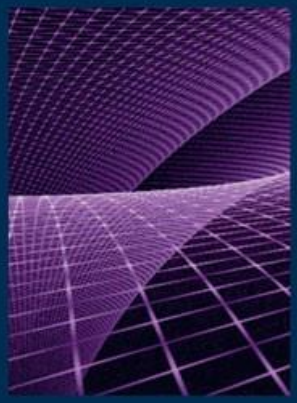


institute imdea networks

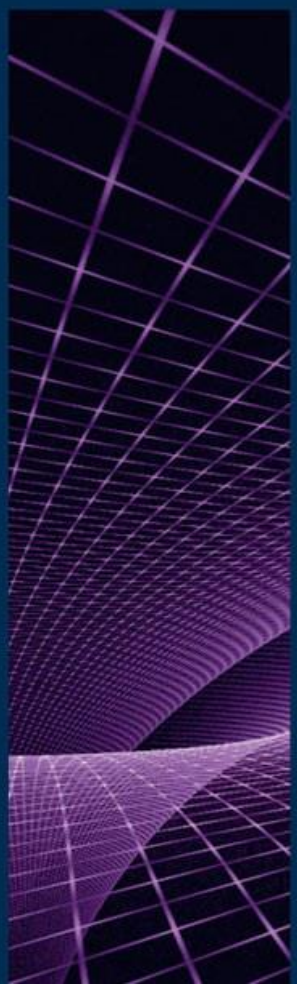
technical report

TR- IMDEA-Networks-2021-6

Offloading Algorithms for Maximizing Inference Accuracy on Edge Device Under a Time Constraint

Andrea Fresa

Jaya Prakash Champati



December 2021

Offloading Algorithms for Maximizing Inference Accuracy on Edge Device Under a Time Constraint

Andrea Fresa and Jaya Prakash Champati
IMDEA Networks Institute, Madrid, Spain
E-mail:{andrea.fresa,jaya.champati}@imdea.org

***Abstract*—With the emergence of edge computing, the problem of offloading jobs between an Edge Device (ED) and an Edge Server (ES) received significant attention in the past. Motivated by the fact that an increasing number of applications are using Machine Learning (ML) inference from the data samples collected at the EDs, we study the problem of offloading *inference jobs* by considering the following novel aspects: 1) in contrast to a typical computational job, the processing time of an inference job depends on the size of the ML model, and 2) recently proposed Deep Neural Networks (DNNs) for resource-constrained devices provide the choice of scaling down the model size by trading off the inference accuracy. Considering that multiple ML models are available at the ED, and a powerful ML model is available at the ES, we formulate a general assignment problem with the objective of maximizing the total inference accuracy of n data samples at the ED subject to a time constraint T on the makespan. Noting that the problem is NP-hard, we propose an approximation algorithm Accuracy Maximization using LP-Relaxation and Rounding (AMR^2), and prove that it results in a makespan at most $2T$, and achieves a total accuracy that is lower by a small constant from the optimal total accuracy. Further, if the data samples are identical we propose Accuracy Maximization using Dynamic Programming (AMDP), an optimal pseudo-polynomial time algorithm. As proof of concept, we implemented AMR^2 on a Raspberry Pi, equipped with MobileNets, that is connected to a server equipped with ResNet, and studied the total accuracy and makespan performance of AMR^2 for image classification.**

I. Introduction

Edge computing is seen as a key component of future networks that augments the computation, memory, and battery limitations of Edge Devices (EDs) (e.g., IoT devices, mobile phones, etc.), by allowing the devices to offload computational jobs to nearby Edge Servers (ESs) [1]. Since the *offloading decision*, i.e., which jobs to offload, is the key to minimizing the execution delay of the jobs and/or the energy consumption at the ED, it received significant attention in the past [2]. Recently, an increasing number of applications are using Machine Learning (ML) inference from the data samples collected at the EDs, and there is a major thrust for deploying pre-trained Deep Neural Networks (DNNs) on the EDs as this has, among other advantages, reduced latency. Thanks to the development of DNN models with reduced computation and storage requirements, possibly with reduced inference accuracy, and the advancements in the hardware of EDs [3], ML frameworks such as Tensorflow Lite [4] and PyTorch Mobile [5] are now able to support the deployment of DNNs on EDs. In this context, we study the offloading decision between an ED and an ES for the *inference jobs*, where an inference job refers to the execution of a pre-trained ML model on a data sample.

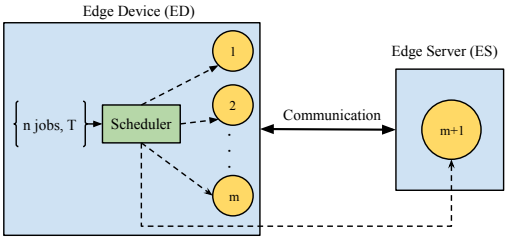


Fig. 1: Scheduling inference jobs between an ED and an ES.

In comparison to the fixed processing time requirement of a generic computational job (typically represented by a directed acyclic task graph), the processing time requirement of an inference job depends on the ML model size: a larger model size results in longer processing time and may provide higher inference accuracy. For example, on Pixel 3 smartphone, ResNet [6] has size 178 MB, requires 526 ms, and provides 76.8% accuracy (Top-1 accuracy) for the ImageNet dataset [7], while the smallest DNN model of MobileNet [8] has size 1.9 MB, requires 1.2 ms, but provides 41.4% accuracy [4]. Furthermore, recently developed DNNs for EDs allow for scaling the model size by simply setting a few hypeparameters (cf. [8]–[10]), enabling the EDs to choose between multiple model sizes. However, as we explain in Section II, the offloading decision for inference jobs considering the above novel aspects has received little attention in the literature.

Taking into account the novel aspects for inference jobs, or simply *jobs* in the sequel, we consider the system in Figure 1, where the ED has m ML models to choose from, and the ES is equipped with a state-of-the-art ML model for a given application. Consider that n jobs (corresponding to n data samples) are available at the ED. It may offload them all to the ES to maximize the inference accuracy. However, offloading each job incurs a communication time to upload the data sample in addition to the processing time at the ES. This may result in a large *makespan*, i.e., the total time to finish all the jobs. On the other hand, executing all the jobs on the smallest ML model at the ED may result not only in a smaller makespan, but also the lowest inference accuracy. Thus, a scheduler at the ED needs to strike a trade-off between the accuracy and the makespan. Toward this end, we formulate the following problem: *given n data samples at time zero, find*

a schedule that offloads a partition of the jobs to the ES and assigns the remaining jobs to m models on the ED, such that the total accuracy is maximized and the makespan is within a time constraint T. Solution to this problem will be beneficial to applications such as Google Photos where a set of photos selected by a user need to be classified into multiple categories in real time. Also, the problem has relevance to applications which do periodic scheduling, i.e., the ED periodically collects all the data samples arrived in a time period T and aims to finish their processing within the next time period T.

Since the true accuracy (Top-1 accuracy) provided by a model for a given data sample can only be inferred after the job is executed, for analytical tractability, we consider the average test accuracy of the model is its accuracy for any data sample. Given the processing and communication times of the jobs, we formulate the problem as an Integer Linear Program (ILP). We note that the ILP is agnostic to the actual ML models used on the ED and the ES. Different ML models on the ED may correspond to different instantiations of the same DNN with different hyperparameter values (cf. [8]), or they may correspond to different ML algorithms such as logistic regression, support vector machines, DNN, etc.

