# On the Optimal Ensemble of Distributed DNN Models

Heejin Kim
*Department of Computer and Science Engineering*
*Konkuk University*
Seoul, South Korea
juliet13579@konkuk.ac.kr

Hyang-Won Lee
*Department of Computer and Science Engineering*
*Konkuk University*
Seoul, South Korea
leehw@konkuk.ac.kr

***Abstract*—In this paper, we introduce a novel problem of ensembling deep neural network (DNN) models distributed across the network, which can be a key element for distributed inference on resource-constrained end devices. We first develop a mixed integer linear program (MILP) that selects a combination of models to maximize accuracy on a negative sample set. Building on this formulation, we develop an MILP-based approach to determine an optimal path for ensembling distributed DNN models while satisfying a given latency constraint. The problem contains elements of the classical Traveling Purchaser Problem (TPP) and submodular function maximization, both of which are NP-hard. As the problem contains doubly NP-hard subproblems, we develop heuristic algorithms (some with approximation guarantee) for each subproblem in order to construct high-accuracy ensembles while adhering to latency constraints. Through extensive simulations, we demonstrate that our approach effectively finds an ensemble of models with high accuracy and low latency in distributed settings.**

***Index Terms*—Distributed Model Ensemble, Accuracy and Latency, Mixed Integer Linear Programming**

## I. Introduction

Artificial intelligence (AI) services leveraging distributed computing resources are a crucial component of the 6G system, where various AI-driven applications are expected to be deployed [1]. This is particularly important for end devices with limited resources, leading to the emergence of techniques such as edge computing, cloud computing, and hybrid approaches that combine both to overcome these limitations [2].

The central problem in this context is to offload the computation of deep neural networks (DNNs) to distributed computing resources, so that the end device can have access to AI services that potentially require heavy computation. There are several approaches to offloading the computation of DNN models. For instance, one can offload the entire computation of a given DNN model (needed for inference tasks) to an edge or cloud server [3]. Alternatively, some works exploit the layered structure of DNN models, and develop algorithms for partitioning and offloading individual or grouped layers [4], [5]. These approaches assume that the model to be executed is given a priori, and hence, the primary focus is on minimizing the inference latency by efficiently utilizing distributed computing resources.

This work was supported by the National Research Foundation of Korea(NRF) grant funded by the Korea government(MSIT)(RS-2025-00516578).

On the other hand, some works consider the problem of selecting one of the models that are placed in end device and edge server [6]. Since the expected accuracy can differ depending on the selected model, the solution must take into account both accuracy and latency. In particular, accuracy tends to improve as model size (and consequently, computational demand) increases [7], and hence, there is a tradoff between accuracy and latency. The solution in this context must strike a balance between accuracy and latency based on the available computing resources.

In this paper, we introduce a novel problem of ensembling DNN models distributed across the network. Unlike the existing works where a "single" model is predetermined or should be selected, our problem seeks to find a combination of distributed models in order to maximize accuracy, while satisfying a latency constraint. The solution to the problem enables to yield accuracy commensurate with available resources. To the best of our knowledge, this problem has not been studied in the literature, although model ensemble without consideration of distributed setting has been studied.

There are several challenges to solve the distributed model ensemble problem. First, the accuracy of ensembled models is highly uncertain. In the field of AI, there are several metrics that quantify the diversity of combined models based on their correct or incorrect classifications [8], [9]. Model diversity is crucial, as a sample misclassified by one model may be correctly classified by the ensemble due to the compensatory effect of other models making accurate predictions. However, whether maximizing diversity metrics directly translates to improved accuracy remains unclear. Second, the problem contains elements of the classical submodular function maximization and TPP, both of which are NP-hard.

In order to address the challenges, we first develop a novel mixed integer linear program (MILP) that selects a combination of models to maximize accuracy on a negative sample set. We show that the negative sample set accuracy is highly correlated with test accuracy, which justifies the use of the MILP formulation for model ensemble. Based on the MILP and layered graph representation of distributed model placement, we formulate the problem of jointly finding a combination of models and a computation path in order to maximize the accuracy subject to an end-to-end latency

ISBN 978-3-903176-73-7 

constraint. We develop several heuristic algorithms for the problem, which can efficiently find an ensemble path with enhanced accuracy and latency.

Our contributions can be summarized as follows:

- We develop a mixed integer linear program (MILP) that selects a combination of models to maximize accuracy on a negative sample set.
- We propose an MILP-based approach to determine an optimal path for ensembling distributed DNN models while satisfying a given latency constraint.
- We develop heuristic algorithms (some with approx. guarantee) for each subproblem in order to construct high-accuracy ensembles while adhering to latency constraints.
- We show through simulations that our algorithms can effectively find an ensemble with its computation path for enhanced accuracy and latency.

The rest of the paper is organized as follows. In Section II, we discuss related work. In Section III, we present the system model and describe the problem. In Section VI, we present heuristic algorithms for our NP-hard subproblems. In Section VII, we present numerical and simulation results demonstrating the performance of our formulations and algorithms. We conclude the paper in Section VIII.

## II. Related Work

Several studies have investigated the problem of offloading entire DNN computations to edge or cloud servers [3]. These approaches do not differ significantly from general edge or cloud computing solutions, where computational tasks are either offloaded or executed locally. In this context, DNN computation is simply treated as a single computational task.

Since DNN models typically consist of multiple layers, many offloading solutions tailored for DNN computation involve partitioning these layers. Specifically, the layers are divided into two or more groups, with each group offloaded as a unit. There are a number of solutions for joint partitioning and offloading of DNN layers, including [10]–[14] based on mathematical optimization, [15]–[19] based on reinforcement learning, and [20]–[22] based on Lyapunov optimization. These works develop partitioning and offloading algorithms for a single DNN model predetermined to be executed, and hence, the primary focus is on latency performance.

