# DIME: Distributed Inference Model Estimation for Minimizing Profiled Latency

Robert Viramontes
*Department of Electrical and Computer Engineering*
*University of Wisconsin - Madison*
Madison, WI
rviramontes@wisc.edu

Azadeh Davoodi
*Department of Electrical and Computer Engineering*
*University of Wisconsin - Madison*
Madison, WI
adavoodi@wisc.edu

2024 IEEE International Conference on Smart Computing (SMARTCOMP) | 979-8-3503-4994-8/24/$31.00 ©2024 IEEE | DOI: 10.1109/SMARTCOMP61445.2024.00081

***Abstract*—Distributed inference allows minimizing metrics such as latency by offloading some computations from an edge device. It is commonly formulated and solved as an Integer Linear Program (ILP) for layer-wise partitioning of a Deep Neural Network (DNN) to decide transition points from an edge device to a hub and/or cloud devices. The formulation requires parameters reflecting latencies to execute different bundles of consecutive layers of DNN on each device. Profiling is the main way to measure these bundle latencies accurately on a device.**

**In this work, we show a recent ILP of the layer-wise partitioning (`JointDNN`) cannot in fact always generate an optimal solution. As we show, this happens due to profiling behavior seen in some devices. We propose `DIME` (Distributed Inference Model Estimation) with novel modifications to accurately estimate the latency of a bundle within the ILP formulation. It guarantees generating the optimal solution regardless of the type of device in the network. Additionally, `DIME` incorporates a new input parameter within the ILP to control the tradeoff between solution quality and the profiling effort. In our experiments we show solving `DIME` always results in the optimal solution, sometimes with significantly less profiling effort.**

***Index Terms*—neural networks, distributed inference, optimization, profiling, heterogeneous systems**

## I. Introduction

In recent years, there has been a rapid growth in applications for Deep Neural Networks (DNNs) as they have shown adept performance at a variety of tasks. As DNNs are utilized in more applications, they are being deployed in a variety of environments, including on edge devices which have a low computational capability. One technique to balance latency requirements of complex DNNs on non-performant edge devices is to distribute the inference over multiple devices, factoring in communication latencies between them. In the most common distributed inference model, the DNN layers are partitioned to decide a transition point from edge to faster devices such as hub and cloud. A recent survey on distributed inference highlights the need for improved partitioning algorithms that can adapt to compute and communication resources [1].

A variety of techniques have been proposed for layer-wise partitioning of a DNN to determine transition points while optimizing for objectives such as latency or energy. In `JALAD` [2], the profiled computation latency is fed as input to an integer linear program (ILP) which minimizes the total latency of edge and cloud. `JALAD` compresses data transfers which may result in accuracy loss. In `ADDA` [3], device profiling is used with greedy search to generate multi-path DNNs. Only the early-exiting portion of the DNN is executed on the edge.

The ILP proposed in [4] uses an analytic model to estimate latencies while considering a weight pre-loading optimization to overlap compute and weight memory transfers in the inference schedule. Recently, `JointDNN` [5] utilizes device latency profiles and formulates the edge-cloud partitioning problem as an ILP, optimizing latency and energy for both inference and training.

These ILP formulations rely on latency models of individual layers or models of a *bundle* of consecutive layers to explore the solution space to find the best transition points. It's been shown that profiles of individual layers are not sufficient to estimate the latency of a bundle (i.e., as sum of individual layers) [5]. In fact, `JointDNN` is one of the only works that incorporates latency profiles of all possible layer bundles of a DNN for each device as parameters fed into the ILP.

**In this work**, we first make the key observation that despite collecting all possible latency profiles, `JointDNN` is still susceptible to generating sub-optimal solutions. This is due to the profiling behavior exhibited by some devices, which can cause the ILP of `JointDNN` to incorrectly select profiles used for latency estimation by its optimization variables. In particular, for devices whose smaller-bundle latencies underestimate larger-bundle latencies, `JointDNN` *incorrectly assigning a larger-sized layer bundle to execute on a device based on an underestimated latency of smaller-sized bundles, when minimizing a distributed latency objective.*

Our work is motivated by the above novel observations. Specifically, using the above observations, we make the following contributions:

- We first generalize the formulation of `JointDNN` for arbitrary number of devices which we refer to as the 'base' ILP. We then propose `DIME`, which incorporates novel modifications to this base ILP and guarantees correct estimation of latency of layer bundles within the optimization formulation, regardless of device type.
- We additionally introduce an input parameter called Maximum Bundle Size (MBS) to our formulation. It

This material is based upon work supported by the National Science Foundation under Grant No. 2006394.

