtual machines, each utilizing the ResNet-18 model. These virtual machines are classified into five distinct categories based on the number of allocated CPU cores. The most computationally capable machine is equipped with eight cores, followed by configurations with six, four, three, and two cores. Notably, none of the devices utilize GPUs; thus, the pre-trained ResNet-18 models are executed solely on the CPU cores.

We employ a modified version of the CIFAR-10 dataset to facilitate task heterogeneity. To achieve this, we increase the image sizes in the dataset. Similar to the categorization of devices, we define five categories for the tasks. Originally, each image in the dataset measured 32×32 pixels; following our modifications, the largest images expanded to 32×224 pixels. Given that the input tasks are relatively small, the data transfer time is deemed negligible compared to the computation time, allowing us to focus solely on the computational aspect when determining the total execution time. It is important to note that this is specifically a result of using the CIFAR-10 dataset. Other datasets may have much larger input sizes, necessitating the consideration of data transfer time.

In our experimental design, we progressively increase the number of tasks executed, ensuring that the total number of tasks remains below the available device capacity. Each experiment is conducted five times, with the final results presented as the average of these five runs.

B. Experimental Results

The experimental results are illustrated in Figure 2, where we compare our proposed method against alternative task allocation strategies. Specifically, we evaluate the performance relative to a random allocation scheme, denoted as RA, and a nearest device strategy, referred to as ND. Additionally, we analyze the worst-case scenario for the ND scheme in juxtaposition with its best-case scenario. Here, we would like to point out that our allocation method matches the best-case runtime efficiency of the ND strategy, confirming its effectiveness in task distribution. As shown in Figure 2, our method aligns with the ND best-case curve.

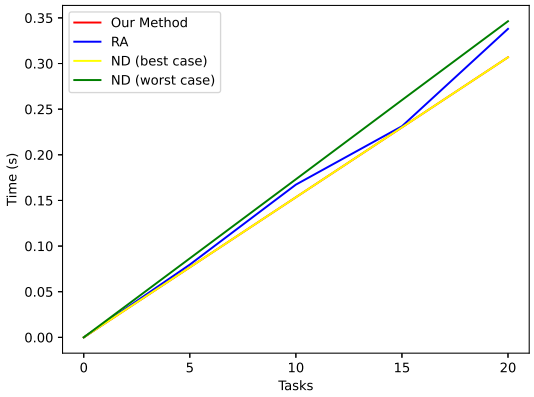

Fig. 2: Total execution time for different task set sizes.

Given that we have prior knowledge of all tasks and, as outlined in Algorithm 2, we prioritize the allocation of the largest tasks, we ascertain that the worst-case scenario for the ND scheme occurs when devices are sorted in the reverse order of their capabilities. In this configuration, the ND method exhibits the longest run time, performing 13% slower than our approach.

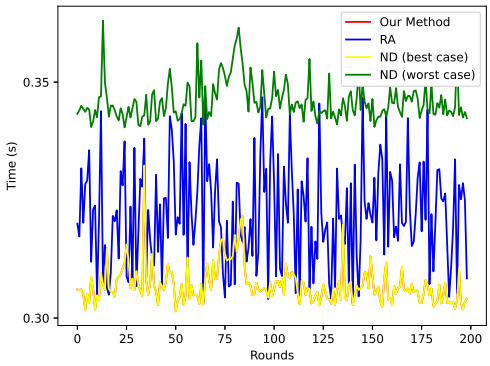

Fig. 3: Total execution time for 200 runs using different methods.

Subsequently, we examine the random assignment strategy, which permits any task to be allocated to any device. This method yields marginally better performance than the worst-case ND scenario, resulting in a run time that is 9.8% slower than our proposed solution. Notably, our allocation method demonstrates an equivalent run time performance to that of the best-case scenario for the nearest device approach. This finding is anticipated, as the sequential distribution of tasks in our method effectively aligns with the allocation mechanism of the nearest device strategy. As a result, the lines that represent the best-case for the nearest device method is overlapping with the line representing our method in both Figure 2 and Figure 3. Note again that our work is focused on how the tasks are distributed, not the model itself.

In Figure 2, we present the total run time as a function of the increasing number of tasks. Each experiment is conducted on the complete dataset, and the average total execution time is calculated across all runs.

As shown in Figure 3, each round consists of a total of 20 tasks, meaning that there are four tasks allocated to each layer of machines. For instance, four are allocated to the 8-core machines, four to the 6-core machines, etc.