The studies most relevant to our work are those that consider both accuracy and latency. In this line of research, the central problem typically involves selecting a model from a set of available options to maximize accuracy while accounting for latency constraints. The work of [6] assumes a scenario where a small, low-accuracy model is cached on an end device, while a larger, high-accuracy model is stored on an edge server. The goal of the problem is to select either the small or large model while jointly optimizing sampling rate control and resource allocation. The work of [23] studies the problem of selecting both resolution and model for video frame processing in an edge computing environment to maximize accuracy while satisfying a latency constraint. In [24], the problem of not only model selection but also model placement (or caching) is addressed in an end-edge-cloud computing environment, and a deep reinforcement learning-based solution is proposed.

Our work in this paper differs from the existing studies in that we aim to combine multiple models distributed across the network, rather than selecting a single model. Furthermore, we consider a general computing network with arbitrary topology where models can be placed anywhere in the network. This introduces substantial challenges that are not present in the existing works, including the prediction on the accuracy of ensembled models and the routing of computation flows.

## III. System Model and Problem Description

In this section, we describe the system model, and the joint ensemble and routing problem that takes into account both accuracy and latency. Let $\mathcal{G}_P = (\mathcal{V}_P, \mathcal{E}_P)$ denote the physical network where $\mathcal{V}_P$ is the set of nodes and $\mathcal{E}_P$ is the set of communication links connecting the nodes. Each node can potentially compute DNN models for inference tasks. Let $\mu_u$ be the computation capacity of node $u$, and the unit of $\mu_u$ can for example be GFLOPS. Likewise let $\mu_{uv}$ be the transmission capacity of link $(u, v)$ of which the unit can be Mbps or Gbps.

Let $\mathcal{M}$ be the set of all available DNN models, each of which is placed in some node(s) across the network. Let $\mathcal{M}_u \subseteq \mathcal{M}$ be the set of models placed in node $u$. Let $\mathcal{V}_m$ be the set of nodes where model $m$ is placed. In this work, we do not consider the partitioning of a model, i.e., each model $m$ must be computed entirely at a single node in $\mathcal{V}_m$. The computation required by model $m \in \mathcal{M}$ is denoted by $c_m$, and thus, the computation time of model $m$ at node $u$ is written as $\frac{c_m}{\mu_u}$. Let $d^i$ and $d^o$ be the sizes of input data (such as image) and output data (such as softmax values), respectively.

The inference job $j$ is defined as tuple $(s_j, t_j)$, so that the data is generated at source $s_j \in \mathcal{V}_P$ and the inference result on the data is destined to $t_j \in \mathcal{V}_P$. We assume that any combination of DNN models can be used for each inference job $j$. Once a combination of models is determined, those selected models are computed sequentially. For instance, consider the scenario in Fig. 1 where the source is node $s$ and the destination is node $t$. Suppose that models $m_1$ and $m_2$ are selected for ensemble. Then, $m_1$ is computed and then $m_2$ is computed. The blue path illustrates an example of this sequence of computations. On the other hand, the reverse order of computations is also possible, which is illustrated by the red path in the figure.

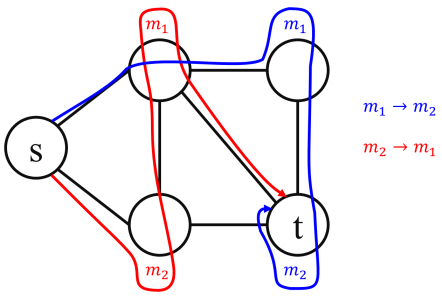


Fig. 1. Illustration of model ensemble and routing problem

In this work, we study the problem of ensembling models and routing the computation of ensembled models in order to maximize accuracy while satisfying a latency constraint. Hence, our problem aims to determine not only the set of models to combine but also the path along which combined models are computed. It is clearly desirable to ensemble models that can lead to best (test) accuracy. While it is generally expected to yield better accuracy with more models, it is hard to predict the accuracy of ensembled models and the accuracy can even be deteriorated with ensemble. Furthermore, even for a given combination of models, the routing problem is NP-hard as it contains the classical TPP. To address these challenges, we develop several techniques including a novel formulation for ensemble and layered graph representation of joint model ensemble and computation path selection.

## IV. Formulation for Model Ensemble

In this section, we develop a novel formulation for model ensemble. This formulation is not only useful for model ensemble but also serves as the foundation for addressing the joint ensemble and routing problem.

We consider the classification problem of $C$ classes, and the set of classes is denoted as $\mathcal{C} = \{1, ..., C\}$. Let $\mathcal{D}$ be the set of data samples. Each model produces the $C$-dimensional output vector of probability for an input sample in $\mathcal{D}$. Specifically, let $P_{ic}^m$ be the probability of class $c \in \mathcal{C}$ produced by model $m$ for input sample $i \in \mathcal{D}$. The ground truth of sample $i$ is denoted as $c_i \in \mathcal{C}$, and hence, if $P_{ic_i}^m < \max_c P_{ic}^m$, then model $m$ misclassifies sample $i$. Let $\mathcal{D}^m$ be the set of data samples that model $m$ misclassifies. This is referred to as the *negative sample set* [9].

Define the variable $x_m$ to be 1 if model $m \in \mathcal{M}$ is selected for ensemble, and 0 otherwise. We adopt the standard soft voting as an aggregation method for ensemble [25], i.e., we aggregate the output probabilities of selected DNN models as

$$P_{ic}(x) = \frac{1}{M} \sum_{m \in \mathcal{M}} x_m P_{ic}^m, \forall c \in \mathcal{C}, \forall i \in \mathcal{D}. \tag{1}$$

This is not exactly the probability, but can be used for classification, i.e., for given ensemble $x$, data sample $i$ is classified as $\arg\max_c P_{ic}(x)$.

Now it remains to develop the criteria for ensembling the models. The challenge lies in the fact that the key objective is to maximize accuracy on test data during inference, yet these data remain unseen at the time of deciding the model ensemble. In the literature of AI, there are several diversity metrics that can be used for ensemble, however we found that those metrics are hard to be mathematically formulated for selecting models. We therefore develop a novel formulation for model ensemble, which is based on negative sample set. The key idea is based on the conjecture that the accuracy of ensembled models is likely to improve if they achieve high accuracy on the negative sample set.