2693-8340/24/$31.00 ©2024 IEEE
DOI 10.1109/SMARTCOMP61445.2024.00081

controls the tradeoff between the device profiling effort and quality of solution (i.e., distributed latency) in DIME.
- Our simulation results conducted for a range of network communication bandwidths show DIME always generates the optimal solution, unlike JointDNN.
- For some network bandwidths, DIME finds an optimal solution with significantly lower profiling effort when using a small MBS (e.g., 22 versus 132 device profiles).

In the remainder of the paper, in Section II-A we first propose the 'base' ILP formulation. We then elaborate the issue related to latency profiles of different devices in Section II-B and finally propose the DIME additions to the base in Section II-C. Simulation results are presented in Section III.

## II. Our Approach

Consider a distributed inference task to be sequential execution of the layers of a DNN across networked devices. The problem has been formulated as an ILP which assigns each layer to a device to be computed on. We focus on a distributed inference latency objective, which is relevant, for example, in the context of maximizing the user's experience to receive the results of an inference in a timely manner.

### A. Base ILP Formulation

Let us denote $d \in S_D$ to be a device from set of $D$ devices in the network, and $\ell \in S_L$ be a layer of the set of $L$ layers of a DNN. Let the constant parameter $t^c_{i,j,d}$ denote the latency to compute a *bundle* of layers $i$ through $j$ on device $d$. This parameter should be computed by profiling on each device, for all combinations of bundles with $1 \leq i \leq j \leq L$.

Denote $t^x_{i,d,d'}$ to be the constant parameter reflecting the network communication latency to send the output of layer $i$ from device $d$ to compute the next layer on device $d'$. Assume $t^x_{i,d,d'} = 0$ when $d = d'$. In our experiments, these parameters are estimated by dividing the total data size in bytes for a given bandwidth (in bytes per second) from $d$ to $d'$.

Next, three groups of binary variables are defined:

- $x_{\ell,d}$: This variable encodes the layer-wise partitioning solution. It is 1 when layer $\ell$ is assigned to device $d$, 0 otherwise.
- $y_{i,j,d}$: This binary variable is 1 when a bundle of consecutive layers $i$ through $j$ is assigned to device $d$. When $i = j$, the bundle includes only one layer. Having $y_{i,j,d} = 1$ does *not* mean that consecutive layers $i$ to $j$ are the largest bundle that is assigned to device $d$; layers immediately before $i$ or after $j$ may also be assigned to device $d$.
- $z_{\ell,d,d'}$: This binary variable is 1 when layer $\ell$ is executed on device $d$ and layer $\ell + 1$ is executed on device $d'$. It decides when the network communication latency should be added toward computing the distributed latency.

The base ILP is formulated using five groups of constraints.

Constraint (1), below, is written for each layer. It ensures each layer is assigned to exactly one device:

$$\sum_{d=1}^{D} x_{\ell,d} = 1 \quad \forall \ell \in S_L \tag{1}$$

Next, Constraint (2) sums all unique $y_{i,j,d}$ bundle variables which include layer $\ell$ in them ($1 \leq i \leq \ell \leq j \leq L$) and sets the $x_{\ell,d}$ variable:

$$x_{\ell,d} = \sum_{\forall i,j \mid 1 \leq i \leq \ell \leq j \leq L} y_{i,j,d} \quad \forall d \in S_D; \forall \ell \in S_L \tag{2}$$

This constraint is written for all combinations of devices $d \in S_D$ and layers $\ell \in S_L$. It ensures that each layer is associated with only one non-zero $y_{i,j,d}$ variable, *if* it is assigned to device $d$.

The corresponding latency profile of layers $i$ to $j$ on device $d$ (i.e., $t^c_{i,j,d}$ parameter) would then be multiplied by $y_{i,j,d}$ toward computing the latency of execution on device $d$.

Next, constraint (3) shows how latency of execution on each device and the overall compute latency across all devices ($T^c$) is estimated:

$$T^c = \sum_{d=1}^{D} \Big( \sum_{i=1}^{L} \sum_{j=i}^{L} t^c_{i,j,d} \times y_{i,j,d} \Big) \tag{3}$$

In Constraint (3) all possible layer bundles $(i, j)$ with ($1 \leq i \leq j \leq L$) are listed for each device $d$. The $y$ variable is multiplied by the corresponding $t^c$ parameter reflecting the corresponding compute latency to execute the bundle on that device. The expression inside the parenthesis expresses the total compute latency on device $d$.