Solving the formulated ILP is challenging due to the following reasons. Partitioning the set of jobs between the ED and the ES is related to scheduling jobs on parallel machines [11], and assigning the jobs to the models on the ED is related to the knapsack problem [12], both are known to be NP-hard. A special case of our problem, where the ED has a single model ($m = 1$), is the Generalized Assignment Problem (GAP) with two machines [13]. GAP is known to be APX-hard, and the best-known approximation algorithm provides a solution that has maskespan at most $2T$ [14]. However, the algorithms for GAP and their performance guarantees are not directly applicable to our problem due to the additional aspect that, on the ED there are multiple models to choose from. We propose a novel algorithm that solves a Linear Programming relaxation (LP-relaxation) of the ILP, uses a counting argument to bound the number of fractional solutions, and solves a sub-ILP problem to round the fractional solution.

Our main contributions are summarized below:

- We formulate the total accuracy maximization problem subject to a constraint T on the makespan as an ILP. Noting that the ILP is NP-hard, we propose an approximation algorithm Accuracy Maximization using LP-Relaxation and Rounding (AMR^2) to solve it. The runtime of AMR^2 is $O(n^3(m+1)^3)$.
- We prove that the total accuracy achieved by AMR^2 is at most a small constant (less than 2), lower than the optimum total accuracy, and its makespan is at most $2T$.
- For the case of identical jobs, i.e., the data samples are identical, we propose an optimal algorithm Accuracy Maximization using Dynamic Programming (AMDP), which does a greedy packing for the ES and solves a Cardinality Constrained Knapsack Problem (CCKP) for job assignments on the ED.
- As proof of concept, we perform experiments using a Raspberry Pi, equipped with MobileNets, and a server, equipped with ResNet50, that are connected over a Local Area Network (LAN). Our application is image classification for the images from ImageNet. We estimate processing and communication times for different sizes of images, and implemented AMR^2 and a greedy algorithm on Raspberry Pi. Our results indicate that the total test accuracy achieved by AMR^2 is close to that of the LP-relaxed solution, and its total true accuracy is, on average, 40% higher than that of the greedy algorithm.

The rest of the paper is organized as follows. In Section II, we present the related work. The system model is presented in Section III. In Sections IV and V, we present AMR^2 and its performance bounds, respectively. In Section VII, we present the experimental results and finally conclude in Section VIII.

II. Related Works

In this section, we first present the related works for computation offloading problem and then discuss closely related classical job scheduling problems.

A. Offloading and ML Inference Jobs

Since the initial proposal of edge computing in [15], significant attention had been given to the computational offloading problem, wherein the ED needs to decide which jobs to offload, and how to offload them to an ES [2]. The objectives that were considered for optimizing the offloading decision are, 1) minimize the total execution delay of the jobs, see for example [16]–[18], and 2) minimize the energy of the ED spent in computing the jobs, subject to a constraint on the execution delay, see for example [19]–[21]. However, the above works consider generic computation jobs, and the aspect of accuracy, which is relevant for the case of inference jobs, has not been considered.

Recently, a few works considered the problem of maximizing accuracy for inference jobs on the ED [22]–[24]. In [22], the authors studied the problem of maximizing the accuracy within a deadline for each frame of a video analytics application. They do not consider offloading to the edge and their solution is tailored to the DNNs that use early exits [9]. Similar problem was studied in [23], where offloading between a mobile device and a cloud is considered. The authors account for the time-varying communication times by using model selection at the cloud and by allowing the duplication of processing the job a the mobile device. A heuristic solution was proposed in [24] for offloading inference jobs for maximizing inference accuracy subject to a maximum energy constraint. In contrast to the above works, we consider multiple models on the ED and provide performance guarantees for AMR^2.

B. Job Scheduling

As noted in Section I, our problem is related to the knapsack problem [12]. To see this, note that if it is not feasible to schedule on the ES and all jobs have to be assigned to the ED, then maximizing the total accuracy is equivalent to

maximizing profit, and the constraint T is equivalent to the capacity of knapsack. In this case, our problem turns out to be a generalization of the CCKP [25]. Another special case of our problem, where the ED has only a single model, can be formulated as a GAP [13], [26], with two machines. In GAP, n jobs (or items) have to be assigned to r machines (or knapsacks). Each job-machine pair is characterized by two parameters: processing time and cost. The objective is to minimize the total cost subject to a time constraint T on the makespan. It is known that GAP is APX-hard [27].

In their seminal work [14], the authors proposed an algorithm for GAP that achieves minimum total cost and has makespan at most $2T$. Their method involves solving a sequence of LP feasibility problems, in order to tackle the processing times that are greater than T, and compute the minimum total cost using bisection search. Their algorithm can also be used for solving a related extension of GAP, where the cost of scheduling a job on a machine increases linearly with decrease in the processing time of the job. In comparison to this setting, the accuracies (equivalent to negative costs) are not linearly related to the processing times of the jobs and thus the proposed method in [14] is not directly applicable to the problem at hand. Our proposed algorithm AMR^2 is different from their method in that it does not require to solve LP feasibility problems and the use of bisection search. Further, we prove the performance bounds using a different analysis technique which is based on a counting argument for the LP-relaxation and solving a sub-problem of the ILP.

III. System Model

Consider an ED and an ES connected over a network and the ED enlists the help of the ES for computation offloading. At time zero, n inference jobs, each representing the processing requirement of a data sample on a pre-trained ML model, are available to a scheduler at the ED. Let j denote the job index and $J = \{1, 2, \ldots, n\}$ denote the set of job indices.

A. ML Models and Accuracy

The ED is equipped with m pre-trained ML models, or simply models. Note that these may correspond to m instantiations of the same DNN with different hyperparameter values resulting in different model sizes; see for example [8], [9]. Or, the models may correspond to different ML algorithms such as logistic regression, support vector machines, DNN, etc. Since the ES is a computationally powerful machine, we consider that it is equipped with a state-of-the-art model. Let i denote the index of the model, and $M = \{1, 2, \ldots, m, m+1\}$ denote the set of model indices, where models 1 to m are on the ED and model $m+1$ is the model on the ES. We note that our problem formulation and the solution are applicable to any family of ML models deployed on the ED and the ES.

Let $a_i \in [0, 1]$ denote the average test accuracy (Top-1 accuracy) of model i. We note that when a job is processed on a model i, the resulting Top-1 accuracy, which we refer to by *true accuracy* for the job, can be quite different from the average test accuracy a_i. However, since the true accuracies are not known apriori, for analytical tractability, we consider that the accuracy of a model i for any job is a_i. Later, in our experimental results (cf. Section VII), we do present the results with the true accuracies achieved under our algorithm. Without loss of generality, we assume that $a_1 \leq a_2 \leq \ldots \leq a_m$, and also assume that the model $m+1$ is a state-of-the-art model with a higher average test accuracy than the models on the ED, i.e., $a_m \leq a_{m+1}$. In the sequel, the term 'accuracy' refers to the average test accuracy, unless otherwise specified.

B. Processing and Communication Times

The processing time of job j on model $i \in M \backslash \{m+1\}$ is denoted by p_{ij}, and on model $m+1$ it is denoted by $p'_{(m+1)j}$. Later, in Section VI-A, we consider the special case of identical jobs, that is relevant in applications where the data samples are identical. In several applications, the data samples may need pre-processing before they are input to the ML model. For example, in computer vision tasks, images require pre-processing and the time required for pre-processing varies with the size of the image [28]. In our experiments with the images from the ImageNet dataset, the pre-processing stage only involves reshaping the images to input to the DNN models. Let τ_{ij} denote the pre-processing time of job j on model i. We consider the pre-processing times are part of the processing times defined above.