Let $y_i^m$ takes the value 1 if negative sample $i$ of model $m$ is classified correctly by ensembled models, and 0 otherwise. We impose the following bound on $y_i^m$:

$$y_i^m \leq 1 + (P_{ic_i}(x) - \max_{c \in C} P_{ic}(x)), \forall i \in \mathcal{D}^m, m \in \mathcal{M}. \tag{2}$$

This ensures that if the ensembled models misclassify negative sample $i$ (i.e., $P_{ic_i}(x) < \max_{c \in C} P_{ic}(x)$), then the variable $y_i^m$ takes the value 0. With linearization of the max operation, we formulate the model ensemble problem as follows:

$$\max_{x,y,z} \quad \sum_{m \in \mathcal{M}} \sum_{i \in \mathcal{D}^m} y_i^m \tag{3}$$

$$\text{s.t.} \quad y_i^m \leq 1 + (P_{ic_i}(x) - z_i^m), \forall i \in \mathcal{D}^m, m \in \mathcal{M} \tag{4}$$

$$z_i^m \geq P_{ic}(x), \forall c \in C, i \in \mathcal{D}^m, m \in \mathcal{M} \tag{5}$$

$$\sum_{m \in \mathcal{M}} x_m \geq 1, \tag{6}$$

where $z_i^m$ is the continuous auxiliary variable. The objective function (3) is the number of negative samples the selected model combination correctly classified. The constraints (4) and (5) are an equivalent linearization of constraint (2). The constraint (6) ensures that one or more DNN models are selected from $\mathcal{M}$. Therefore, this mixed integer linear program (MILP) aims to find a combination of models that maximize the accuracy on negative sample set. In Section VII, we show that this formulation can effectively find a model ensemble with high test accuracy.

The above optimization problem is closely related to the classical submodular function maximization, and in Section VI, we present a greedy algorithm for maximizing $f(x)$ under knapsack (or latency) constraint.

## V. Joint Ensemble and Path Selection

By leveraging the MILP formulation for ensemble, we develop a formulation for joint ensemble and computation path selection. We start with the layered graph representation of the joint model and path selection. As shown in Fig. 2, $M + 1$ replications of the original network $\mathcal{G}_P$ are layered, which are denoted as $\mathcal{G}_l = (\mathcal{V}_l, \mathcal{E}_l)$ for $l = 0, ..., M$. There are directional cross-layer edges connecting from $\mathcal{G}_l$ to $\mathcal{G}_{l+1}$. Let $u_l \in \mathcal{V}_l$ be the copy of original node $u \in \mathcal{V}$. There are $|\mathcal{M}_u| + 1$ edges from $u_l$ to $u_{l+1}$ for each $l(= 0, ..., M - 1)$. Each of these cross-layer edges is used to indicate whether the corresponding model is selected for ensemble and computed at the corresponding node (see Fig. 2). Note that the number of models available at node $u$ is $|\mathcal{M}_u|$, however, we have one extra edge to represent the bypass that no computation is carried out at the corresponding node (dashed gray edge in the figure). The cross-layer edge is denoted as tuple $(u, v, m)$, which indicates that for some $w \in \mathcal{V}_P$, we have $u = w_{l-1}, v = w_l, l = 1, ...M$ and $m \in \mathcal{M}_w \cup \{0\}$. Note that $(u, v, 0)$ is the bypass edge. In order to unify the notation for edges, we sometimes use the notation $(u, v, -1)$ for intra-layer edge $(u, v)$ with $u, v \in \mathcal{V}_l$ for some $l$.

Suppose that the source and destination nodes are $s \in \mathcal{V}_P$ and $t \in \mathcal{V}_P$ respectively. In the layered graph, the source node

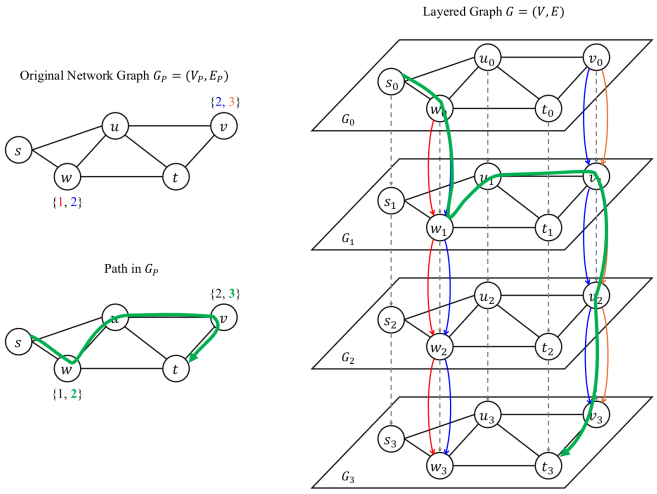


Fig. 2. Layered graph representation of joint model ensemble and routing

is assumed to be $s_0$, and the destination node is $t_M$. Consequently, the path from $s_0$ to $t_M$ can be used to simultaneously represent the model ensemble and computation path, but we need some constraints. Specifically, at most one of the cross-layer edges corresponding to each model should be selected because otherwise, the same model can be ensembled multiple times. This constraint is encoded with decision variable $x_m$.

To express the latency, the capacity of each link is defined as follows:

- $\mu_{u_l v_l} = \mu_{uv}, \forall (u,v) \in V_P, l \in \mathcal{M} \cup \{0\}$
- $\mu_{u_{l-1} u_l} = \mu_u, \forall u \in V_P, l \in \mathcal{M}$.

In addition, the size $d_l$ of data flowing in each layer $l$ is defined as follows:

$$d_l = \begin{cases} d^i, & \text{if } l = 0, \\ d^i + d^o, & \text{if } l = 1, ..., M-1, \\ d^o, & \text{if } l = M. \end{cases}$$

The data size with $l = 0$ is the pure input data size before passing through any model, whereas the size with $l = 1, ..., M-1$ is the input data size plus the size of softmax vector after passing through a model. The amount of data that should be processed at each link in the layered graph is defined as

$$q_{uvm} = \begin{cases} d_l, & \text{if } u, v \in \mathcal{V}_l \text{ and } m = -1 \\ c_m, & \text{if } m \geq 0 \text{ and } \exists l : u \in \mathcal{V}_{l-1}, u \in \mathcal{V}_l \end{cases}$$

where $c_0 = 0$ for bypassing.

