Scheduling with Fully Compressible Tasks: Application to Deep Learning Inference with Neural Network Compression

Tiago da Silva Barros, Frédéric Giroire, Ramon Aparicio-Pardo, Stephane Perennes, Emanuele Natale

Scheduling with Fully Compressible Tasks: Application to Deep Learning Inference with Neural Network Compression

Tiago da Silva Barros, Frédéric Giroire, Ramon Aparicio-Pardo, Stéphane Pérennes, and Emanuele Natale
Université Côte d'Azur/CNRS/I3S/Inria, Sophia Antipolis, France
tiago.da-silva-barros@univ-cotedazur.fr

***Abstract*—With the advent and the growing usage of Machine Learning as a Service (MLaaS), cloud and network systems are now offering the possibility to deploy ML tasks on heterogeneous clusters. Then, network and cloud operators have to schedule these tasks, determining both when and on which devices to execute them. In parallel, several solutions, such as neural network compression, were proposed to build small models which can run on limited hardware. These solutions allow choosing the model size at inference time for *any* targeted processing time without having to re-train the network.**

In this work, we consider the Deadline Scheduling with Compressible Tasks (DSCT) problem: a *novel scheduling problem* with task deadlines where the *tasks can be compressed*. Each task can be executed with a certain compression, presenting a trade-off between its compression level (and, its processing time) and its obtained utility. The objective is to maximize the tasks utilities. We propose an *approximation algorithm with proved guarantees* to solve the problem. We validate its efficiency with extensive simulation, obtaining near optimal results. As application scenario, we study the problem when the tasks are Deep Learning classification jobs, and the objective is to maximize their global accuracy, but we believe that this new framework and solutions apply to a wide range of application cases.

***Index Terms*—scheduling, compressible tasks, neural network compression, approximation algorithms, convex programming.**

I. Introduction

Nowadays, with the deployment of new networks, such as 5G and IoT (Internet of Things), the traditional paradigms for placing and deploying many cloud and network services have been changed. The new hardware capabilities of devices and the network requirements, such as latency and delay, create a new scenario in which the services and functions, in particular Machine Learning as a Service (MLaaS), are deployed along the cloud-to-edge continuum [1]. This topic has attracted a lot of attention and can be applied in many applications, such as self-driving cars, unmanned aerial vehicles (UAVs) [2] and Mobile Edge Computing (MEC) [3]. In this domain, the researchers study the deployment of Deep Learning (DL) Models on devices in the network edge, such as gateways, microcontrollers, antennas, and mobile phones [4]. However, along this continuum, the edge devices present strong constraints in terms of disk space, computational power, and memory, limiting the latency and accuracy of the models [5].

Thus, it is necessary to find ways for running complex neural networks in devices with constrained hardware [6]. A first solution is to design light neural networks to be used on specific devices, e.g. MobileNet [7] for cellphones. Another technique, suitable for a a large set of devices, is to use model compression techniques. The principle is to reduce the size of an efficient neural networks in order to reduce its memory and storage usage while having a minimal accuracy drop [8]. Recent works, such as Yu and Huang [9] and Cai et al. [10], propose to train an adaptable Deep Neural Network (DNN), which can be configured during the inference without requiring to re-train the network. For example, the Once-For-All (OFA) solution [10] allows to adjust the width and kernel size of each layer, the model depth, and the input image size to build faster models at inference time.

In this work, we study how to *use such adjustable neural networks* to deploy a set of DL models in a networked system. The inference requests are considered as tasks, which have to be scheduled along the cloud-edge continuum before their deadlines. Each task usually has a fixed duration and the scheduling problem can be solved by traditional techniques, such as Earliest Deadline First (EDF) and Shortest Job First (SJF) [11]. The requests can be executed on very heterogeneous devices using compressed networks. Recent works [12]–[14] have proposed to use a collection of pretrained models with different sizes (i.e., different number of parameters) and to select the best one to optimize latency and accuracy. Here, we propose to go a step further by allowing a machine to use *any* level of compression on the model size. Indeed, given any processing time target, adjustable networks, thanks to their very large number of potential configurations (e.g. for MobileNet, more than 10^{19} possible architectures), allow to produce a model with a size (and then, a processing time) very close to the targeted time. This leads to the following optimization problem.