Finally, to compute the communication latency, the two constraints below are added:

$$0 \leq x_{\ell,d} + x_{\ell+1,d'} - 2z_{\ell,d,d'} \leq 1 \qquad \forall \ell \in S_{L-1}; \forall d \neq d' \in S_D \tag{4}$$

Constraint (4) is written for all combinations of devices in $S_D$ and layers in $S_{L-1}$, where $S_{L-1}$ is the set of all the layers except the last one. It sets $z_{\ell,d,d'}$ when layer $\ell$ executes on device $d$ and the outputs are sent to device $d'$. So $z_{\ell,d,d'} = 1$ only when $x_{\ell,d} = x_{\ell+1,d'} = 1$. This constraint (consisting of two inequalities) expresses a logical AND operation.

Using Constraint (4), and pre-known communication latency parameters $t^x_{i,d,d'}$, the total communication latency $T^x$ is calculated as:

$$T^x = \sum_{d=1}^{D} \sum_{d'=1}^{D} \sum_{i=1}^{L-1} t^x_{i,d,d'} \times z_{i,d,d'} \tag{5}$$

Using all the presented constraints, the optimization objective is written as: $\min \; T^c + T^x$. This expression minimizes the distributed inference latency as the sum of all communication and computation latency variables across all assigned devices.

The above base ILP is a general form of the inference ILP in JointDNN [5], which explicitly defines separate notations for edge and cloud device types. In contrast, our base ILP is written with fewer notations, incorporates an arbitrary set of devices, and defines fewer groups of variables. We note that [5] conducts experiments with asymmetric bandwidth between devices (i.e. $t^x_{i,d,d'} \neq t^x_{i,d',d}$). Our formulation supports this, though for simplicity we assume symmetric bandwidth.

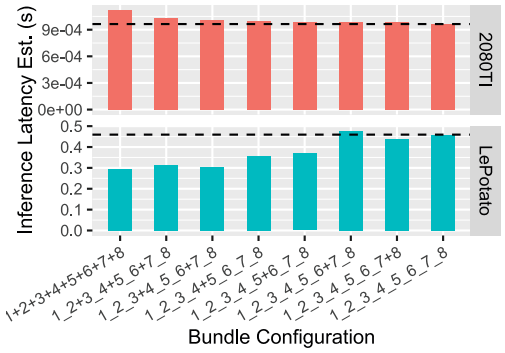


Fig. 1. Example showing inference latency of AlexNet when estimated as sum of different profiled bundles for two different devices. A bundle configuration of '1_2+3' would indicate the sum of profiled latency for a bundle of layers (1,2) with the profiled latency of layer 3. The right-most bar (bundle of 1_2_3..._8) is the accurate latency profiling all layers as one bundle.

In the presented 'base' ILP, similar to all other ILP-based partitioning works such as [2], [4], [5], we assume the devices in the network are (a) aware of the mapping of layers to devices and (b) have implemented a communication protocol where the receiving intermediate results triggers computation and forwarding results to the next device.

### B. Impact of Device-Based Latency Profiles

The base ILP formulation requires latency parameters $t^c_{i,j,d}$ corresponding to a bundle of consecutive layers (from $i$ to $j$) to be specified by profiling execution on each device $d$. Profiles should be collected for all unique combinations of $i$ and $j$ with $1 \leq i \leq j \leq L$.

It is possible for the base ILP to generate solutions based on layer $x_{l,d}$ assignments that a bundle has its latency computed using interior bundles of smaller sizes. For instance, two different assignments $y_{1,2,d} = y_{3,4,d} = 1$ versus $y_{1,4,d} = 1$ both indicate layers 1 through 4 are executed on device $d$. Therefore, $x_{\ell,d} = 1$ in both cases for $\ell = 1, \ldots, 4$. However, the computation latency on $d$ is estimated by the ILP as $t^c_{1,2,d} + t^c_{3,4,d}$ in the former case, and as $t^c_{1,4,d}$ in the latter. The ILP actually picks the assignment which minimizes its objective the most. For example it may assign $y_{1,2,d} = y_{3,4,d} = 1$ and $y_{1,4,d} = 0$ if $t^c_{1,2,d} + t^c_{3,4,d} < t^c_{1,4,d}$.