Define the routing variable $r_{uvm}$ that takes the value 1 if the path traverses edge $(u, v, m)$, and 0 otherwise. The joint ensemble and routing problem can be formulated as follows:

$$\max_{x,y,z,r} \quad \sum_{m \in \mathcal{M}} \sum_{i \in \mathcal{D}^m} y_i^m \tag{7}$$

$$\text{s.t.} \quad \sum_{(u,v,m) \in \mathcal{E}} \frac{q_{uvm}}{\mu_{uvm}} r_{uvm} \leq \delta \tag{8}$$

$$\sum_{(u,v,m') \in \mathcal{E}: m' = m} r_{uvm'} = x_m, \quad \forall m \in \mathcal{M} \tag{9}$$

$$[r_{uvm}, \forall (u,v,m) \in \mathcal{E}] \text{ forms a path } s_0 \rightsquigarrow t_M \tag{10}$$

$$(4), (5), (6) \tag{11}$$

The constraint in (8) requires that the end-to-end latency should not exceed a threshold $\delta$. The constraint in (9) ensures that if model $m$ is selected for ensemble (i.e., $x_m = 1$), then exactly one edge corresponding to model $m$ should be traversed. Otherwise, with $x_m = 0$, no edge should be traversed. Therefore, the above formulation seeks to find an ensemble of models while satisfying a latency constraint. While this MILP can be solved using standard solvers, it may take too long to find a solution, and hence, we develop efficient heuristic algorithms in the next section.

## VI. Heuristic Algorithms

We first show that the joint problem contains elements of NP-hard problems. The optimal ensemble problem in (3)-(6) can indeed be equivalently rewritten as the maximization of the following function:

$$f(x) = \sum_{m \in \mathcal{M}} \sum_{i \in \mathcal{D}^m} \mathbf{1}_{\{P_{ic_i}(x) = \max_c P_{ic}(x)\}}, \tag{12}$$

where $\mathbf{1}_{\{\cdot\}}$ is the indicator function. We can show the following two lemmas highlighting the difficulty of the joint ensemble and routing problem. The proofs of these lemmas are omitted due to limited space.

*Lemma 1:* Suppose that there is only a single node which is the source and destination, and that the function $f(x)$ in (12) is monotone and satisfies the condition: $f(x + y) = 0$ if $f(x) = f(y) = 0$. Then, the optimization problem in (7)-(11) contains the submodular function maximization problem under knapsack constraint [26].

*Lemma 2:* Suppose that the model ensemble is given. Then, the optimization problem in (7)-(11) contains the traveling purchaser problem (TPP).

Both the submodular function maximization problem and TPP are NP-hard, and hence, it is necessary to develop heuristic algorithms. The above lemmas offer valuable insights for designing efficient heuristics. We indeed develop heuristic algorithms inspired by approximation techniques for submodular function maximization under knapsack constraints and the traveling purchaser problem.

### A. Greedy Algorithm

We first describe a greedy algorithm, which also gives an overall structure of our heuristic algorithms in the following. To this end, we modify the greedy algorithm proposed in [26] that solves the submodular function maximization under

knapsack constraints. The knapsack constraint corresponds to the latency constraint in our problem. While the knapsack constraint in [26] allows for easy verification of violations in a selected combination, our latency constraint is significantly more complex to evaluate, as it involves an NP-hard Traveling Purchaser Problem (TPP).

Our greedy algorithm is presented in Algorithm VI-A. In Phase 1, it searches all the combinations of up to size 2, and selects a combination with largest value of $f(S)$ while satisfying the latency constraint $\hat{c}(S) \leq \delta$. Here, we use $\hat{c}(S)$ to denote the estimate of latency for computing the ensemble $S$. It is desirable to compute the smallest possible value of $\hat{c}$, however it is an NP-hard TPP. We will discuss heuristics to estimate the latency in the next subsection.

In Phase 2, each subset of $\mathcal{M}$ with size 3 is used an initial combination, and a model is added iteratively by selecting the one that maximizes the ratio of the increase in negative sample set accuracy to the increase in cost (or latency). However, such a model is not always added to the set $S$. If the addition of the model reduces the objective function $f$, then the while loop is broken (see lines 9-11). Furthermore, if adding a model violates the latency constraint, then the loop continues without adding the model (see lines 14-16). After Phase 2, the combination (among those obtained from Phases 1 and 2) with larger objective function value is selected (see lines 22-26).

If the conditions in Lemma 1 hold, then Algorithm VI-A gives an $(1 - 1/e)$ approximation, which is an immediate consequence of the results in [26].

**Algorithm 1** Greedy Algorithm

1: **Phase 1**
2: $S_1 \leftarrow \arg\max_{S \subset \mathcal{M}} \{f(S) : \hat{c}(S) \leq \delta, 1 \leq |S| \leq 2\}$;
3: **Phase 2**
4: $Card_3 = \{S \in 2^{\mathcal{M}} \quad | \quad |S| = 3\}$; $S_2 \leftarrow \emptyset$;
5: **for** $U \in Card_3$ **do**
6: $S^0 = U, t = 1, I^0 = \mathcal{M}$;
7: **while True do**
8: $i_t = \arg\max_{i \in I^{t-1} \setminus S^{t-1}} \{\frac{f(S^{t-1} \cup \{i\}) - f(S^{t-1})}{\hat{c}(S^{t-1} \cup \{i\}) - \hat{c}(S^{t-1})}\}$
9: **if** $f(S^{t-1} \cup \{i_t\}) - f(S^{t-1}) \leq 0$ **then**
10: **break**;
11: **end if**
12: **if** $\hat{c}(S^{t-1} \cup \{i_t\}) \leq \delta$ **then**
13: $S^t = S^{t-1} \cup \{i_t\}$ and $I^t = I^{t-1}$;
14: **else**
15: $S^t = S^{t-1}$ and $I^t = I^{t-1} \setminus \{i_t\}$;
16: **end if**
17: **end while**
18: **if** $f(S_2) \leq f(S^t)$ **then**
19: $S_2 \leftarrow S^t$
20: **end if**
21: **end for**
22: **if** $f(S_1) \geq f(S_2)$ **then**
23: **return** $S_1$
24: **else**
25: **return** $S_2$
26: **end if**