In this paper, we study the new problem of scheduling a set of *fully compressible tasks* with deadlines, in which *each task can be run by a compressed algorithm with an arbitrary number of parameters*, presenting a trade-off between its processing time and the obtained utility. The goal is to decide

This work has been supported by the UCA JEDI (ANR-15-IDEX-01), EUR DS4H (ANR-17-EURE-004), and ARTIC (ANR-19-CE-25-0001-01) projects, by the France 2030 program under grant agreements No. ANR-22-PECL-0003 and ANR-22-PEFT-0002, by SmartNet, and by the European Network of Excellence dAIEDGE under Grant Agreement Nr. 101120726.

when, on which machine, and with which compression level, each task should be executed in order to maximize the global utility. We propose *convex optimization models* and *an approximation algorithm* to solve the problem for a large family of utility functions. We used as example classification inference tasks solved using the OFA technique, but our framework is general, and can be applied to any task following the classic law of diminishing marginal utility [15]. The contributions can be summarized as follows:

- We study the scheduling problem in which tasks can be compressed. The tasks have a utility function expressing their trade-off between processing time and utility. We focus on classification tasks solved with compressible neural networks. In this case, the utility is given by the attained accuracy.
- We carry out experiments to study this trade-off for the OFA [10] solution, particularly for low processing times.
- We model the accuracy function of large families of compressible tasks, as concave differentiable functions, and we propose convex optimization models to solve the problem exactly.
- We discuss the problem complexity and propose (i) an exact algorithm to solve the problem on one machine, and (ii) an approximation algorithm with proved guarantees for several heterogeneous machines.
- Finally, the solutions are validated and compared to state-of-the-art solutions with extensive experimentation.

The rest of this paper is organized as follows. In Section II, we review related works in more details. In Section III, we describe the experiments done to generalize the OFA solution and to get its accuracy function. In Section IV, we present the problem formulation. In Sections V and VI, we present details and analysis of our scheduling algorithms. In Section VII, we evaluate our proposed algorithms. In Section VIII, we discuss our solution and conclusions are drawn, together with open questions for future work.

II. Related Work

NN Compression. Model compression plays an important role in Edge Computing, since it allows the deployment of light models, leading to computation, memory, and communication bandwidth savings. A large review [8] investigated model compression techniques (e.g. pruning and knowledge distillation) for Deep Learning (DL). These methods are usually applied during the training phase and use a significant time to update the weights.

To address this problem, some approaches [9], [10] were recently proposed for compressing the Neural Network (NN) during the inference. Cai et al. [10] propose a specific training technique referred to as Once-For-All (OFA). That technique consists of a progressive shrinking and an evolutionary search during the inference phase to select the best configuration for a given targeted processing time from a large number of parameters (depth, width, kernel size, and resolution).

Scheduling. A large number of works were proposed for addressing the scheduling problem for classic non compressible tasks with hard deadlines, a seminal example being the work of Leyland and Liu [16] in which they proposed the Earliest Deadlines First (EDF) algorithm. A wide review in this topic was developed by Chen et. al. [17].

Some works addressed the problem of scheduling allowing to compress the task processing time. Vickson [18] was the first to propose controlling the processing times in a single machine without deadlines. Alidaee and Ahmadian [19] then considered a single deadline with the objective of minimizing tardiness. Wu et al [20] and Shabtay and Kaspi [21] studied the multiple machines scenario. However, they do not consider deadlines.

In the cloud-to-edge continuum, a large number of works have addressed the scheduling problems such as optimizing completion time, latency and energy [22]–[26]. In this context, one of the first scenarios considering *compressible* tasks with several possible processing times is the transcoding of video streams [27] . In such a case, video chunks can be transcoded into different video quality levels, leading to a trade-off between the utility (the Quality of Experience , QoE) and the task processing time (the transcoding time).

Machine Learning inference Scheduling and Allocation Recently, a collection of works, e.g. [28]–[31] , studied how to efficiently allocate inference requests for Machine Learning (ML) models among network nodes with available computational resources.

The closest work to ours are [12]–[14]. They investigate the scheduling of DL inference tasks, while proposing creating a collection of DDN models using compression. The authors propose methods to select the model configuration, and the hardware setting, to maximize the QoE (Quality of Experience) [12], and a utility function based on latency and accuracy [13]. Both works, however, have a different objective function and do not consider deadlines requirements as hard constraints for scheduling. Nigade et al. [14] study how to maximize the accuracy of a set of DL model inference requests, while satisfying network and processing latency constraints.

However, all the above mentioned works use a limited number of models, constrained by the storage constraints of devices. Such a system does not scale up if we want to arbitrarily increase the number of possible configurations to adapt to any real system. On the contrary, we consider fully compressible tasks using adjustable neural networks to any arbitrary size, without requiring to store a high number of models. Studying it allows us to be the first, to the best of our knowledge, to propose a *solution with proved guarantees* to choose any arbitrary compression level and the machine on which to execute each task, while maximizing the global accuracy and satisfying deadline constraints for a large family of accuracy functions.