Let c_j denote the communication time for offloading job j. It is determined by the data size of the job, i.e., the size of the data sample in bits, and the data rate of the connection between the ED and the ES. Given $p'_{(m+1)j}$ and c_j, the *total time* to process job j on the ES, denoted by $p_{(m+1)j}$, is given by $p_{(m+1)j} = c_j + p'_{(m+1)j}$. We deliberately use similar notation for the processing times p_{ij} on the ED and the total times $p_{(m+1)j}$ on the ES because it simplifies the expressions in the sequel. We consider that the communication times c_j are fixed and are known apriori. This is possible in the scenarios where the ED and the ES are connected in a LAN or in a private network with fixed bandwidth. In our experiments, the ED and the ES are connected via our institute's LAN, and the communication times have negligible variance. We also consider that the processing times of the jobs are known apriori and that they can be estimated from the historical job executions.

C. Optimization Problem

Given the set of jobs J at time zero, the *makespan* is defined as the time when the processing of the last job in J is complete. The objective of the scheduler at the ED is to assign the set of jobs J to the set of models M such that the total accuracy, denote by A, is maximized and the makespan is within the time constraint T. Note that a schedule involves the partitioning of the set J between the ED and the ES, and the constraint on the makespan implies that the completion time of all the jobs should be with in T on both ED and ES. The above objective is relevant in applications where the ED periodically collects the data samples in a period T and aims to finish their processing within the next period. By choosing

a small period T, a real-time application can aim for fast ML inference at a reduced total accuracy.

Let x_{ij} denote a binary variable such that $x_{ij} = 1$, if the scheduler assigns job j to model i, and $x_{ij} = 0$, otherwise. Note that, if $x_{(m+1)j} = 1$, then job j is offloaded to the ES. Therefore, a schedule is determined by the matrix $\mathbf{x} = [x_{ij}]$. We impose the following constraints on $\mathbf{x}$:

$$\sum_{i=1}^{m}\sum_{j=1}^{n} p_{ij}x_{ij} \leq T \tag{1}$$

$$\sum_{j=1}^{n} p_{(m+1)j}x_{(m+1)j} \leq T \tag{2}$$

$$\sum_{i=1}^{m+1} x_{ij} = 1,\ \forall j \in J \tag{3}$$

$$x_{i,j} \in \{0,1\},\ \forall i \in M, \forall j \in J, \tag{4}$$

where constraints (1) and (2) ensure that the total processing times on the ED and the ES, respectively, are within T, and thus the makespan is within T. Constraints in (3) imply that each job is assigned to only one model and no job should be left unassigned, and (4) are integer constraints. We are interested in the following accuracy maximization problem $\mathcal{P}$:

$$\begin{aligned} \underset{\mathbf{x}}{\text{maximize}} \quad & A = \sum_{i=1}^{m+1}\sum_{j=1}^{n} a_i x_{ij} \\ \text{subject to} \quad & (1), (2), (3), \text{ and } (4). \end{aligned}$$

Note that $\mathcal{P}$ is an ILP. We will show later that a special case of $\mathcal{P}$ reduces to CCKP, which is NP-hard, and thus $\mathcal{P}$ is NP-hard. Let A^* denote the optimal total accuracy for $\mathcal{P}$.

IV. Accuracy Maximization using LP-Relaxation and Rounding (AMR^2)

In this section, we first present the two key components of AMR^2: 1) the LP-relaxation of $\mathcal{P}$, and 2) an ILP with two jobs, which we refer by sub-ILP. Later, we present the details of AMR^2.

A. LP-Relaxation and sub-ILP

Given $\mathcal{P}$, we proceed with solving the LP-relaxation of $\mathcal{P}$, where the integer constraints in (4) are replaced using the following non-negative constraints:

$$x_{ij} \geq 0, \forall i \in M \text{ and } \forall j \in J. \tag{5}$$

Note that, the constraints $x_{ij} \leq 1$ are not required as this is ensured by the constraints in (3). Let the matrix $\bar{\mathbf{x}} = [\bar{x}_{ij}]$ and A^*_{LP} denote the schedule and the total accuracy, respectively, output by the LP-relaxation. Note that the LP-relaxed solution provides an upper bound on the total accuracy achieved by an optimal schedule, and thus we have $A^*_{\text{LP}} \geq A^*$.

Note that the solution to the LP-relaxation may contain x_{ij} values that are fractional, and the rounding procedure is critical to proving the performance bounds. To design a rounding procedure, we first refer to a key result in [29], where the author studied the problem of assigning N jobs to K parallel machines with the objective of minimizing the makespan. For the LP-relaxation of this problem, the author presented the following counting argument: there exists an optimal basic solution in which there can be at most $K-1$ *fractional jobs*, i.e., the jobs that are divided between machines, and all the other jobs are fully assigned. Further, the simplex algorithm outputs such a basic optimal solution. In our problem, there are two parallel machines, the ED and the ES, but in contrast to [29], the ED has multiple models and the jobs assigned to the ED are processed in sequence. Taking this new aspect into account, we extend the counting argument for the problem at hand and show that solving the LP-relaxation of $\mathcal{P}$ results in at most two fractional jobs. This structural result is stated in the following lemma.

Lemma 1. *For the LP-relaxation of $\mathcal{P}$, there exists an optimal basic solution with at most two fractional jobs.*

Proof. Since LP-relaxation of $\mathcal{P}$ has $n+2$ constraints, apart from the non-negative constraints in (5), one can show using LP theory that there exists an optimal basic solution with $n+2$ basic variables that may take positive values and all the non-basic variables take value zero. Under such an optimal basic solution, for the n constraints in (3) to be satisfied, at least one positive basic variable should belong to each of those n constraints. The remaining 2 basic variables may belong to at most two equations. This implies that at least $n-2$ equations should have exactly one positive basic variable whose value should be one in order to satisfy the constraint. Therefore, there can be at most two equations with multiple basic variables whose values are in $(0,1)$, and the two jobs that correspond to these equations are the fractional jobs. □

Given the basic optimal solution to the LP-relaxation, the result in Lemma 1 reduces the rounding procedure to assigning at most two fractional jobs. Without loss of generality, we re-index the jobs and refer to the fractional jobs by job 1 and job 2. We define the set $I = J\backslash\{1,2\}$ and refer to the assignment of I under $\bar{\mathbf{x}}$ as the *integer solution of the LP-relaxation*. We formulate the following ILP, which we refer to by sub-ILP, for computing the assignments for jobs 1 and 2.

$$\begin{aligned} \underset{x_{i1},x_{i2}}{\text{maximize}} \quad & \sum_{i=1}^{m+1} a_i(x_{i1}+x_{i2}) \\ \text{subject to} \quad & \sum_{i=1}^{m} p_{i1}x_{i1} + p_{i2}x_{i2} \leq T \\ & p_{(m+1)1}\,x_{(m+1)1} + p_{(m+1)2}\,x_{(m+1)2} \leq T \\ & \sum_{i=1}^{m+1} x_{ij} = 1,\ j \in \{1,2\} \\ & x_{ij} \in \{0,1\},\ \forall i \in M, \forall j \in J. \end{aligned} \tag{6}$$

Later, in Section V, we will see that the sub-ILP in (6) is crucial to proving performance guarantees for AMR^2.

B. AMR^2 Description

The main steps of AMR^2 are summarized in Algorithm 1. We use $\mathbf{x}^\dagger$ and $A^\dagger$ to denote the schedule and the total accuracy, respectively, output by AMR^2. The schedule $\mathbf{x}^\dagger$ comprises the integer solution of the LP-relaxation and the assignment of the fractional jobs that are obtained for the cases of one fractional job and two fractions jobs in line 4 and line 7, respectively, in Algorithm 1.