### *B. Nearest Neighbor Algorithm for TPP (NN-TPP)*

We now specify how the value $\hat{c}(S)$ can be calculated. The value $\hat{c}(S)$ is an estimate of the minimum possible end-to-end latency $c(S)$ for a given combination $S$ of models. Hence, if $c(S)$ can be computed, the value $\hat{c}(S)$ in Algorithm VI-A can be replaced with the minimum value $c(S)$. The value $c(S)$ can indeed be obtained by solving the integer linear program (ILP) which is derived from the formulation (7)-(11) by fixing the variables $x_m$ according the combination $S$. The objective function in this case is the left hand side of (8), and consequently, the problem is to find a computation path for computing the ensemble $S$ with minimum latency. Algorithm VI-A with this ILP applied for computing $c(S)$ is denoted by Greedy+ILP.

However, it can take excessively long time to solve such an ILP especially for large scenarios. We thus develop a heuristic algorithm based on the greedy algorithm for traveling salesman problem (TSP). Note that in the classical TPP or TSP, the source and destination are identical, however it is not necessarily true in our problem. Accordingly, we modify the greedy algorithm for TSP. The key idea of our algorithm is to iteratively add the closest model, defined as the one that incurs the least transmission and computation delay if selected. The process begins with the source node as the initial reference point, and subsequently, the current node is updated to the location where the most recently selected model is computed. The details of the algorithm are presented in Algorithm VI-B where $T_{u,v}$ and $C_{m,v}$ are the smallest transmission delay from node $u$ to $v$ and the computation delay of model $m$ at node $v$. Algorithm VI-A with Algorithm VI-B for computing $\hat{c}(S)$ is denoted by Greedy+NN-TPP.

**Algorithm 2** NN-TPP

**Init:** $i = 1$, $v_0 = s$, $\mathcal{M}' = \mathcal{M}$;
**while** $\mathcal{M} \neq \emptyset$ **do**
$(v_i, m_i) = \arg \min_{v \in \mathcal{V}_m, m \in \mathcal{M}} T_{v_{i-1},v} + C_{m,v}$
$\mathcal{M}' = \mathcal{M}' \setminus \{m_i\}$
$i = i + 1$
**end while**
Add shortest path from $v_{i-1}$ to $t$
**Output:** $\{v_0, P_{v_0,v_1}, (v_1, m_1), ..., (v_M, m_M), P_{v_M,t}, t\}$

## VII. Numerical and Simulation Results

We evaluate the performance of our algorithms under various environments, and show that our algorithms can effectively find a model ensemble with high accuracy while satisfying a given latency constraint.

### *A. Model Ensemble based on Negative Sample Set*

We first show that negative sample set accuracy serves as a reliable metric, providing a strong indication of test accuracy. The experiments are conducted using a pool of ten DNN models presented in Table I, and hence, we have $M = 10$.

To train and evaluate DNN models, we use CIFAR-100 as a benchmark dataset [27]. The negative sample set $\mathcal{D}^m$ of each model $m$ is constructed by sampling 100 out of all the samples misclassified by model $m$. The ensemble of models is obtained by solving the MILP (3)-(6).

TABLE I
MODEL POOL FOR ENSEMBLE

| Model ID | Models | Accuracy (%) | GFLOPs |
|---|---|---|---|
| 1 | Mobilenet | 64.66% | 0.095 |
| 2 | Mobinenetv2 | 66.75% | 0.133 |
| 3 | Squeezenet | 67.96% | 0.108 |
| 4 | Shufflenet | 69.62% | 0.088 |
| 5 | Shufflenetv2 | 67.89% | 0.089 |
| 6 | Resnet18 | 74.08% | 1.113 |
| 7 | Resnet34 | 74.75% | 2.322 |
| 8 | Densenet121 | 76.66% | 1.791 |
| 9 | Googlenet | 73.88% | 1.068 |
| 10 | Nasnet | 76.14% | 1.339 |

As shown in Figure 3, negative sample set accuracy and actual test accuracy are strongly correlated with the correlation coefficient of 0.87. We also compare our MILP method with some diversity metrics including Cohen's Kappa (CK) [28] and Kohavi-Wolpert Variance (KW) [29]. We select the model combination having the highest diversity metric. As shown in Table 2, our ensemble method based on negative sample set accuracy outperforms the other methods based on diversity metrics existing in the literature. This result demonstrates that negative sample set accuracy is an appropriate metric for model ensemble.

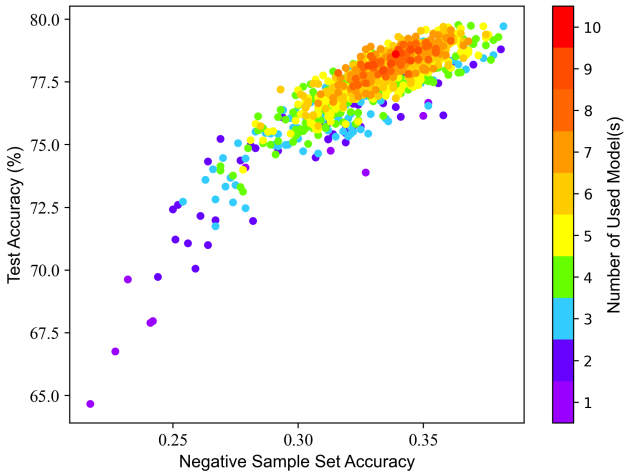

Fig. 3. Negative Sample Set Accuracy (CIFAR-100)