III. Experiments and Latency-accuracy tradeoff analysis

In this section, we derive the latency-accuracy trade off of families of compressible neural networks such as [9], [10]

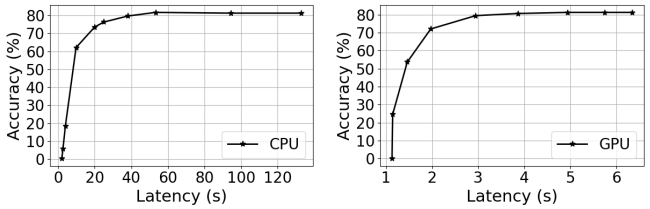


Fig. 1. Accuracy vs. Latency using the OFA framework for classifying a set of 1000 images from `imagenet-1k`. Experiments on a CPU (left) and GPU (right).

experimentally. Indeed, in their works, the authors study a limited range of reached accuracy, mostly between 70% and 80%, as their goal is to obtain efficient models on devices with limited hardware. For our problem, we need to have the latency-accuracy trade-off for the full accuracy range for processing times from 0 to the maximum one. We thus made new experiments. We selected Once-For-All (OFA) [10], as it obtains better results than other similar approaches.

As discussed in Section II, OFA approach reduces the model by changing the value of 4 different parameters: depth, width, kernel size, and resolution. We extended the possible configurations by adding new values (in bold) for *resolution* ([**16, 32, 48, 64, 96**, 128, 144, 160, 176, 192, 224]) and *depth* ([**1**, 2, 3, 4]), as they are the parameters with the largest impact on the accuracy. We tested the accuracy of the configurations with the classical `imagenet-1k` dataset [32], performing image classification over around 1000 classes.

We used the pre-trained `resnet` model [33] made available by the authors. Then, we performed fine-tuning to adapt the weights over the new configuration range over 20 epochs. OFA also uses an evolutionary search during the inference to choose the best configuration for a maximal input latency. This approach is based on a lookup table to predict the latency, and on a neural network to predict the accuracy of a configuration.

We then evaluated the updated `OFA-resnet` in two different devices, one equipped with a CPU (Intel Core i9-12900H), and the other one with a GPU (NVIDIA RTX A2000). Results are shown in Fig. 1 (left) and (right), respectively. The latency and accuracy of the configurations selected by the evolutionary search algorithm are plotted. Each point represents the average accuracy reached when classifying a set with 1000 images for a given maximum latency. First, we observe that we obtained a full range for the trade-off. Then, for both CPU and GPU cases, we see that the accuracy function is a non decreasing function with a very strong accuracy gains for small latency times and an almost flat curve for large latency times.

IV. Modeling

A. Problem formulation

We define the Deadline Scheduling with Compressible Tasks (DSCT) problem, which consists of scheduling a set of n tasks which can be fully compressed on m machines. Formally, we have a set J of tasks. Each task $j \in J$ has a deadline d_j, a maximum processing time $t_j^{\max}$, and a utility function $u_j(t)$ giving its utility when executed for a time t. The goal is to choose the processing time $t_j \in [0, t_j^{\max}]$ for each task $j \in J$ in order to maximize the global utility, i.e., $\sum_{j=0}^{n} u_j(t_j)$, the sum of the utility of all executed tasks.

In this paper, we focus on the case where tasks are *classification inference tasks* done with compressible deep neural networks. Nevertheless, the same framework also applies to other tasks as *live video transcoding* [27] , where we can *compress* the transcoding time generating video streams with lower video quality levels.

The utility function for a task $j \in J$ corresponds to the accuracy $a_j(t)$ reached by the neural network (or any computing process) for a given processing time. In the following, we use a standard order of the tasks by non-decreasing deadlines. The tasks are often denoted by their index in that order. This way the set of tasks is noted $J = \{1, 2, \ldots, n\}$ and $d_i \leq d_j$ iff $i \leq j$.

B. Accuracy Function

The accuracy of a compressible neural networks (i.e., OFA [10]) can be modeled as a function, denoted as $a(t)$, of the processing time t with the following properties:

- Concave and non decreasing. It is based on the fact that the marginal gain of running an inference model is decreasing with the task processing time, and tending to 0 when the processing time is large. We argue that such dependency can be generally found in many tasks utilities (accuracy functions), not only in compressible neural networks. For example, in Approximation Theory polynomials of degree s can be approximated with Chebyshev's polynomials of degree d up to an error which scales as $e^{-s^2/d}$ [34, Theorem 3.3], while the classical analysis of the famous AdaBoost algorithm shows that, for a weakly PAC learnable class of functions $\mathcal{C}$, T weak learners can be combined to obtain a classifier with empirical error of order $e^{-\Theta(T)}$ [35].
- Support $[0, t^{\max}]$. Processing time 0 means accuracy $a^{\min}$. Note that $a^{\min}$ for a classification class with C classes corresponds to a random choice between the classes leading to an accuracy $a^{\min} = 1/C$. processing time $t^{\max}$ has an accuracy $a^{\max} \leq 1$. Note that our model also allows setting $t^{\max} = \infty$. In this case, $a(t)$ converges to the constant $a^{\max}$, when t goes to infinity.

We add a technical property:

- Differentiability (and thus continuity) over $\mathbb{R}^+$. Differentiability allows to define the *marginal gain* $a'_j(t)$ of a task $j \in J$ when executed for a time t.