Algorithm 1: AMR^2

1: **Input**: p_{ij}, for all $i \in M$ and $j \in J$.
2: Solve the LP-relaxation of $\mathcal{P}$.
3: **if** One fractional job **then**
4: Assign job 1 to model $\arg\max_{i \in M}\{a_i : p_{i1} \leq T\}$.
5: **end if**
6: **if** Two fractional jobs **then**
7: Solve the sub-ILP in (6) using Algorithm 2.
8: **end if**
9: **Output**: Assignment matrix $\mathbf{x}^\dagger$ and total accuracy $A^\dagger$

The LP-relaxation is solvable in polynomial time. To solve the sub-ILP we consider different cases based on the processing times of jobs 1 and 2, and greedily pack them to maximize accuracy subject to the constraint T. The steps for solving the sub-ILP are presented in Algorithm 2. There are two main cases. For the case where processing time of at least one of the jobs is at most T on the ES, we assign at least one job to the ES. This case is presented in line 2 of Algorithm 2 and is satisfied for problem instances where $p_{ij} \leq T$, for all $i \in M$ and $j \in J$. For the case where processing times of both the jobs are greater than T, we schedule both of them on the ED. This case is presented in line 12 of Algorithm 2. In line 13, we use enumeration for finding the models i' and i''.

Computational complexity: The computational complexity of solving an LP with l variables is $O(l^3)$ (cf. [30]). In the LP-relaxation, the number of variables are $n(m+1)$, and thus its runtime is $O(n^3(m+1)^3)$. Solving sub-ILP using Algorithm 2 requires m^2 iterations due to the step in line 14, where we enumerate m choices for each job. Therefore, the runtime of AMR^2 is $O(n^3(m+1)^3)$.

V. Analysis of AMR^2

In this section, we analyse AMR^2 and present a $2T$ bound for its makespan and its total accuracy is at most $2(a_{m+1}-a_1)$ lower than the optimal accuracy.

We designed Algorithm 2 for computing an optimal solution, with makespan at most T, for the sub-ILP by considering all possible cases. From its description one can easily verify that it is indeed optimal for sub-ILP. We state this in the following lemma without proof.

Lemma 2. *Algorithm 2 is an optimal algorithm for the sub-ILP.*

In the following theorem, we state the bound for the makespan under AMR^2.

Algorithm 2: Algorithm for solving sub-ILP

1: **Input**: p_{i1} and p_{i2}, for all $i \in M$
2: **if** $p_{(m+1)1} \leq$ T **or** $p_{(m+1)2} \leq T$ **then**
3: **if** $p_{(m+1)1} + p_{(m+1)2} \leq T$ **then**
4: Assign both jobs to model $m+1$.
5: **else**
6: **if** $\max\{a_i : p_{i1} \leq T\} \geq \max\{a_i : p_{i2} \leq T\}$ **then**
7: Assign job 1 to model $\arg\max_i\{a_i : p_{i1} \leq T\}$ and job 2 to the ES.
8: **else**
9: Assign job 2 to model $\arg\max_i\{a_i : p_{i2} \leq T\}$ and job 1 to the ES.
10: **end if**
11: **end if**
12: **else if** $p_{(m+1)1} > T$ **and** $p_{(m+1)2} > T$ **then**
13: Assign job 1 to i' and job 2 to i'', where models i' and i'' are on the ES such that, $p_{i'1} + p_{i''2} \leq T$ and $a_{i'} + a_{i''}$ is the maximum.
14: **end if**
15: **Output**: Assignment for jobs 1 and 2.

Theorem 1. *If $\mathcal{P}$ is feasible, then the makespan of the system under AMR^2 is at most 2T.*

Proof. Given that $\mathcal{P}$ is feasible, the LP-relaxation of $\mathcal{P}$ is also feasible. Since the makespan under the LP-relaxation solution is at most T, the makespan of its integer solution cannot exceed T. Again, the feasibility fo $\mathcal{P}$ implies that the sub-ILP is feasible as it is a sub problem of $\mathcal{P}$ involving only two jobs from J. Using this and Lemma 2 we infer that Algorithm 2 always finds an assignment for jobs 1 and 2 such that the makespan for these two jobs is at most T. Now, $\mathbf{x}^\dagger$ comprises the integer solution of the LP-relaxation, with makespan at most T, and the assignment of the fractional jobs in both the cases of one fractional job and two fractional jobs, which also have makespans of at most T. Therefore, AMR^2 has a makespan at most 2T. □

Theorem 2. *The total accuracy achieved by an optimal schedule is at most $2(a_{m+1} - a_1)$ higher than the total accuracy achieved by AMR^2, i.e., $A^* \leq A^\dagger + 2(a_{m+1} - a_1)$.*

Proof. We prove that $A^*_{\text{LP}} \leq A^\dagger + 2(a_{m+1} - a_1)$ in the worst-case, and the result follows from the relation $A^*_{\text{LP}} \geq A^*$. If there are no fractional jobs in the LP-relaxed solution, then the result is true since in this case $A^*_{\text{LP}} = A^\dagger$. If there is one fractional job, which we refer by job 1, then the difference between A^*_{LP} and $A^\dagger$ is caused by reassigning job 1 by AMR^2 (cf. line 4 Algorithm 1). Since job 1 can contribute at most a_{m+1} to A^*_{LP} and its reassignment by AMR^2 results in at least a contribution of a_1 to $A^\dagger$, we obtain $A^*_{\text{LP}} - A^\dagger \leq a_{m+1} - a_1$.

When there are two fractional jobs, for some models i, $\hat{i}$, k and $\hat{k}$, it should be true that $\bar{x}_{i1} + \bar{x}_{\hat{i}1} = 1$ and $\bar{x}_{k2} + \bar{x}_{\hat{k}2} = 1$, meaning that job 1 is divided between models i and $\hat{i}$ and job 2 is divided between models k and $\hat{k}$ under the LP-relaxed

solution $\bar{\mathbf{x}}$. Let $A^*_{\text{LP,I}}$ denote the total accuracy for the set $I = J\backslash\{1,2\}$ under $\bar{\mathbf{x}}$. We have,

$$A^*_{\text{LP}} = A^*_{\text{LP,I}} + a_i\bar{x}_{i1} + a_{\hat{i}}\bar{x}_{\hat{i}1} + a_k\bar{x}_{k2} + a_{\hat{k}}\bar{x}_{\hat{k}2} \quad (7)$$

$$\leq A^*_{\text{LP,I}} + 2a_{m+1}. \quad (8)$$

In the last step above, we used $a_{m+1} \geq a_i$, for all $i \in M$. We now consider different cases based on the job assignments output by the Algorithm 2.

Case 1: Both jobs are offloaded to model $m+1$. Since AMR^2 comprises the integer solution solution of the LP-relaxation, i.e., the schedule of I under $\bar{\mathbf{x}}$, and the assignment from the Algorithm 2, we have $A^\dagger = A^*_{\text{LP,I}} + 2a_{m+1}$. Using (8), we obtain $A^\dagger \geq A^*_{\text{LP}}$. Note that $A^\dagger$ could exceed A^*_{LP} because the makespan of $\mathbf{x}^\dagger$ can exceed T in contrast to $\bar{\mathbf{x}}$.

Case 2: One job is offloaded on model $m+1$ and one job is assigned to model $\hat{i} \in J\backslash\{m+1\}$ on the ED. This implies, the offloaded to the ES contributes a_{m+1} and the other job contributes at least a_1 to to the total accuracy $A^\dagger$. Therefore, we have,

$$A^\dagger \geq A^*_{\text{LP, I}} + a_{m+1} + a_1 \geq A^*_{\text{LP}} - (a_{m+1} - a_1).$$

In the second step above we used (8).

Case 3: Both jobs are assigned to the ED. This case occurs when $p_{(m+1)1} > T$ and $p_{(m+1)2} > T$, and we have

$$A^\dagger \geq A^*_{\text{LP,I}} + 2a_1. \quad (9)$$

From (9) and (8), we obtain $A^*_{\text{LP}} - A^\dagger \leq 2(a_{m+1} - a_1)$.