TABLE II
COMPARISON OF MILP WITH EXISTING DIVERSITY METRICS

| Methods | Selected Models | Test Acc(%) | NegSampSet Acc |
|---|---|---|---|
| CK | 2, 8 | 76.98% | 0.329 |
| KW | 1,2,3,4,5,7,8,9,10 | 78.34% | 0.345 |
| MILP | 8, 9, 10 | **79.71%** | 0.382 |

### *B. Optimal Ensemble on Distribued DNN Models*

Next, we conduct numerical evaluations in the topology shown in Fig. 4. All the links are bidirectional, and unless otherwise specified, the transmission rate of each link is 577.6Mbps. In this scenario, the data rates of all the links incident to red nodes $C$ and $D$ are scaled down by a factor of 0.2 in order to diversify the transmission delay. Hence, visiting nodes $C$ or $D$ for ensemble may incur large delay due to slow links, although $C$ and $D$ have high-accuracy models. Every node has the same computing capacity of 472 GFLOPS, which is the same capacity as Nvidia Jetson Nano. Since the input data is the image in CIFAR-100 dataset with 100 classes, we have $d_0 = 3072$ bytes of 224x224x3 RGB image, $d_M = 400$ bytes of 100-dimensional probability vector and $d_l = 3472$ bytes of image plus probability vector for $0 < l < M$. Recall that the computational demand of each model is presented in Table I. We compare the performance of MILP in (7)-(11), Greedy+ILP and Greedy+NN-TPP.

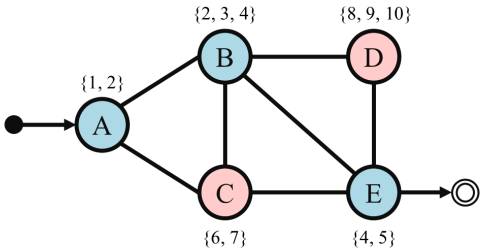

Fig. 4. 5-Node Topology

Figure 5 shows the ensembled accuracy of each method for different latency budgets $\delta$. Note first that the negative sample set accuracy of ensembled models is non-decreasing as the latency budget increases. This is because an increasingly accurate ensemble becomes feasible as the latency constraint becomes loose. The test accuracy exhibits similar tendency to negative sample set accuracy. Note, however, that in some cases, the test accuracy drops even with enhanced negative sample set accuracy. This shows that better negative sample set accuracy does not necessarily translate to better test accuracy, although for most of the cases, the negative sample set accuracy serves as a reliable proxy for test accuracy.

Next, all the link capacities are scaled down by a factor of 0.0001, so as to create the large transmission latency regime. Fig. 6 shows similar results to those in Fig. 5. It should be noted that in some cases, NN-TPP fails to find a path that satisfies the latency constraint, which can inherently occur under greedy algorithms such as NN-TPP. Nonetheless, these results show that all of our methods can effectively find a good combination of models to achieve better accuracy while satisfying latency constraint.

We also compare the runtime of each method in Table III. MILP is a time-consuming method even though it finds an optimal solution of the problem. On the other hand, other heuristic algorithms find a solution relatively quickly, with comparable performance to the optimal MILP. Hence, in

practice especially with large scenarios, our heuristics can be a reasonable alternative to MILP.

### C. Simulation Results

While our algorithm is designed for the static scenario where there is only a single inference task to be executed, we test our algorithm in the dynamic scenario where inference tasks are generated periodically. We use the packet-level simulator OMNeT++ [30]. The topology and model placement is the same as Fig. 4, except that models 3 and 10 are additionally placed in node $C$. The computing capacity of each node is set to 20GFLOPS, and the link bandwidth is between 10Mbps and 1Gbps. The latency constraint $\delta$ is fixed at 0.157. In the beginning of simulation, each inference task takes the same path precomputed by each method. However, relying on a single path may lead to increased queueing delays, either at the computing nodes or along the communication links. To address this, we instruct each algorithm to recompute the path every 20 seconds, taking the current compute queue status into account.

Figure 7 shows the end-to-end latency and accuracy of each task which arrives for every 0.067s. At the beginning of the simulation, each method attempts to achieve the highest possible accuracy. Since the Greedy+NN-TPP method fails to discover certain paths identified by other methods, it selects a path that yields lower accuracy compared to the others. After 20 seconds, all methods select the same model combination to avoid computation on nodes with large compute queues. MILP and Greedy+ILP decide to compute all selected models on the same node, while Greedy+NN-TPP distributes the models across different nodes. Notably, even though all methods achieve the same accuracy between 20 s and 40 s, the end-to-end latency of Greedy+NN-TPP tends to be lower than that of the others. When there is no accumulated compute queue, NN-TPP exhibits higher latency due to additional data transmission time. However, in scenarios where inference tasks continuously arrive, computing multiple models on the same node can lead to significant delays due to long queueing times. These results highlight the importance of effectively distributing the computing workload across multiple nodes, which we leave as future study. Overall, all methods successfully update the path every 20 seconds in order to detour congested resources.

Next, in Fig. 8, we additionally consider the runtime of each proposed algorithm. We assume that the runtime of MILP is 10 s, Greedy+ILP is 1 s, and Greedy+NN-TPP is 0.5 s, reflecting the relative scale observed in empirical measurements. Given its longer runtime, MILP may be less suitable for adapting to dynamically changing network environments.