In the following, we study this *general class* of accuracy functions. We also use the *exponential accuracy function* as an illustration, as it models well the behavior of OFA, studied in the previous section (See also the discussion in Section VIII). It is defined as follows:

$$a(t) = (a^{\max} - a^{\min})(1 - \exp(-\theta t)) + a^{\min},$$

where θ is the slope at the origin of the accuracy function and is a measure of the job efficiency.

V. Problem on a single machine

A. Formulation as a Convex Program

The DSCT on a single machine can be formulated as a convex program (because of the concavity of the accuracy function) with only fractional variables. Indeed, the continuity over $\mathbb{R}^+$ of the accuracy makes that no binary variables are needed to indicate if a task is executed. Not executing a task is equivalent to setting its processing time to 0.

Objective functions. The general objective function is $\max \sum_{j=1}^{n} a_j(t)$ This is equivalent to minimize the error defined as $1 - a(t)$: $\min \sum_{j=1}^{n} 1 - a_j(t)$. As the accuracy $a_j(t)$ is concave, the error $1 - a_j(t)$ is convex. The sum of the error thus is convex as a sum of convex functions.

Convex optimization program with linear constraints. The variables of the program are the processing times t_j for all $j \in J$.

$$\min \quad \sum_{j=1}^{n} 1 - a_j(t_j) \tag{1}$$

$$\text{s.t.} \quad \sum_{i=1}^{j} t_i \leq d_j, \qquad j \in J = \{1, 2, 3...n\} \tag{1a}$$

$$t_j \leq t_j^{\max}, \qquad j \in J \tag{1b}$$

$$t_j \in \mathbb{R}^+, \qquad j \in J \tag{1c}$$

Equation (1a) ensures that, for each deadline, all tasks with inferior deadlines must be executed before it. Equations (1b) and (1c) state that t_j is chosen in $[0, t_j^{\max}]$.

Algorithmic complexity and Resolution methods. The DSCT thus is polynomial for a concave accuracy function. For a *differentiable* accuracy function, it can be solved using *Kelley's Cutting-Plane Method* with a linear solver or *gradient descent*. *We study faster methods, with closed form formulas for the exponential accuracy function, in the following.*

B. Optimal Algorithm

We first discuss the KKT (Karush-Kuhn-Tucker) conditions for the optimization problem and, then, we present an algorithm to solve it with polynomial time complexity.

1) Optimality Conditions: In our convex problem, the KKT conditions are necessary and sufficient conditions for optimality [36] (Chap. 5 p. 226 and 244). To analyze them, we first write the Lagrangian of the problem:

$$L(t, \mu) = \sum_{j=1}^{n} (1 - a_j(t_j)) + \boldsymbol{\mu}^T \mathbf{g(t)} \tag{2}$$

where $\boldsymbol{\mu} = [\mu_1, ..., \mu_{3n}]$ and $\boldsymbol{g(t)} = [t_1 - d_1, \ldots, \sum_{i=1}^{n} t_i - d_n, t_1 - t_1^{\max}, \ldots, t_n - t_n^{\max}, -t_1, \ldots, -t_n]$.

The *complementary slackness conditions* give that, when a task $j \in J$ is neither constrained by its deadline, nor by its maximum execution time, and has a strictly positive execution time (we say that such tasks are *3-unconstrained*), we have $\mu_j = \mu_{n+j} = \mu_{2n+j} = 0$. Furthermore, if $j + 1$ is neither constrained by its maximum time nor by its minimum times (i.e., $\mu_{n+j+1} = \mu_{2n+j+1} = 0$), the optimum execution times of both tasks are derived from the *zero-gradient condition* (i.e., the gradient at the optimum $(\boldsymbol{t^*}, \boldsymbol{\mu^*})$ should be $\mathbf{0}$) and we have:

$$\frac{\partial}{\partial t_j} a(t_j) = \frac{\partial}{\partial t_{j+1}} a(t_{j+1}). \tag{3}$$

The intuition is that, when tasks are *3-unconstrained*, the *optimum schedule* is to choose their execution time so that they have the *same marginal gain* i.e., the same value for the partial derivative of their accuracy functions. This is due to the concavity of the accuracy function or equivalently to its decreasing marginal gain. In the case of the exponential accuracy function, this gives:

$$\Delta a_j \theta_j e^{-\theta_j t_j} = \Delta a_{j+1} \theta_{j+1} e^{-\theta_{j+1} t_{j+1}}, \tag{4}$$

where $\Delta a_j \stackrel{\text{def}}{=} a_j^{\max} - a_j^{\min}$. The optimum execution time of task j is thus given by:

$$t_j = \frac{\theta_{j+1}}{\theta_j} t_{j+1} - \frac{1}{\theta_j} \ln \left(\frac{\Delta a_{j+1} \theta_{j+1}}{\Delta a_j \theta_j} \right), \tag{5}$$

if the value is non negative, otherwise $t_j = 0$ (in this case, task j is constrained by the non negativity bound and $\mu_{2n+j} > 0$).