We obtain the result by taking the worst case bounds in all the three cases. □

Since $2(a_{m+1} - a_1) \leq 2$, the difference between A^* and $A^\dagger$ becomes negligible as the number of jobs n increases. It is worth noting that the worst-case bound $2(a_{m+1} - a_1)$ results from **Case 3** in the proof of Theorem 2. While this case, where the processing times of jobs 1 and 2 are greater than T, can happen when T is small, typical problem instances have processing times less than T on the ES. Note that a T value that is smaller than the processing time of a job either becomes infeasible or results in very low optimal total accuracy. We also assert this in our experimental results. In the following corollary we present the worst-case bound for a case that is true for typical problem instances.

Corollary 1. *If the processing times of all jobs on the ES are at most T, then $A^* \leq A^\dagger + a_{m+1} - a_1$.*

Proof. Since processing times of all jobs on the ES are at most T, **Case 3** in the proof of Theorem 2 cannot happen. In all other cases, the worst-case bound we have is $A^*_{\text{LP}} \leq A^\dagger + a_{m+1} - a_1$, and the result is true since $A^* \leq A^*_{\text{LP}}$. □

Remark: The schedule $\mathbf{x}^\dagger$ given by AMR^2 may result in a makespan greater T. As noted before, a special case of our problem is GAP for which the best known approximation algorithm, proposed in [14], has the makespan bound $2T$ and produces a schedule that may exceed T. In our experimental results, we show that the percentage of violation on an average is at most 40% for the considered image classification application.

VI. Identical Jobs

In this section, we consider the problem $\mathcal{P}_{\text{I}}$, a special case of $\mathcal{P}$ where the jobs are identical, i.e., $p_{ij} = p_i$, for all models $i \in M$. We present Accuracy Maximization using Dynamic Programming (AMDP) for $\mathcal{P}_{\text{I}}$. The formulation for $\mathcal{P}_{\text{I}}$ is given below.

$$\underset{\mathbf{x}}{\text{maximize}} \quad \sum_{i=1}^{m+1} a_i \sum_{j=1}^{n} x_{ij}$$

$$\text{subject to} \quad \sum_{i=1}^{m} p_i \sum_{j=1}^{n} x_{ij} \leq T \quad (10)$$

$$p_{m+1} \sum_{j\in 1}^{n} x_{(m+1)j} \leq T \quad (11)$$

$$\sum_{i=1}^{m+1} x_{ij} = 1, \quad \forall j \in J \quad (12)$$

$$x_{ij} \in \{0,1\}, \quad \forall i \in M, \forall j \in J. \quad (13)$$

Next, we exploit the structure of $\mathcal{P}_{\text{I}}$ and reduce it to solving a Cardinality Constrained Knapsack Problem (CCKP).

A. CCKP

Our first observation is that the number of jobs assigned to the ES under an optimal schedule is given by $n_c = \left\lfloor \frac{T}{p_{m+1}} \right\rfloor$. To see this, assigning number of jobs less than n_c can only reduce the accuracy as the ES provides highest accuracy, and no more than n_c can be assigned due to constraint (11). We present this observation in the following lemma.

Lemma 3. *Under an optimal schedule, the number of jobs assigned to the ES is given by $n_c = \left\lfloor \frac{T}{p_{m+1}} \right\rfloor$.*

We define $n_l = n - n_c$. Since the jobs are identical, without loss of generality, we assign the last n_c jobs to the ES. We are now only required to compute the optimal assignment for jobs $j \in \{1,\ldots,n_l\}$ to the models 1 to m on the edge device. Thus, given Lemma 3, solving $\mathcal{P}_{\text{I}}$ is reduced to solving the following problem $\mathcal{P}_{\text{I}}'$:

$$\underset{\mathbf{x}}{\text{maximize}} \quad \sum_{i=1}^{m} a_i \sum_{j=1}^{n_l} x_{ij}$$

$$\text{subject to} \quad \sum_{i=1}^{m} p_i \sum_{j=1}^{n_l} x_{ij} \leq T, \quad (14)$$

$$\sum_{i=1}^{m} x_{ij} = 1, \quad \forall j \in \{1,\ldots,n_l\} \quad (15)$$

$$x_{ij} \in \{0,1\}, \forall i \in M\backslash\{m+1\}, \forall j \in \{1,\ldots,n_l\}.$$

We do a variable change to formulate the CCKP. Let r denote an index taking values from $\{1,\ldots,mn_l\}$. We define new

variables z_r, accuracies $\bar{a}_r$, and processing times $\bar{p}_r$ as follows: for $i \in M\backslash\{m+1\}$ and $j \in \{1,\ldots,n_l\}$,

$$z_r = \{x_{ij} : r = j + (i-1)n_l\},$$
$$\bar{a}_r = a_i,\ (i-1)n_l \le r < in_l,$$
$$\bar{p}_r = p_i,\ (i-1)n_l \le r < in_l.$$

The CCKP using $\{z_r : 1 \le r \le mn_l\}$ as the decision variables is stated below.

$$\underset{\{z_r\}}{\text{maximize}} \quad \sum_{i=1}^{mn_l} \bar{a}_r z_r$$

$$\text{subject to} \quad \sum_{r=1}^{mn_l} \bar{p}_r z_r \le T, \tag{16}$$

$$\sum_{r=1}^{mn_l} z_r = n_l, \tag{17}$$

$$z_r \in \{0,1\},\ \forall r \in \{1,\ldots,mn_l\}. \tag{18}$$

Let $\{z_r^* : 1 \le r \le mn_l\}$ denote an optimal solution for CCKP. The CCKP can be interpreted as follows. We have mn_l items, where each item represents a model and there are n_l copies of the same model. Since the jobs are identical, the problem reduces to the number of times a model is selected, equivalent to the number of jobs assigned to it, such that all jobs are assigned. In the following lemma we state that solving CCKP results in an optimal solution for $\mathcal{P}_{\text{I}}{}'$.

Lemma 4. *The solution $x_{ij}^* = \{z_r^* : r = j + (i-1)n_l\}$ is an optimal solution for $\mathcal{P}_{\text{I}}{}'$.*

Proof. By construction, $\mathcal{P}_{\text{I}}{}'$ and CCKP have one-to-one mapping between the decision variables, have equivalent objective functions and constraints in (14) and (16). They only differ in the constraints (15) and (17). We note that (17) is equivalent to

$$\sum_{j=1}^{n_l}\sum_{i=1}^{m} x_{ij} = n_l. \tag{19}$$

Let $\mathcal{P}_{\text{I}}{}^{\ddagger}$ denote the problem $\mathcal{P}_{\text{I}}{}'$ with the constraint (15) replaced by (19). From the above observations, $\mathcal{P}_{\text{I}}{}^{\ddagger}$ is equivalent to CCKP, and thus it is sufficient to show that an optimal solution $\{x_{ij}^{\ddagger}\}$ for $\mathcal{P}_{\text{I}}{}^{\ddagger}$ is optimal for $\mathcal{P}_{\text{I}}{}'$. Since (19) is a relaxation of the constraint in (15), the optimal objective value of $\mathcal{P}_{\text{I}}{}^{\ddagger}$ should be at least the optimal objective value of $\mathcal{P}_{\text{I}}{}'$. On the other hand, given $\{x_{ij}^{\ddagger}\}$, consider the assignment where for each model i, we assign $\sum_{j=1}^{n_l} x_{ij}^{\ddagger}$ jobs to it. Given that the jobs are identical, and from (19), all the n_l jobs will be assigned exactly once to some model. Thus, this assignment is feasible for $\mathcal{P}_{\text{I}}{}'$ and objective value under this assignment will be equal to the optimal objective value of $\mathcal{P}_{\text{I}}{}^{\ddagger}$. Thus, $\{x_{ij}^{\ddagger}\}$ is also an optimal solution for $\mathcal{P}_{\text{I}}{}'$. □

In Algorithm 3 we present AMDP for solving $\mathcal{P}_{\text{I}}$. The optimality of AMDP is a direct consequence of Lemmas 3 and 4 and is stated in the following theorem.