To demonstrate this, Figure 1 shows the latency of AlexNet network [6] when estimated using different combinations of layer bundles, for a representative subset of combinations. This is done for two devices: the NVIDIA 2080TI GPU and LibreComputer LePotato. In this example, the latencies are for non-distributed case and computed assuming only one of the two devices is used. We segment the DNN at the boundaries of convolution and fully-connected layers because the latencies of these layers dominate computation latency.

*To compute the profile of each $(i, j)$ bundle, we use the PyTorch [7] Sequential model to create a feed-forward bundle of layers $i$ through $j$. We profile 20 runs of this bundle, and take the average of the runs after discarding the top 10% and bottom 10% of observations to eliminate potential outliers. This is is done for all the experiments in the paper.*

In Figure 1, we observe that for the LePotato edge-class device, the sum of layers tends to significantly underestimate the execution latency of the whole network (right-most bar). The underestimation exists even when using larger interior bundles. We also observe that for the 2080TI cloud-class device, the sum often overestimates the latency of a bundle. The causes of under- and over-estimation behavior are outside the scope of this work, but our results support the claim in `JointDNN` [5] that larger bundles are more accurate and *add* that this is true in both under- and over-estimation cases.

Under the latency minimization scheme of the base ILP, using the LePotato profiles would result in setting, $y_{1,1,d}, y_{2,2,d}, \ldots, y_{8,8,d}$ variables to 1 instead of $y_{1,8,d}$ even though this selection significantly underestimates the true latency captured by $t^c_{1,8,d}$.

### C. From the Base ILP to `DIME`

To fix the discussed issue, we add the following constraint to the base ILP which is written for each device:

$$\sum_{i=1}^{L} \sum_{j=i}^{L} y_{i,j,d} \leq \lceil \frac{\sum_{\ell=1}^{L} x_{\ell,d}}{L} \rceil \quad \forall d \in S_D \text{[1]} \tag{6}$$

In the above inequality, the left-hand-side is sum of the binary variables $y_{i,j,d}$ on each device $d$ for all layer combinations $1 \leq i \leq j \leq L$. This right-hand-side numerator is the *number* of layers assigned to device $d$. The denominator is number of layers $L$ in the DNN. The expression on the right-hand-side will evaluate to 1 if at least one layer of the DNN is assigned to $d$ and 0 otherwise. The inequality therefore requires the sum of all $y_{i,j,d}$ variables assigned to device $d$ to be at most 1, *if* at least one layer is assigned to device $d$.

This forces the optimizer to set the $y_{i,j,d}$ variable with the smallest index for $i$ and largest index $j$, and the rest of the $y_{i,j,d}$ variables to 0. In other words, the $y_{i,j,d}$ that is set to 1 corresponds to the largest-sized bundle assigned to device $d$ which ensures the correct latency profile ($t^c$) is used for $d$.

Adding Constraint 6 fixes the issue identified by us which happens when 'underestimating' devices are present in the network. When 'overestimating' devices are present, the added constraint will not cause an issue and will still enforce selecting the correct latency parameter.

Constraint 6 guarantees the use of correct latency parametersunder the assumption of single-entry point to a device. We note, this assumption is also made in other prior works on ILP-based partitioning of DNN [2], [4], [5]. This is an acceptable assumption because often it is not practical to switch compute back to a slow (edge) device after switching to a faster (cloud) device when minimizing distributed inference latency.

Recall, the base ILP requires profiling each device $d$ to obtain the $t^c_{i,j,d}$ latency parameters for all combinations of $1 \leq i \leq j \leq d$ which equals $\frac{L \times (L+1)}{2} \times D$. The expression shows that the profiling effort quadratically grows with the

[1]Ceil is a non-linear operation that can be approximated in a solver by the procedure in [8]

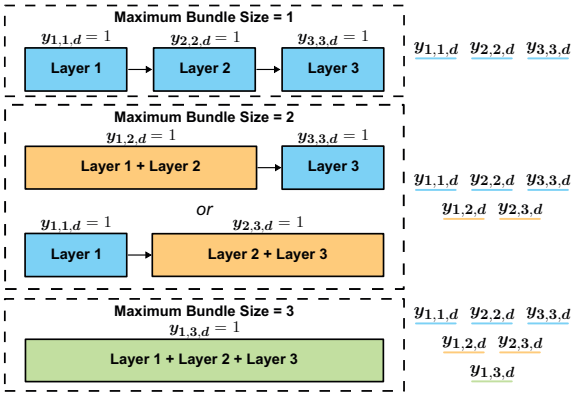


Fig. 2. Example showing the impact of incorporating an MBS parameter in Constraint 7. For simplicity, we assume there is only one device available. The example shows how the $y_{i,j,d}$ variables (shown on the right) are assigned to define layer bundles (shown on the left) for three MBS values. When MBS=3, the ILP's objective is accurately computed but has the most profiling effort.

number of layers in the DNN and linearly grows with the number of different devices in a heterogeneous network.