Note that if both tasks have the same efficiency $\theta_j = \theta_{j+1}$ and same efficiency range $\Delta a_j = \Delta a_{j+1}$, then the condition simplifies to: $t_j = t_{j+1}$. Thus, the *optimal execution time* for two consecutive tasks j and $j + 1$, when j is *not constrained by its deadline*, is taking *equal execution time* $t_j = t_{j+1}$.

On the other hand, the non negativity constraints of the KKT multipliers ($\mu_j \geq 0$) give that, from the zero-gradient condition, $\forall j \in \{1, \ldots, n-1\}$,

$$\frac{\partial}{\partial t_j} a(t_j) \geq \frac{\partial}{\partial t_{j+1}} a(t_{j+1}) + \mu_{n+j+1} - \mu_{n+j} - \mu_{2n+j+1} + \mu_{2n+j}. \tag{6}$$

If both j and $j+1$ are not constrained by their maximum and minimum execution times, we have

$$\frac{\partial}{\partial t_j} a(t_j) \geq \frac{\partial}{\partial t_{j+1}} a(t_{j+1}). \tag{7}$$

The intuition is that, when considering two tasks i and j with $i < j$, j is less constrained by its deadline than i. Thus, its marginal gain may always be lower or equal than the one of i (except if constrained by its maximum execution time). Otherwise, its execution time could have been increased, leading to a better solution. For the exponential accuracy function, it gives:

$$-\Delta a_i \theta_i e^{-\theta_i t_i} \geq -\Delta a_j \theta_j e^{-\theta_j t_j} \text{ and } t_i \leq \frac{\theta_j}{\theta_i} t_j - \frac{1}{\theta_i} \ln \left(\frac{\Delta a_j \theta_j}{\Delta a_i \theta_i} \right). \tag{8}$$

Note, that for two tasks i and j with same efficiency $\theta_i = \theta_j$ and same accuracy range $\Delta a_i = \Delta a_j$, the equation simplifies to: $t_i \leq t_j$.

2) Algorithm proposal: Based on the KKT analysis, we propose Algorithm 1 which aims to allocate the processing time t_j for each task $j \in J$. The algorithm takes as input the list of task deadlines and maximum execution times, and returns the execution time for each task. It first sorts the tasks

Algorithm 1 Exact algorithm for scheduling in one machine

Input: List of task deadlines $[d_1, ..., d_n]$ and $[t_1^{\max}, ..., t_n^{\max}]$.
Output: List of task execution times $[t_1, ..., t_n]$

1: Sort the tasks by non decreasing deadlines.
2: $t_1 \leftarrow \min(d_1, t_1^{\max})$
3: $T_{\text{current}} = t_1$ ▷ $T_{\text{current}} \stackrel{\text{def}}{=} \sum_{i=1}^{j} t_i$ for current task j
4: **for** $j : 2 \leq j \leq n$ **do**
5: $t_j \leftarrow \min(d_j - T_{\text{current}}, t_j^{\max})$; $T_{\text{current}} \leftarrow T_{\text{current}} + t_j$
6: **if** $t_j = t_j^{\max}$ **then**
7: Continue to next iteration of the `for` loop
8: $a' = a'_j(t_j)$ ▷ current gain
9: $K \leftarrow \{j\}$ ▷ Set of 3-unconstrained tasks with $a'(i) \leq a'$. K sorted by non increasing $a'(t_k(t_k))$
10: $Z \leftarrow \emptyset$ ▷ Set of tasks with $t_i = 0$
11: $T \leftarrow t_j$; $i \leftarrow j - 1$
12: **while** ($a'_i(t_i) < a'$ or $t_i = t_i^{\max}$) and $i \geq 0$ **do**
13: **if** ($t_i = 0$) or ($t_i = t_i^{\max}$ and $a'_i(t_i) \geq a'$) **then**
14: Continue to next while iteration
15: $K \leftarrow K \cup \{i\}$; $T \leftarrow T + t_i$
16: $t_i \leftarrow \max(\tau(T, K, i), 0)$
17: **if** $t_i = 0$ **then**
18: $K \leftarrow K \setminus \{i\}$; $Z \leftarrow Z \cup \{i\}$ ▷ Add in Z sorted by non increasing $a'(t_k(0))$
19: move $\leftarrow$ True
20: **while** move **do**
21: move $\leftarrow$ False
22: **for** $k \in Z$ **do** ▷ Z sorted
23: **if** $\tau(T, K, k) \geq 0$ **then**
24: $Z \leftarrow Z \setminus \{k\}$; $K \leftarrow K \cup \{k\}$
25: $move \leftarrow True$
26: **else** Break
27: **for** $k \in K$ **do** ▷ K sorted
28: $t_k \leftarrow \min(\tau(T, K, k), t_k^{\max})$
29: **if** $t_k = t_k^{\max}$ **then**
30: $K \leftarrow K \setminus \{k\}$; $T \leftarrow T - t_k^{\max}$
31: $move \leftarrow True$
32: **else** Break
33: **for** $k \in K$ **do**
34: $t_k \leftarrow \tau(T, K, k)$
35: $a' \leftarrow a_k(\tau(T, K, k))$
36: **return** $[t_1, ..., t_n]$

by order of non decreasing deadlines. It incrementally sets the execution time of task j to $\min(d_j - T_{\text{current}}, t_j^{\max})$, where T_{current} is the current load in the machine. If $t_j \leq t_j^{\max}$, it then tests if the marginal gain of task j is lower than the ones of tasks i for $i \leq j$, which is a optimality condition (Equation 7). If not, it updates their execution times in order to have equal marginal gains.