Theorem 3. *AMDP is an optimal algorithm for $\mathcal{P}_{\text{I}}$.*

Algorithm 3: AMDP

1: $n_l = n - \left\lfloor \frac{T}{p_{m+1}} \right\rfloor$
2: Assign the jobs $j \in \{n_l+1,\ldots,n\}$ to the ES
3: Solve the CCKP for $\{z_r^*\}$ using the DP algorithm
4: Assignment for remaining jobs:
$x_{ij}^* = \{z_r^* : r = j+(i-1)n_l\}$ for all $i \in M\backslash\{m+1\}$, and $j \in \{1,\ldots,n_l\}$.

B. The DP Algorithm

The main step in AMDP is to solve the CCKP for which one can leverage existing branch-and-bound or Dynamic Programming (DP) algorithms [12]. We use the DP algorithm since it has pseudo-polynomial runtime for computing the optimal solution. The summarize the main steps of the algorithm. Let s, k, and τ denote positive integers. We define $y_s(\tau,k)$ as the maximum accuracy that can be achieved by selecting items from the set $\{1,\ldots,s\}$, where $s \le mn_l$, given a time constraint τ ($\le T$) and the number of items to be selected are k ($\le n_l$).

$$y_s(\tau,k) = \max\left\{\sum_{r=1}^{s} \bar{a}_r z_r \Big| \sum_{r=1}^{s} \bar{p}_r z_r \le \tau, \sum_{r=1}^{j} z_r = k, z_r \in \{0,1\}\right\} \tag{20}$$

The DP iterations are given below:

$$y_s(\tau,k) = \begin{cases} y_{s-1}(\tau,k) & \text{if } \bar{p}_s \ge \tau \\ \max\{y_s(\tau-\bar{p}_s,k-1)+\bar{a}_s, y_{s-1}(\tau,k)\} & \text{otherwise.} \end{cases}$$

We compute the solution for $y_s(T,n_l)$, where $s = mn_l$.

The computational complexity of the DP algorithm is $O(mnT)$ and AMDP has the same computational complexity.

Remark: We note that AMDP can be easily extended to a slightly more general setting where the processing times of the jobs only depend on the models, but they may have different communication times. This implies that $p_{ij} = p_i$ for all the models on the ED, and $p'_{(m+1)j} = p'_{(m+1)}$ for all j, but c_j values could be different. This setting is applicable in scenarios where the data samples are heterogeneous, but their processing times on the models do not vary much. In this case, we order the list of jobs in the increasing order of their communication times and offload the jobs from the start of the list to the ES until constraint T is met. Since the jobs have same processing times on the models of the ED, it is easy to argue that this assignment optimal for the ES. For the remaining jobs we solve the CCKP.

VII. Experimental Results

In this section, we first present the experimental setup. We then present the implementation details for estimating the processing and communication times. As explained in Section II, the aspect of multiple models on the ED has not been considered in computation offloading literature and there are no existing algorithms that are applicable for the problem at hand for a performance comparison. Therefore, we present

the performance comparison between AMR^2 and a baseline Greedy Round Robin Algorithm (Greedy-RRA). Given the list of jobs, Greedy-RRA offloads them from the start of the list to the ES until the constraint T is met. The remaining jobs are assigned in a round robin fashion to the models on the ED until the constraint T is met. Any further remaining jobs are assigned to model 1. Note that Greedy-RRA solution may violate the time constraint T and its runtime is $O(n)$.

A. Experimental Setup

Our experimental setup comprises a Raspberry Pi device (the ED) and a local server (the ES) that are connected and located in the same LAN. Raspberry Pi has 4 cores, 1.5 GHz CPU frequency, and 4 GB RAM, with the operating system Raspbian 10, while the server has 512 cores, 1.4 GHz CPU frequency, and 504 GB RAM, with the operating system Debian 11. All the functions on Raspberry Pi and on the server are implemented using Python 3. To offload images from Raspberry Pi to the ES we used HTTP protocol, and implemented HTTP Client and Server using Requests and Flask, respectively.

The data samples are images from the ImageNet dataset for which we use DNN models for inference. On Raspberry Pi we import, from the TensorFlow Lite library, two pre-trained MobileNets corresponding to two values 0.25 and 0.75 for the hyperparameter α, which is a width multiplier for the DNN [31]. Both the models are quantized and require input images of dimensions 128×128. On the ES, we import a pre-trained ResNet50 model [6] from the Tensorflow library. The ResNet50 model requires input images of dimensions 224×224. Images of different dimensions need to be reshaped to the respective dimensions on the ES and the ED. The average test accuracies for the three models are presented in Table I.

Model	Average test Accuracy
MobileNet $\alpha = 0.25$ (model 1)	0.395
MobileNet $\alpha = 0.75$ (model 2)	0.559
ResNet50 (model 3)	0.771

TABLE I: Test accuracies of the considered DNN models [4].

We implemented both AMR^2 and Greedy-RRA on Raspberry Pi in Python 3. AMR^2 takes up to 50 ms for computing a schedule for 40 jobs. The runtime of AMR^2 is dominated by the runtime for solving LP-relaxation. In future, we plan to reduce this runtime by implementing AMR^2 in C. We, however, implemented AMDP in C and observed that it has a runtime less than 1 ms on Raspberry Pi for computing an optimal schedule for 300 jobs. This shows the advantage of using AMDP over AMR^2 for the case of identical jobs.

Remark: The testbed that we implemented uses a single thread on the Raspberry Pi. To further improve the performance of this system one may use two different threads, one for handling the offloading of the tasks to different models and another for retrieving the responses and rendering the results. This implementation improves the quality of experience of the user as the jobs will be rendered as soon as they are processed.

B. Estimation of Processing and Communication Times

In our experiments, we consider images of dimensions 128×128, 512×512, and 1024×1024, for which we estimate the processing and communication times using the following procedure. On Raspberry Pi, we run 30 samples of same image dimensions and use the median processing times as our estimate. Note that median is an unbiased estimate, and unlike the mean, it is not affected by cold start. We note that the estimates for the processing times include the reshape times.

In order to estimate the total time on the ES, we use the HTTP client/server connection to send 30 images of same image dimensions from Raspberry Pi to the server. For each image we measure the time till the reception of an inference for the image from the ES, and finally use the median. At the server we also measure the reshape time and the processing time, and the estimate for the communication time is obtained by subtracting the reshape time and the processing time from the total time. Since Raspberry Pi and the dedicated local server are in the same LAN, the observed communication times are almost constant with negligible variance. This is also true for the observed processing times, and we will later verify this when implementing the schedules using these estimates.

The estimates for the processing times are presented in Table II. Observe that the processing times increase with the model size. On Raspberry Pi, the variance in the processing times on a model is small. In contrast, the total times on the ES vary with the dimensions of the image and are an order of magnitude higher than the processing times on Raspberry Pi. In Figure 2, we present the communication, reshape, and processing times on the ES. It is worth noting that, as the dimensions of the image increases, both communication time and the reshape times increase. Thus, it is more advantageous to offload images with smaller dimensions.

Model/Image Dimension	128×128	512×512	1024×1024
MobileNet $\alpha = 0.25$	0.01	0.011	0.011
MobileNet $\alpha = 0.75$	0.04	0.04	0.043
ResNet50	0.28	0.32	0.38

TABLE II: Estimated processing times (in seconds), for MobileNets on Raspberry Pi and ResNet50 on the server.