We introduce a new parameter called Maximum Bundle Size (MBS) to be incorporated as input to the ILP. It sets the maximum number of consecutive layers which may be assigned the same $y_{i,j,d}$ variable. Specifically, only $y_{i,j,d}$ variables are defined in which $i \leq j \leq i + MBS - 1$. For example, when MBS=1, we require $j = i$, indicating only bundle of 1 layer may be used. This means, for a given MBS, only a subset of $y_{i,j,d}$ variables may be defined compared to the base ILP. MBS allows the user to define a tradeoff between the device profiling effort and latency calculation accuracy.

To incorporate an MBS parameter, Constraint 6 may be generalized and replaced by the one below:

$$\sum_{i=1}^{L} \sum_{j=i}^{L} y_{i,j,d} \leq \lceil \frac{\sum_{\ell=1}^{L} x_{\ell,d}}{MBS} \rceil \quad \forall d \in S_D \tag{7}$$

In the above constraint, the right-hand-side expression computes an upper bound on the number of layer bundles that may be assigned to device $d$ to estimate its latency.

The case when there is no error in approximating Constraint 6 is when MBS=L (number of layers) which requires profiling all devices for all combinations of bundles $i$ through $j$ with $1 \leq i \leq j \leq L$. In this case, Constraint 7 simplifies to 6.

The other extreme case is when MBS=1. This case has the least profiling effort because only the $y_{i,j,d}$ variables with $i = j$ are defined. In this case, the only option that can be considered by the ILP is to select $L$ bundles of size 1 layer.

Figure 2 shows a toy example to demonstrate the impact of MBS for a neural network composed of 3 layers on one device. The figure shows the different solutions possible for MBS=1, MBS=2, and MBS=3. This demonstrates how the bundle selection varies with MBS under our novel constraint, impacting the latency calculation for the ILP. The profiling effort in each case is equal to the number of $y$ variables shown on the right-side of the figure.

## III. Experimental Results

We present our results utilizing the `DIME` formulation and compare with the base ILP presented in Section II-A. To compare with `JointDNN` [5] we used the same two-device setting (i.e., an edge and a cloud device). We utilize Gurobi [9] to solve the ILP formulation. We also compare with the optimal solution determined via exhaustive search of all feasible layer assignments and using the correct device latency profiles based on the generated solution.

We examine the results for two popular neural networks, AlexNet and VGG11, when varying the bandwidth of the connection between devices. For the edge-cloud setup in our experiments, we use the LePotato (as edge) and NVIDIA 2080TI (as cloud) devices. We additionally present results of an experiment for VGG11 when adding NVIDIA Jetson Nano as an intermediate hub device to the edge-cloud setup.

In our experiments, we also vary the input parameter Maximum Bundle Size (MBS) from 1 to number of layers ($L$). This demonstrates the the impact of MBS on trading off solution quality with profiling effort.

Communication latencies ($t^x_{i,d,d'}$) are estimated by dividing the total data size in bytes by the bandwidth (BW) in bytes per second, assuming symmetric bandwidth. They are integrated as shown in Constraints (4) and (5). We vary the network bandwidth over a range including 4G and WiFi, similar to the values in [5] and specified in Tables I and II.

In all of our experiments, we consider an application where the edge device collects data and requires the result of inference. For example, a smart camera that has to generate a local alert based on the results of inference. One such system is a real-time fire detection system [10], which requires the local alerts to have low latency and high accuracy to allow rapid fire response while minimizing false positives. We enforce these requirements by inserting zero-compute `input` and `output` pseudo-layers in the DNN and constraining the associated $x$ variable to be assigned to the edge device.

### A. Results of AlexNet for Edge-Cloud Setting

We first present the results for AlexNet. This neural network has five convolution and 3 fully-connected layers, for a total of 8 possible transition points under our profiling approach.

We experiment with three approaches: (1) `JointDNN` [5]; (2) `DIME` (ours); (3) optimal. The approach (1) is implemented using our base ILP given in Section II-A which simplifies to [5] given that we are also using an edge-cloud setting. For `DIME`, we add Constraint 7 to the base ILP and vary the MBS parameter. For the Optimal case (3), we exhaustively list all possible transitions from edge to cloud. We accurately evaluate the latency for each combination (considering both compute and communication) using the device latency profiles that exactly match the bundle assigned to device. We then pick the assignment with smallest overall distributed latency.