Our task allocation method demonstrates a significant performance advantage over existing approaches. Experimental results indicate that, compared with the random allocation scheme (RA) and the nearest device strategy (ND), our method excels in total execution time. Furthermore, our allocation method performs on par with the best-case scenario of the ND strategy in terms of runtime efficiency, validating its superior performance in task distribution. This advantage is attributed to our greedy strategy, which prioritizes assigning the most computationally intensive tasks to the most capable devices, thereby effectively reducing overall computation time. Consequently, our method not only enhances resource utilization but also optimizes task execution efficiency, rendering it highly applicable in edge computing environments.

VI. Conclusion

In this study, we aimed to optimize the total execution time required to process multiple tasks on a set of edge devices. We achieved this objective through the design of a greedy task allocation algorithm that establishes a one-to-one mapping between tasks and devices. By consistently assigning the most computationally intensive tasks to the most capable machines, we effectively minimized the overall execution cost. Extensive experimental results demonstrate that our method not only achieves joint-best performance but also outperforms alternative allocation strategies. It is of our future interest to consider data transfer time and task distribution across different layers in the hierarchy of computing continuum.

References

[1] O. Aouedi, T.-H. Vu, A. Sacco, D. C. Nguyen, K. Piamrat, G. Marchetto, and Q.-V. Pham, "A survey on intelligent internet of things: Applications, security, privacy, and future directions," *arXiv:2406.03820*, 2024.

[2] Y. Cai, J. Llorca, A. M. Tulino, and A. F. Molisch, "Mobile edge computing network control: Tradeoff between delay and cost," pp. 1–6, 2020.

[3] M. A. Lema, A. Laya, T. Mahmoodi, M. Cuevas, J. Sachs, J. Markendahl, and M. Dohler, "Business case and technology analysis for 5g low latency applications," *IEEE Access*, vol. 5, pp. 5917–5935, 2017.

[4] Y. Mao, C. You, J. Zhang, K. Huang, and K. B. Letaief, "A survey on mobile edge computing: The communication perspective," *IEEE Communications Surveys & Tutorials*, vol. 19, no. 4, pp. 2322–2358, 2017.

[5] G. Pan, H. Zhang, S. Xu, S. Zhang, and X. Chen, "Joint optimization of dnn inference delay and energy under accuracy constraints for ar applications," pp. 2230–2235, 2022.

[6] X. Ma, S. Lin, S. Ye, Z. He, L. Zhang, G. Yuan, S. H. Tan, Z. Li, D. Fan, X. Qian, X. Lin, K. Ma, and Y. Wang, "Non-structured dnn weight pruning—is it beneficial in any platform?" *IEEE Transactions on Neural Networks and Learning Systems*, vol. 33, no. 9, pp. 4930–4944, 2022.

[7] Y. Kim, J. Kim, D. Chae, D. Kim, and J. Kim, "ulayer: Low latency on-device inference using cooperative single-layer acceleration and processor-friendly quantization," 2019. [Online]. Available: https://doi.org/10.1145/3302424.3303950

[8] H. Cheng, M. Zhang, and J. Q. Shi, "A survey on deep neural network pruning: Taxonomy, comparison, analysis, and recommendations," *IEEE Transactions on Pattern Analysis and Machine Intelligence*, pp. 1–20, 2024.

[9] B. Eccles, P. Rodgers, P. Kilpatrick, I. Spence, and B. Varghese, "Dnnshifter : An efficient dnn pruning system for edge computing," *Future Generation Computer Systems*, vol. 152, 09 2023.

[10] C. Zhang, W. Zhang, Q. Wu, P. Fan, Q. Fan, J. Wang, and K. B. Letaief, "Distributed deep reinforcement learning based gradient quantization for federated learning enabled vehicle edge computing," *IEEE Internet of Things Journal*, pp. 1–1, 2024.

[11] S. Teerapittayanon, B. McDanel, and H. Kung, "Distributed deep neural networks over the cloud, the edge and end devices," pp. 328–339, 2017.

[12] T. Ouyang, Z. Zhou, and X. Chen, "Follow me at the edge: Mobility-aware dynamic service placement for mobile edge computing," *arXiv:1809.05239*, 2018.

[13] L. Zeng, X. Chen, Z. Zhou, L. Yang, and J. Zhang, "Coedge: Cooperative dnn inference with adaptive workload partitioning over heterogeneous edge devices," *arXiv:2012.03257*, 2020.