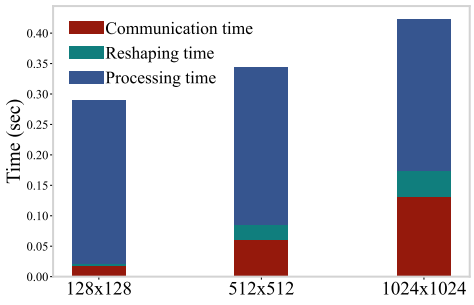


Fig. 2: Estimated total time for inference on the ES.

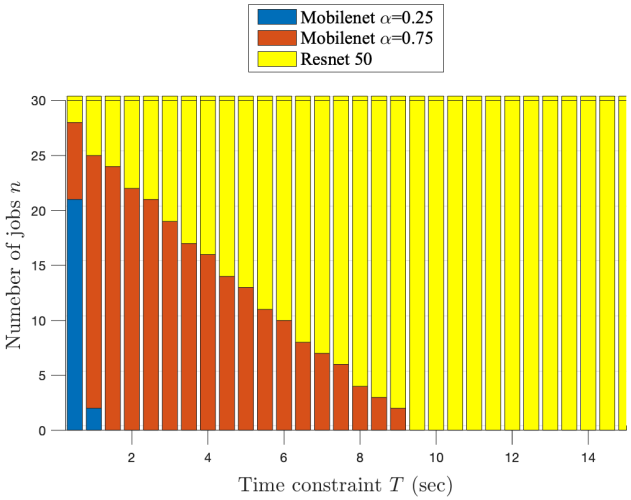


Fig. 3: Job assignment under AMR^2 for varying T.

C. Performance of AMR^2

In Figure 3, we examine the number of jobs assigned to different models under AMR^2. Observer that as T increases the number of jobs assigned to larger models increases. Also, note that MobileNet $\alpha = 0.25$ is only being used when T is small. In all the subsequent figures, for each point, we run 30 experiments and compute the average. Recall that the total accuracy $A^\dagger$ is based on the average test accuracy of the models. In addition to $A^\dagger$, we also present the total true accuracy for AMR^2 by summing the Top-1 accuracies we observe from executing the images under the given schedule $\mathbf{x}^\dagger$. In Figures 4 and 5, we compare total accuracy achieved under different schedules, by varying T and n, respectively. For $n = 60$, no LP-relaxed solution exists for $T = 0.5$ sec.

From both figures, we observe that $A^\dagger$ overlaps with, and in some cases exceeds, the total accuracy of the LP-relaxed solution A^*_{LP}. This is because all the processing times (cf. Table II) are less than 0.5 sec, the minimum value used for T, and therefore, from Corollary 1 the worst-case bound is at most $a_3 - a_1 = 0.376$ (cf. Table I). Furthermore, in some cases, where T is large enough, AMR^2 may assign both the fractional jobs to the server and $A^\dagger$ exceeds A^*_{LP}. In these cases, however, the makespan under $\mathbf{x}^\dagger$ exceeds T.

In Figure 4 we observe that the total true accuracy of AMR^2 is lower than $A^\dagger$, while in Figure 5 it exceeds when the number of jobs are small. We note that this behaviour is highly dependent on the set of images we chose and the DNN models. This is expected because the true accuracy for an image on a model can have a large deviation from the average test accuracy of that model. From Figure 4, we observe that AMR^2 always has higher total true accuracy than Greedy-RRA with a percentage gain between 20–60% with an average of 40%, with lower percentage gains at smaller T. The latter fact is also confirmed in Figure 5 when $T = 0.5$ sec. This is expected, because for $T = 0.5$ sec, not many jobs can be offloaded to the server as the processing times are around 0.3 seconds. For $T = 4$ we again see significant gains of around 40–50%.

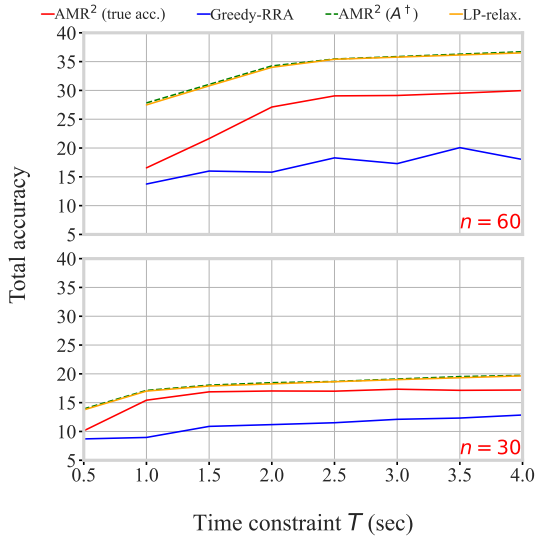


Fig. 4: Comparison of the total accuracy under different schedules for varying T and for $n = 30$ and $n = 60$.

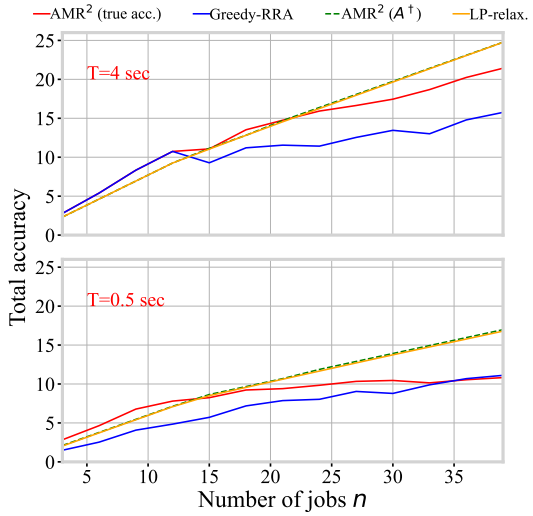


Fig. 5: Comparison of the total accuracy under different schedules for varying n and $T = 0.5$ sec and $T = 4$ sec.

In Figure 6, we present the makespan achieved by AMR^2 and Greedy-RRA for varying n. The real-time makespan, i.e., the time elapsed at Raspberry Pi from the start of scheduling the jobs till the finishing time of the last job, is indicated by AMR^2 in the legend. The estimated makespan that is numerically computed using the schedule $\mathbf{x}^\dagger$ and the estimated processing and communication times is indicated by AMR^2 (estd. proc. time). Observe that both these makespans have negligible difference asserting that the variances in our estimates for both communication and processing times are

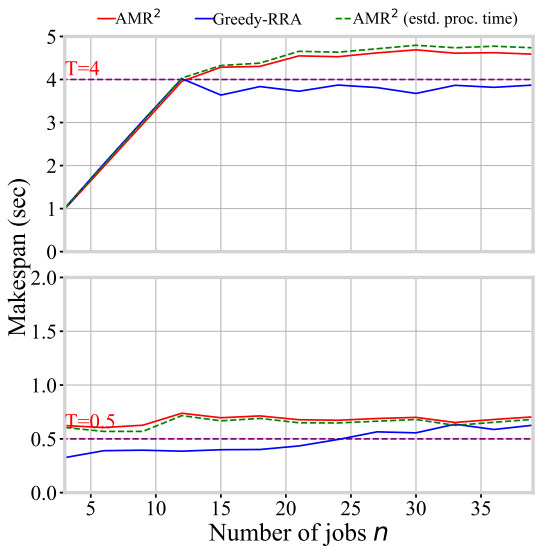


Fig. 6: Makespan under AMR^2 and Greedy-RRA for varying n, and $T = 0.5$ sec and $T = 4$ sec.

small. Also, observe that both AMR^2 and Greedy-RRA violate the time constraint for different problem instances. For $T = 4$, AMR^2 violates T for $n \geq 13$, but then it saturates at a makespan with maximum percentage of violation of 15%. This is expected, because from Lemma 1 there cannot be more than two fractional jobs irrespective of n value and thus, the constraint violation due to the reassignment of the fractional jobs do not increase beyond $n = 30$. This saturation effect can also be observed for $T = 0.5$. In this case, the percentage of violation under AMR^2 is higher, up to 40%, because the processing times on the server are comparable to $T = 0.5$ sec and reassigning a fractional job to the server results in higher percentage of violation.