We show the results of our approach in Table I for different bandwidths when MBS is varied from 1 to 8 (number of layers in AlexNet), ommitting values of MBS with same results for brevity. For each network bandwidth two columns are listed

TABLE I
RESULTS OF OPTIMIZATION FOR ALEXNET, SHOWING TRANSITION LAYER FROM EDGE TO CLOUD (TRANS.) AND DISTRIBUTED LATENCY (LAT.).

| | | BW = 1E5 MBPS | | BW=7.3125E5 (4G Upload) | | BW=1E6 | | BW=2.36E6 (WiFi Upload) | | |
|---|---|---|---|---|---|---|---|---|---|---|
| Method | MBS | Trans. | $\text{Lat.}_{\text{est/act}}$ | Trans. | $\text{Lat.}_{\text{est/act}}$ | Trans. | $\text{Lat.}_{\text{est/act}}$ | Trans. | $\text{Lat.}_{\text{est/act}}$ | #Profiles |
| DIME | 1 | none | 0.433 / 0.456 | **fc1** | 0.233 / **0.275** | fc1 | 0.140 / 0.156 | **conv2** | **0.116 / 0.116** | 16 |
| | 4 | none | 0.482 / 0.456 | **fc1** | 0.250 / **0.275** | fc1 | 0.156 / 0.156 | **conv2** | **0.116 / 0.116** | 52 |
| | 5 | none | 0.477 / 0.456 | **fc1** | **0.275 / 0.275** | **conv3** | **0.184 / 0.184** | **conv2** | **0.116 / 0.116** | 60 |
| | 8 | **fc2** | **0.508 / 0.508** | **fc1** | **0.275 / 0.275** | **conv3** | **0.184 / 0.184** | **conv2** | **0.116 / 0.116** | 72 |
| JointDNN | | none | 0.433 / 0.456 | **fc1** | 0.233 / **0.275** | fc1 | 0.140 / 0.156 | **conv2** | **0.116 / 0.116** | 72 |
| **Optimal** | | **fc2** | **0.508** | **fc1** | **0.275** | **conv3** | **0.184** | **conv2** | **0.116** | 72 |

in the table: (1) layer name when transition from edge to cloud happens; (2) latency in seconds. For layer name, we indicate the transition point as the first layer executed on the cloud device, where 'none' means the cloud is not utilized. For latency, we report both estimated and actual latencies based on the generated solution of each ILP. The estimated latency is the value determined by the ILP objective expression; the actual latency is the sum of the largest bundles to implement the required transition points. By definition, 'optimal' and DIME with MBS = L report exactly the 'actual' latency.

The number of latency profiles is reported in the last column of the table. This value does not depend on BW so is only listed once per row. The number of profiles is same as the number of $t^c_{i,j,d}$ parameters that should be measured on both devices, and equals the number of $y_{i,j,d}$ variables defined in the ILP per device.

In the table, the bold entries indicate the cases when the estimated or actual latency is the same as the latency for the optimal solution under the given precision. We make the following observations from the table:

- For the highest MBS(=8), our approach always generates the same results as the optimal, across all bandwidths. This is solely due to adding Constraint 6 to the base ILP.
- As MBS decreases, the estimated latency becomes more erroneous as its gap with the actual latency grows. For example in bandwidth BW=1E5 for MBS=1 the error between the estimated and actual latencies is 0.456-0.433=0.023 which is the highest error in this bandwidth.
- Interestingly, for some higher bandwidths, the same transition layer *and* latency as the optimal row may be achieved for even smaller MBS. For instance, in the highest bandwidth (WiFi Upload), MBS=1 generates the same quality solution as the optimal approach.
- In the WiFi upload case, we achieve the same quality as the optimal solution for MBS=1 but significantly lower profiling effort (16 instead of 72 for two devices).
- JointDNN does not generate the optimal solution in two of the four bandwidths, while requiring the maximum number of profiles. For 4G, it coincidentally picks the correct transition despite underestimating latency.

We validate that the additional constraint and profiling approach in DIME always generates the optimal solution.

### B. Results of VGG11 for Edge-Cloud Setting

We also experimented with the VGG11 network [11] which has 8 convolution layers and 3 fully-connected layers, for a total of 11 possible transition points under our profiling scheme. Table II shows the results for this network. For brevity, we only show the results for odd values of MBS which is sufficient to observe the trends for the VGG11 network.