To compute the optimum execution times, we use the fact that two *3-unconstrained* tasks i and j should have equal marginal gains (Equation 3). Consider now we want to compute the optimum execution times of a set K of *3-unconstrained* tasks executed in an interval of time T (i.e., $\sum_{i \in K} t_i = T$). In the exponential case, we can derive from Equation (5) a formula for the execution time $\tau(T, K, j)$ of any task $j \in K$:

$$\tau(T, K, j \in K) = \frac{1}{1 + \sum_{i \in K \setminus \{j\}} \frac{\theta_j}{\theta_i}} \left(T + \sum_{i \in K \setminus \{j\}} \frac{1}{\theta_i} \ln \frac{\Delta a_j \theta_j}{\Delta a_i \theta_i} \right). \tag{9}$$

The subtlety in the algorithm is that we apply the formula in a context where constrained tasks (by their minimum, maximum execution time, or deadline) may become unconstrained in the process and vice versa. In this case, τ could be negative for a task i in K or be greater than $t_i^{\max}$. Then, we set $t_i = 0$ or $t_i = t_i^{\max}$, i is deleted from K, and the execution times are recomputed. Note that, for the exponential case, it takes a time linear in the number of tasks in K to compute a value of τ using Equation 9 for the first computation. However, computing the value of another element of K can then be done in constant time by updating both sums of the formula. Similarly, when K is updated by addition or removal of an element, updating the value of τ for an element in K can also be done in constant time. Hence, the computations of τ in the algorithm are done in constant amortized times. For a general accuracy function, we note $c(n)$ the amortized complexity of computing a value of τ.

Theorem 1. *Algorithm 1 returns an optimal schedule in time $O(n^3(\log n + c(n)))$, where $c(n)$ is the amortized complexity of computing a value of $\tau(T, K, j \in K)$ for a set of 3-unconstrained tasks K. In the case of the exponential accuracy function, $c(n) = O(1)$, and the complexity of the algorithm is $O(n^3 \log n)$.*

Proof. The proof is omitted due to space constraints. The correctness is ensured by showing by induction that at the end of step j of the `for` loop of line 4, the KKT conditions are satisfied for $i \in \{1, \ldots, j\}$. The complexity proof consists in proving that tasks may only pass a limited number of times from 3-unconstrained to any-constrained, and vice versa. This is ensured by sorting the sets of tasks by non increasing marginal gains using an AVL tree implementation. □

VI. Problem on Several Machines

We now consider the problem in which the tasks can be executed on m different machines. We consider a scenario in which machines have *different speeds* (e.g. servers, laptops, cellphones, IoT devices, etc.). We model this by introducing the accuracy function of a task j on a machine r of speed s_r and setting:

$$a_{j,r}(t) = a_j(s_r, t).$$

As an example, for the exponential accuracy function, we have: $a_{j,r}(t) = 1 - e^{-s_r \theta_j t}$.

A. Formulation as a Mixed Integer Program

The DSCT problem on several machines can be formulated as the mixed integer program given below.

Variables. We introduce the Boolean variables $x_{j,r}$, which indicate if task j is executed on machine r, for all $j \in J$ and $r \in M \stackrel{\text{def}}{=} \{1, \ldots, m\}$. The processing time is now chosen for each machine by the variables $t_{j,r} \in \mathbb{R}^+$.

Objective function. The processing time, t_j, of task j is given by $\sum_{r=1}^{m} t_{j,r}$. The accuracy of j thus is $a_j(\sum_{r=1}^{m} s_r \cdot t_{j,r})$. The objective function is: $\min \sum_{j=1}^{n} 1 - a_j(\sum_{r=1}^{m} s_r \cdot t_{j,r})$. As an example, for the exponential accuracy function, the objective is: $\min \sum_{j=1}^{n}(a_j^{\max} - a_j^{\min}) e^{-\frac{1}{\theta_j}\sum_{r=0}^{m} s_r \cdot t_{j,r}}$. We thus have:

$$\min \quad \sum_{j=1}^{n} \left[1 - a_j \left(\sum_{r=1}^{m} s_r \cdot t_{j,r}\right)\right] \tag{10}$$

$$\text{s.t.} \quad \sum_{i=1}^{j} t_{i,r} \leq d_j, \qquad j \in J, r \in M \tag{10a}$$

$$\sum_{r \in M} t_{j,r} \leq t_j^{\max}, \qquad j \in J \tag{10b}$$

$$t_{j,r} \leq x_{j,r} \cdot d_j, \qquad j \in J, r \in M \tag{10c}$$

$$\sum_{r \in M} x_{j,r} = 1, \qquad j \in J \tag{10d}$$

$$t_{j,r} \in \mathbb{R}^+, \qquad j \in J, r \in M \tag{10e}$$

B. NP-Hardness complexity

On the contrary to the problem on one machine which is polynomial, DSCT on several machines is formulated with binary variables. In fact, we have the following result.