VIII. Conclusion

We have studied the offloading decision for inference jobs between an ED and an ES, where the ED has m models and the ES has a state-of-the-art model. Given n data samples at the ED, we proposed an approximation algorithm AMR^2 for maximizing the total accuracy for the inference jobs subject to a time constraint T on the makespan. We proved that the makespan under AMR^2 is at most $2T$, and its total accuracy is lower than the optimal total accuracy by at most 2, and for typical problem instances it is lower by at most $a_{m+1} - a_1$. When the data samples are identical, we have proposed AMDP, a pseudo-polynomial time algorithm to compute the optimal schedule. We have implemented AMR^2 on Raspberry Pi and demonstrated its efficacy in improving the inference accuracy for classifying images within a time constraint T. Also, under the considered scenarios AMR^2 provides, on average, 40% higher total accuracy than that of Greedy-RRA.

In our problem model, we considered that the communication times are deterministic and in our testbed we used an architecture where the ED and the ES are connected via Ethernet. However, communication times will be random if one considers offloading over wireless links, which we leave for future work.

References

[1] W. Shi, J. Cao, Q. Zhang, Y. Li, and L. Xu, "Edge computing: Vision and challenges," *IEEE Internet of Things Journal*, vol. 3, no. 5, pp. 637–646, 2016.

[2] P. Mach and Z. Becvar, "Mobile edge computing: A survey on architecture and computation offloading," *IEEE Communications Surveys Tutorials*, vol. 19, no. 3, pp. 1628–1656, 2017.

[3] L. Deng, G. Li, S. Han, L. Shi, and Y. Xie, "Model compression and hardware acceleration for neural networks: A comprehensive survey," *Proceedings of the IEEE*, vol. 108, no. 4, pp. 485–532, 2020.

[4] "Image classification using TensorFlow Lite," https://www.tensorflow.org/lite/guide/hosted_models.

[5] "Pytorch mobile," https://pytorch.org/mobile/home/.

[6] K. He, X. Zhang, S. Ren, and J. Sun, "Deep residual learning for image recognition," in *Proc. IEEE CVPR*, 2016, pp. 770–778.

[7] J. Deng, W. Dong, R. Socher, L.-J. Li, K. Li, and L. Fei-Fei, "Imagenet: A large-scale hierarchical image database," in *Proc. IEEE CVPR*, 2009, pp. 248–255.

[8] M. Sandler, A. Howard, M. Zhu, A. Zhmoginov, and L.-C. Chen, "Mobilenetv2: Inverted residuals and linear bottlenecks," in *Proc. IEEE CVPR*, 2018, pp. 4510–4520.

[9] S. Teerapittayanon, B. McDanel, and H. Kung, "Branchynet: Fast inference via early exiting from deep neural networks," in *Proc. ICPR*, 2016, pp. 2464–2469.

[10] H. Cai, C. Gan, and S. Han, "Once for all: Train one network and specialize it for efficient deployment," *CoRR*, vol. abs/1908.09791, 2019.

[11] M. L. Pinedo, *Scheduling: Theory, Algorithms, and Systems*, 3rd ed. Springer Publishing Company, Incorporated, 2008.

[12] H. Kellerer, U. Pferschy, and D. Pisinger, *Knapsack Problems*. Springer, Berlin, Germany, 2004.

[13] G. Ross and R. Soland, "A branch and bound algorithm for the generalized assignment problem," *Mathematical Programming*, vol. 8, pp. 91–103, 1975.

[14] D. Shmoys and E. Tardos, "An approximation algorithm for the generalized assignment problem," *Mathematical Programming*, no. 62, pp. 461–474, 1993.

[15] M. Satyanarayanan, P. Bahl, R. Caceres, and N. Davies, "The case for vm-based cloudlets in mobile computing," *IEEE Pervasive Computing*, vol. 8, no. 4, pp. 14–23, 2009.

[16] J. Liu, Y. Mao, J. Zhang, and K. B. Letaief, "Delay-optimal computation task scheduling for mobile-edge computing systems," in *Proc. IEEE ISIT*, 2016, pp. 1451–1455.

[17] Y. Mao, J. Zhang, and K. B. Letaief, "Dynamic computation offloading for mobile-edge computing with energy harvesting devices," *IEEE Journal on Selected Areas in Communications*, vol. 34, no. 12, pp. 3590–3605, 2016.

[18] J. P. Champati and B. Liang, "Semi-online algorithms for computational task offloading with communication delay," *IEEE Transactions on Parallel and Distributed Systems*, vol. 28, no. 4, pp. 1189–1201, 2017.

[19] M.-H. Chen, B. Liang, and M. Dong, "A semidefinite relaxation approach to mobile cloud offloading with computing access point," in *Proc. IEEE SPAWC Worskhop*, 2015, pp. 186–190.

[20] M. Kamoun, W. Labidi, and M. Sarkiss, "Joint resource allocation and offloading strategies in cloud enabled cellular networks," in *Proc. IEEE ICC*, 2015, pp. 5529–5534.

[21] Y. Wang, M. Sheng, X. Wang, L. Wang, and J. Li, "Mobile-edge computing: Partial computation offloading using dynamic voltage scaling," *IEEE Transactions on Communications*, vol. 64, no. 10, pp. 4268–4282, 2016.

[22] Z. Wang, W. Bao, D. Yuan, L. Ge, N. H. Tran, and A. Y. Zomaya, "See: Scheduling early exit for mobile dnn inference during service outage," in *in Proc. MSWIM*, 2019, p. 279–288.

[23] S. S. Ogden and T. Guo, "Mdinference: Balancing inference accuracy and latency for mobile applications," in *Proc. IEEE IC2E*, 2020, pp. 28–39.

[24] I. Nikoloska and N. Zlatanov, "Data selection scheme for energy efficient supervised learning at iot nodes," *IEEE Communications Letters*, vol. 25, no. 3, pp. 859–863, 2021.

[25] K. Dudzinski, "On a cardinality constrained linear programming knapsack problem," *Operations Research Letters*, vol. 8, no. 4, pp. 215–218, 1989.

[26] D. G. Cattrysse and L. N. Van Wassenhove, "A survey of algorithms for the generalized assignment problem," *European Journal of Operational Research*, vol. 60, no. 3, pp. 260–272, 1992.

[27] C. Chekuri and S. Khanna, "A ptas for the multiple knapsack problem," in *Proceedings of the Eleventh Annual ACM-SIAM Symposium on Discrete Algorithms*, ser. SODA '00. USA: Society for Industrial and Applied Mathematics, 2000, p. 213–222.

[28] P. Ross and A. Luckow, "Edgeinsight: Characterizing and modeling the performance of machine learning inference on the edge and cloud," in *2019 IEEE International Conference on Big Data (Big Data)*, 2019, pp. 1897–1906.

[29] C. Potts, "Analysis of a linear programming heuristic for scheduling unrelated parallel machines," *Discrete Applied Mathematics*, vol. 10, no. 2, pp. 155–164, 1985.

[30] J. van den Brand, "A deterministic linear program solver in current matrix multiplication time," in *Proc. ACM SODA*. Society for Industrial and Applied Mathematics, 2020, p. 259–278.

[31] A. G. Howard, M. Zhu, B. Chen, D. Kalenichenko, W. Wang, T. Weyand, M. Andreetto, and H. Adam, "Mobilenets: Efficient convolutional neural networks for mobile vision applications," *CoRR*, vol. abs/1704.04861, 2017.