*The trends and conclusions are similar to the previous experiments.* In particular, we observe for the two highest bandwidths that an MBS=1 provides the same solution quality (in terms of transition layer and estimated latency) as in the optimal, while having a significantly lower number of profiles (22 for MBS=1 versus 132 in JointDNN and Optimal cases). With higher bandwidths, cloud resources become more accessible and all three optimization schemes pick a cloud-only (indicated by a conv1 transition) partition for the computation of the DNN layers. This is due to the significantly-lower computation latency of the cloud device compared to edge.

We clarify one counter-intuitive aspect of the results in the table. For BW=1E5 and BW=1.375E5, it appears that JointDNN generates a better solution under the estimated latency. However, this is because it is subject to the estimation error imposed by under-estimating devices. When evaluating the *actual* latency from the correct bundles, it becomes clear that it generates a worse solution.

Similar to AlexNet, our formulation guarantees selecting the optimal solution when MBS is set to the number of layers in the network, 11. JointDNN does not provide the same guarantee, especially at lower bandwidths that rely more on the edge device.

### C. Results of VGG11 for Edge-Hub-Cloud

We also consider an experiment with the same edge and cloud devices but adding the NVIDIA Jetson Nano board as an intermediate 'hub' device. The compute capability of the hub device is somewhere between edge and cloud devices. Latency profiles for all related bundles of layers are additionally collected on the hub device before running this experiment. The hub is added to the set of devices $S_D$ and the relevant parameters are included in the ILP.

TABLE II
RESULTS OF OPTIMIZATION FOR VGG11, SHOWING TRANSITION LAYER FROM EDGE TO CLOUD (TRANS.) AND DISTRIBUTED LATENCY (LAT.).

| | | BW = 1E5 MBPS | | BW=1.375E5 MBPS (3G Upload) | | BW=7.3125E5 MBPS (4G Upload) | | BW=1E6 MBPS | | |
|---|---|---|---|---|---|---|---|---|---|---|
| Method | MBS | Trans. | Lat.$_{est/act}$ | Trans. | Lat.$_{est/act}$ | Trans. | Lat.$_{est/act}$ | Trans. | Lat.$_{est/act}$ | #Profiles |
| DIME | 1 | none | 1.534 / 6.021 | none | 1.534 / 4.379 | **conv1** | **0.823 / 0.823** | **conv1** | **0.602 / 0.602** | 22 |
| | 3 | fc3 | 2.715 / 3.514 | fc3 | 2.670 / 3.469 | **conv1** | **0.823 / 0.823** | **conv1** | **0.602 / 0.602** | 60 |
| | 5 | none | 2.904 / 6.021 | none | 2.904 / 4.379 | **conv1** | **0.823 / 0.823** | **conv1** | **0.602 / 0.602** | 90 |
| | 7 | **fc2** | 3.185 / **3.451** | **fc2** | 3.140 / **3.406** | **conv1** | **0.823 / 0.823** | **conv1** | **0.602 / 0.602** | 112 |
| | 9 | none | 3.180 / 6.021 | none | 3.180 / 4.379 | **conv1** | **0.823 / 0.823** | **conv1** | **0.602 / 0.602** | 126 |
| | 11 | **fc2** | **3.451 / 3.451** | **fc2** | **3.406 / 3.406** | **conv1** | **0.823 0.823** | **conv1** | **0.602 / 0.602** | 132 |
| JointDNN | | none | 1.534 / 6.021 | none | 1.534 / 4.379 | **conv1** | **0.823 / 0.823** | **conv1** | **0.602 / 0.602** | 132 |
| **Optimal** | | **fc2** | **3.451** | **fc2** | **3.406** | **conv1** | **0.823** | **conv1** | **0.602** | 132 |

TABLE III
RESULTS OF VGG11 FOR EDGE→ HUB→ CLOUD SETTING.

| | LAN BW=1.375E5 MBPS Cloud BW=1E5 MBPS | |
|---|---|---|
| Method | Transition | Latency$_{est/act}$ |
| DIME (MBS=11) | **fc2 → Hub** | **3.418 / 3.418** |
| JointDNN | none | 1.534 / 3.533 |
| **Optimal** | **fc2 → Hub** | **3.418 / 3.418** |

A 'hub' device adds a higher-compute device as a *more local* resource. Communication from edge does not have to traverse, for instance, the public internet to reach a distant resource. We consider that edge and hub communicate via a local area network (LAN) that has higher bandwidth than either have to access the cloud over wide area network (WAN).