## VIII. Conclusion and Future Work

In this paper, we introduced a novel problem of ensembling distributed deep neural network (DNN) models. We proposed a framework for model ensemble based on a mixed integer linear programming (MILP) formulation. Building on this MILP, we formulated an optimization problem to jointly find a combination of models and computation path for ensembled

TABLE III
Runtime of proposed methods to solve Optimal Ensemble of Distributed DNN Models (B=0.01, computing burden scenario)

| Methods | Runtime (s) |
|---|---|
| MILP | 197.5 |
| Greedy + ILP | 34.17 |
| Greedy + NN-TPP | 14.96 |

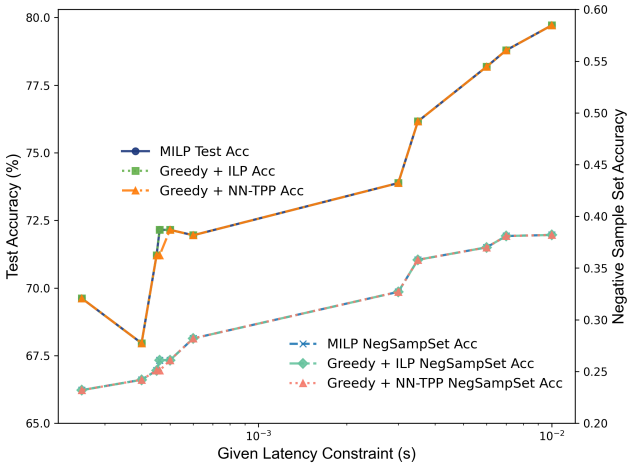


Fig. 5. Results in the large computation delay regime

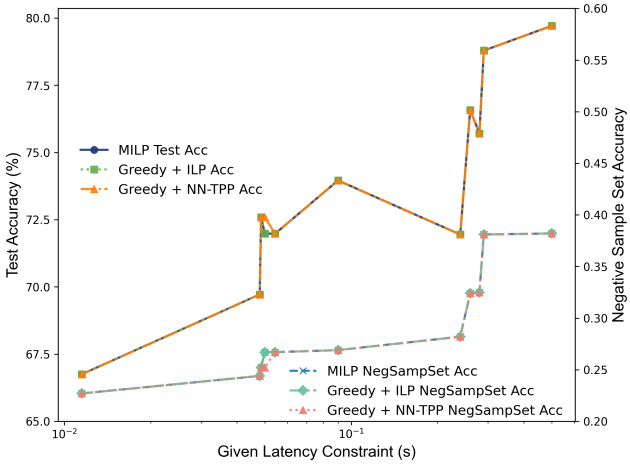


Fig. 6. Results in the large transmission delay regime

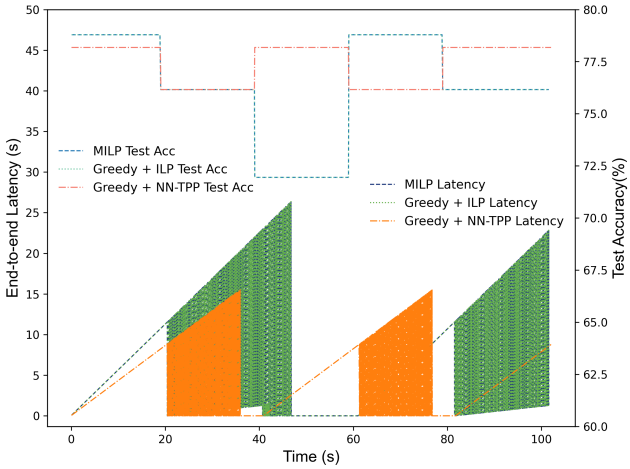


Fig. 7. Latency and accuracy of each method (w/o consideration of runtime)

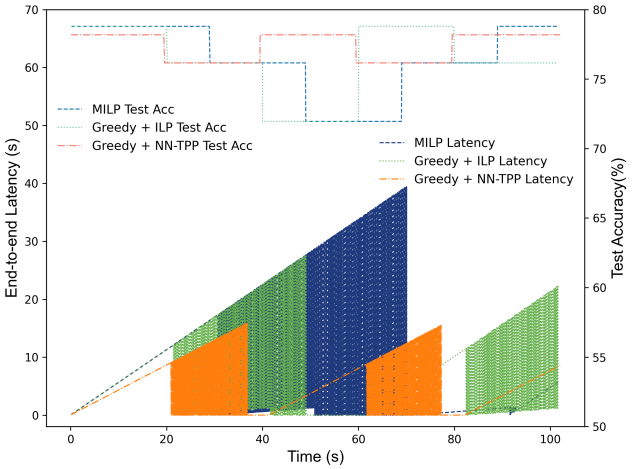


Fig. 8. Latency and accuracy of each method (with consideration of runtime)

models. In order to alleviate the difficulty of the problem, we also presented several heuristic algorithms. Our algorithms can effectively find a model ensemble with high accuracy while meeting a given latency constraint. We believe that the problem defined in this paper serves as a foundational step toward enabling resource-constrained end devices to enjoy AI services requiring heavy computation.

In this work, we focused on the scenario with a single service composed of single source-destination pair. While our algorithms for the single-service scenario can be readily used as a building block for addressing multi-service scenarios, as observed in simulations, it is worth developing algorithms tailored to the scenarios with multiple services featuring dynamic arrival and departure. As the general problem in such a scenario tends to be too complicated to develop useful insights, it would be interesting to study simple but more realistic scenarios such as end-edge-cloud environments. Such a scenario would not only enable mathematical tractability for developing algorithms but also provide insights for designing paradigms of distributed inference. Furthermore, if the edge server is located at a cellular base station or WiFi access point (which is a likely scenario in practice), it is important to address the scheduling problem for optimal distributed inference. We believe that these problems are of fundamental importance in the future network such as 6G which is expected to support diverse AI services while mitigating resource constraints.

## REFERENCES


[1] B. Rong, "6g: The next horizon: From connected people and things to connected intelligence," *IEEE Wireless Communications*, vol. 28, no. 5, pp. 8–8, 2021.

[2] B. Lin, Y. Huang, J. Zhang, J. Hu, X. Chen, and J. Li, "Cost-driven offloading for dnn-based applications over cloud, edge, and end devices," *IEEE Trans. on Industrial Inform.*, vol. 16, no. 8, pp. 5456–5466, 2020.

[3] X. Wang, Y. Han, V. C. M. Leung, D. Niyato, X. Yan, and X. Chen, "Convergence of edge computing and deep learning: A comprehensive survey," *IEEE Comms. Surv. & Tut.*, vol. 22, no. 2, pp. 869–904, 2020.