Definition 1. *The decision problem of* DSCT *with m machines is to decide if there exists a schedule $\{t_{j,r} | j \in J, r \in M\}$ such that the global accuracy is greater than $x \in \mathbb{R}$, that it $\sum_{j \in J} a_j(t_j) \geq x$.*

Theorem 2. *The decision problem of* DSCT *(for any strictly increasing accuracy function) is NP-complete when the number of machines is $m \geq 2$.*

Proof. The proof is done via a reduction from the PARTITION SET PROBLEM and omitted due to lack of space. □

C. Approximation Algorithm

We propose here an approximation algorithm, DEADLINE SCHEDULING WITH COMPRESSIBLE TASKS - APPROXIMATION ALGORITHM (or DSCT-APPROX in short), presented in Algorithm 3, to solve DSCT with a proved guarantee, see Theorem 3. DSCT-APPROX consists in three main steps:

- Step 1 (Fractional relaxation). We first solve the fractional relaxation. It gives a schedule with maximum accuracy value $OPT_f \geq OPT$ in polynomial time, where OPT is the maximum accuracy of DSCT. In fact, an optimal fractional solution can be found by using the algorithm for one machine (Algorithm 1) with deadlines $d_j^1 = Sd_j$ and by taking $t_{j,r} = \frac{t_j^1}{s_r}$, see Lemmas 1 and 2).
- Step 2 (Rounding). We consider the tasks by non decreasing deadlines and assign them incrementally to the machine with the least amount of work. At the end of this step, each deadline may only be violated by a single task on every machine.
- Step 3 (Incremental shift). For each machine $r \in M$, we consider the tasks in the order of their execution on r. We denote by $t_{j,r}^{(i)}$ the starting time of task j on machine r. If a task j violates the deadline with a violation $v_j = t_{j,r}^{(i)} + t_{j,r} - d_j$, we cut the execution of the task done after the deadline, i.e., $t_{j,r} \leftarrow t_{j,r} - v_j$, and we shift the processing times of the all the tasks executed after j on r, i.e., $t_{i,r}^{(i)} \leftarrow t_{i,r}^{(i)} - v_j$ for all $i \geq j$.

We first show in Lemma 2, that Algorithm 2 solves the fractional relaxation of the problem. To this end, we use the preliminary result of Lemma 1. We then prove that Algorithm 3 is an approximation algorithm and give its approximation guarantee in Theorem 3.

Algorithm 2 Exact algorithm for solving the fractional relaxation

Input: List of task deadlines $[d_1, ..., d_n]$, of maximum processing times $[t_1^{\max}, ..., t_n^{\max}]$, and of machine speeds $[s_1, \ldots, s_m]$
Output: List of task processing times $[t_{j,r} : \forall j \in J, r \in M]$

```
S ← 0                                              ▷ The sum of all speeds
for r ∈ M do
    S ← S + s_r
for j ∈ J do
    d_j^1 ← S d_j
[t_1^1, ..., t_n^1] ← Algorithm 1([d_1^1, ..., d_n^1], [t_1^max, ..., t_n^max])
for j ∈ J do
    for r ∈ M do
        t_{j,r} ← t_j^1 / s_r
return [t_{j,r} : ∀j ∈ J, r ∈ M]
```

Lemma 1. *The* fractional relaxation *of the* DSCT *problem with m machines has an* optimal fractional solution *which allocates the same processing time on each machine, i.e., $t_{j,r_1}^{f*} = t_{j,r_2}^{f*}$ for all $r_1, r_2 \in M$.*

Proof. The omitted proof is done by checking that the symmetric version of an optimum solution is also optimum. □

Lemma 2. *Given an instance of* DSCT *with n tasks and m machines, Algorithm 2 provides an optimal solution of the fractional relaxation for the instance in time $O(n^3(\log n + c(n)) + nm)$. Recall that $c(n)$ is the amortized complexity of computing a value of τ, see Theorem 1.*

Proof. The omitted proof verifies that the original problem has the same optimum fractional solution as the problem with one machine of speed one but with deadlines $S \cdot d_j$ and maximum processing times $S \cdot t_j^{\max}$ $\forall j \in J$, where $S \stackrel{\text{def}}{=} \sum_{r \in M} s_r$. □