The solution for LAN bandwidth BW=1.375E5 and WAN bandwidth BW=1E5 is shown in Table III. The Transition column shows the layer when transition from edge to hub happens. We find that the introduction of the moderate-compute hub device, even with only moderately increased LAN bandwidth compared to WAN, allows reduction in latency without relying on access to cloud resources. DIME generates the same solution as in the optimal, while JointDNN assigns all layers to the edge device. This is because the LePotato edge device is an 'underestimating device', as explained in Section II-B, which results in JointDNN underestimating the latency of the edge device, and consequently assigning all layers to execute on the edge device when minimizing global latency.

## IV. CONCLUSIONS AND DISCUSSIONS

In this work, we examined a recent ILP for assigning the layers of a DNN to be executed on heterogeneous devices in a distributed environment, and found that it is prone to generating sub-optimal solutions depending on the characteristics of devices present in the network. We specifically identified 'underestimating devices' to be the root-cause of the issue. We then proposed DIME which incorporated novel modifications to the base ILP to always guarantee the optimal solution is found across all devices. We also show that our formulation allows extension beyond simple edge-cloud paritioning, but can address schemes that include multiple devices such as a local hub device.

## REFERENCES


[1] W.-Q. Ren, Y.-B. Qu, C. Dong, Y.-Q. Jing, H. Sun, Q.-H. Wu, and S. Guo, "A survey on collaborative DNN inference for edge intelligence," *Machine Intelligence Research*, vol. 20, no. 3, pp. 370–395, 2023.

[2] H. Li, C. Hu, J. Jiang, Z. Wang, Y. Wen, and W. Zhu, "JALAD: Joint Accuracy-And Latency-Aware Deep Structure Decoupling for Edge-Cloud Execution," in *IEEE International Conference on Parallel and Distributed Systems*, 2018, pp. 671–678.

[3] H. Wang, G. Cai, Z. Huang, and F. Dong, "ADDA: Adaptive Distributed DNN Inference Acceleration in Edge Computing Environment," in *IEEE International Conference on Parallel and Distributed Systems*, 2019, pp. 438–445.

[4] R. Viramontes and A. Davoodi, "Neural network partitioning for fast distributed inference," in *IEEE International Symposium on Quality Electronic Design*, 2023, pp. 1–7.

[5] A. E. Eshratifar, M. S. Abrishami, and M. Pedram, "JointDNN: An Efficient Training and Inference Engine for Intelligent Mobile Cloud Computing Services," *IEEE Transactions on Mobile Computing*, vol. 20, no. 2, pp. 565–576, 2021.

[6] A. Krizhevsky, I. Sutskever, and G. E. Hinton, "ImageNet classification with deep convolutional neural networks," in *Advances in Neural Information Processing Systems*, F. Pereira, C. J. C. Burges, L. Bottou, and K. Q. Weinberger, Eds., vol. 25, 2012.

[7] A. Paszke, S. Gross, F. Massa, A. Lerer, J. Bradbury, G. Chanan, T. Killeen, Z. Lin, N. Gimelshein, L. Antiga, A. Desmaison, A. Kopf, E. Yang, Z. DeVito, M. Raison, A. Tejani, S. Chilamkurthy, B. Steiner, L. Fang, J. Bai, and S. Chintala, "Pytorch: An imperative style, high-performance deep learning library," in *Advances in Neural Information Processing Systems 32*, H. Wallach, H. Larochelle, A. Beygelzimer, F. d'Alché-Buc, E. Fox, and R. Garnett, Eds. Curran Associates, Inc., 2019, pp. 8024–8035.

[8] S. Horn, "Linear program with ceiling or floor functions (how?)," https://support.gurobi.com/hc/en-us/community/posts/360054499471-Linear-program-with-ceiling-or-floor-functions-HOW- retrieved Sept. 16, 2023.

[9] Gurobi Optimization, LLC, "Gurobi Optimizer Reference Manual," 2021. [Online]. Available: https://www.gurobi.com

[10] S. Saponara, A. Elhanashi, and A. Gagliardi, "Real-time video fire/smoke detection based on CNN in antifire surveillance systems," *Journal of Real-Time Image Processing*, vol. 18, pp. 889–900, 2021.

[11] K. Simonyan and A. Zisserman, "Very deep convolutional networks for large-scale image recognition," 2014. [Online]. Available: https://arxiv.org/abs/1409.1556