[4] S. Jung and H.-W. Lee, "Optimization framework for splitting dnn inference jobs over computing networks," *Computer Networks*, vol. 232, p. 109814, 2023.

[5] S. Kim, S. Jung, and H.-W. Lee, "Distributed computation of dnn via drl with spatiotemporal state embedding," *IEEE Internet of Things Journal*, vol. 11, no. 7, pp. 12 686–12 701, 2024.

[6] W. Wu, P. Yang, W. Zhang, C. Zhou, and X. Shen, "Accuracy-guaranteed collaborative dnn inference in industrial iot via deep reinforcement learning," *IEEE Trans. on Industrial Inform.*, vol. 17, no. 7, pp. 4988–4998, 2021.

[7] Y. Bahri, E. Dyer, J. Kaplan, J. Lee, and U. Sharma, "Explaining neural scaling laws," *Proceedings of the NAS*, vol. 121, no. 27, 2024.

[8] L. I. Kuncheva and C. J. Whitaker, "Measures of diversity in classifier ensembles and their relationship with the ensemble accuracy," *Machine Learning*, vol. 51, pp. 181–207, 2003.

[9] Y. Wu, L. Liu, Z. Xie, K.-H. Chow, and W. Wei, "Boosting ensemble accuracy by revisiting ensemble diversity metrics," in *CVPR*, 2021, pp. 16 469–16 477.

[10] S. Disabato, M. Roveri, and C. Alippi, "Distributed Deep Convolutional Neural Networks for the Internet-of-Things," *IEEE Transactions on Computers*, vol. 70, no. 8, pp. 1239–1252, 2021.

[11] E. Di Pascale, I. Macaluso, A. Nag, M. Kelly, and L. Doyle, "The Network As a Computer: A Framework for Distributed Computing Over IoT Mesh Networks," *IEEE Internet of Things Journal*, vol. 5, no. 3, pp. 2107–2119, 2018.

[12] R. Stahl, Z. Zhao, D. Mueller-Gritschneder, A. Gerstlauer, and U. Schlichtmann, "Fully distributed deep learning inference on resource-constrained edge devices," in *SAMOS*, 2019.

[13] W. He, S. Guo, S. Guo, X. Qiu, and F. Qi, "Joint dnn partition deployment and resource allocation for delay-sensitive deep learning inference in iot," *IEEE Internet of Things Journal*, vol. 7, no. 10, pp. 9241–9254, 2020.

[14] W. Fan, L. Gao, Y. Su, F. Wu, and Y. Liu, "Joint dnn partition and resource allocation for task offloading in edge–cloud-assisted iot environments," *IEEE Internet of Things Journal*, vol. 10, no. 12, pp. 10 146–10 159, 2023.

[15] Y. Huang, X. Qiao, W. Lai, S. Dustdar, J. Zhang, and J. Li, "Enabling dnn acceleration with data and model parallelization over ubiquitous end devices," *IEEE Internet of Things Journal*, pp. 1–1, 2021.

[16] M. Xue, H. Wu, G. Peng, and K. Wolter, "Ddpqn: An efficient dnn offloading strategy in local-edge-cloud collaborative environments," *IEEE Transactions on Services Computing*, vol. 15, no. 2, pp. 640–655, 2022.

[17] C. Zhang, W. Song, Z. Cao, J. Zhang, P. S. Tan, and X. Chi, "Learning to dispatch for job shop scheduling via deep reinforcement learning," in *NeurIPS*, vol. 33, 2020, pp. 1621–1632.

[18] J. Li, W. Liang, Y. Li, Z. Xu, X. Jia, and S. Guo, "Throughput maximization of delay-aware dnn inference in edge computing by exploring dnn model partitioning and inference parallelism," *IEEE Transactions on Mobile Computing*, vol. 22, no. 5, pp. 3017–3030, 2023.

[19] H. Li, X. Li, Q. Fan, Q. He, X. Wang, and V. C. Leung, "Distributed dnn inference with fine-grained model partitioning in mobile edge computing networks," *IEEE Transactions on Mobile Computing*, 2024.

[20] Y. Chen, L. T. Yang, and Z. Cui, "Tensor-based lyapunov deep neural networks offloading control strategy with cloud-fog-edge orchestration," *IEEE Transactions on Industrial Informatics*, 2023.

[21] J.-A. Lim and Y. Kim, "Real-time dnn model partitioning for qoe enhancement in mobile vision applications," in *IEEE SECON*, 2022.

[22] V. Sohn, S. Kim, and H.-W. Lee, "Joint frame drop and object detection task offloading for mobile devices via rl with lyapunov optimization," *IEEE Transactions on Mobile Computing*, pp. 1–15, 2025.

[23] T. Tan and G. Cao, "Deep learning video analytics through edge computing and neural processing units on mobile devices," *IEEE Transactions on Mobile Computing*, vol. 22, no. 3, pp. 1433–1448, 2023.

[24] W. Zhang, D. Yang, H. Peng, W. Wu, W. Quan, H. Zhang, and X. Shen, "Deep reinforcement learning based resource management for dnn inference in industrial iot," *IEEE Transactions on Vehicular Technology*, vol. 70, no. 8, pp. 7605–7618, 2021.

[25] C. Ju, A. Bibaut, and M. v. d. Laan, "The relative performance of ensemble methods with deep convolutional neural networks for image classification," *Jrn. of Applied Statistics*, vol. 45, pp. 2800–2818, 2017.

[26] M. Sviridenko, "A note on maximizing a submodular set function subject to a knapsack constraint," *ORL*, vol. 32, no. 1, pp. 41–43, 2004.

[27] A. Krizhevsky, "Learning multiple layers of features from tiny images," 2009.

[28] M. L. McHugh, "Interrater reliability: the kappa statistic," *Biochemia Medica*, vol. 22, no. 3, pp. 276–282, 2012.

[29] R. Kohavi and D. H. Wolpert, "Bias plus variance decomposition for zeroone loss functions," in *ICML*, 1996.

[30] A. Varga and R. Hornig, "An overview of the omnet++ simulation environment," in *Simutools*, 2008, pp. 1–10.