Theorem 3. *Algo 3 is an approximation algorithm with an absolute performance guarantee G:*

$$OPT - G \leq SOL \leq OPT,$$

Algorithm 3 DSCT-APPROX: Approximation algorithm for scheduling in several machines

Input: List of task deadlines $[d_1, \ldots, d_n]$, $[t_1^{\max}, ..., t_n^{\max}]$, and of machine speeds $[s_1, \ldots, s_m]$

Output: List of task processing times $[t_{j,r} : \forall j \in J, r \in M]$

1: Sort the tasks by non decreasing deadlines.
2: $\boldsymbol{t^f}$ = Algorithm 2$([d_1, \ldots, d_n], [t_1^{\max}, ..., t_n^{\max}], [s_1, \ldots, s_m])$ ▷ Compute the optimal fractional solution $\boldsymbol{t^f}$.
3: **for** $r \in M$ **do**
4: $S_r \leftarrow []$ ▷ List of tasks scheduled on machine r
5: $work_r \leftarrow 0$ ▷ $work_r$ amount of work on r
6: **for** $j \in J$ **do** ▷ Schedule each task on the machine with the least amount of work, $r_{\min}$
7: $r_{\min} \leftarrow \arg\min_{r \in M} work_m$
8: $t_{j,r_{\min}} \leftarrow \sum_{r=1}^{m} t_{j,r}^f$
9: $S_{r_{\min}} \leftarrow S_{r_{\min}} + [j]$
10: **for** $r \in M$ **do** ▷ Cut tasks violating their deadlines and shift the following ones.
11: **for** $j \in S_r$ **do**
12: **if** $t_{j,r}^{(i)} + t_{j,r} \geq d_j$ **then**
13: $v_j \leftarrow t_{j,r}^{(i)} + t_{j,r} - d_j$
14: $t_{j,r} \leftarrow t_{j,r} - v_j$ ▷ Cut task
15: **for** $i > j \in S_r$ **do** ▷ Shift the following tasks
16: $t_{i,r}^{(i)} \leftarrow t_{i,r}^{(i)} - v_j$
17: **return** $[t_{j,r} : \forall j \in J, r \in M]$

where OPT is the global accuracy of an optimal solution and SOL is the solution returned by Algo 3 and with

$$G = m \int_0^{\infty} \max_{\theta, j} \frac{\partial a_j(t, \theta)}{\partial t} dt.$$

For the exponential accuracy function:

$$G = mA \left(1 + \frac{1}{e} \ln \left(\frac{\theta_{\max}}{\theta_{\min}}\right)\right),$$

where $A \stackrel{def}{=} \max_{j \in J}(a_j^{\max} - a_j^{\min})$.
SOL is computed in time $O(n^3(\log n + c(n)) + n^2 m)$.

Proof. The algorithm starts from a optimal solution of the fractional relaxation of the problem of value OPT_f. We know that $OPT \leq OPT_f$.

The algorithm then assign tasks to the machine with the least amount of work. Note that on any machine $r \in M$, a deadline d may be only violated by a single task. Indeed, consider deadline d is violated on machine r by task i. When Algo 3 places a task $j > i$ with the same deadline d at step j in the `for` loop on line 6, there exists a machine r' such that $work_{r'} < d$. Indeed, $\sum_{r \in M} work_r + t_j^f = \sum_{i=0}^{j-1} t_{i,r}^f \leq md$, because the schedule is feasible for the fractional relaxation. Thus, Algo 3 does not assign j to machine r.

Finally, the algorithm incrementally cuts tasks violating their deadlines (For a cut task j, we note v_j the cut time.) and shifts by v_j the starting times of the tasks executed afterwards. We compute the difference between OPT_f and the total accuracy of the solution returned by the algorithm SOL.

The total lost accuracy, TLA, is the sum the lost accuracy for each cut task. Let us first bound the accuracy lost when cutting task j on a machine r of speed s_r. We denote by $t = d_j - t_{j,r}^{(i)}$ the execution time after the job was cut. We have

$$LA_j \stackrel{\text{def}}{=} a_j(s_r(t + v_j)) - a_j(s_r t)).$$

First, note that the function is decreasing in t due to concavity. Moreover, a cut task j was first shifted by $v_j^c \stackrel{\text{def}}{=} \sum_{i<j} v_i$. This part is executed. Thus, the worst case is when $t = v_j^c$, and $LA_j \leq a_j(s_r(v_j^c + v_j)) - a_j(s_r v_j^c)$. We now bound LA_j by the accuracy of the worst task that could be cut at time v_c. Note that the worst cut task depends on the time at which it is cut, see the discussion for the exponential accuracy function below for an illustration.

$$LA_j \leq \max_{i \in J}(a_i(s_r(v_j^c + v_j)) - a_i(s_r v_j^c)).$$

Now, we know that two functions $f(x)$ and $g(x)$ such that $f(x_1) = g(x_1)$ and $f'(x) \leq g'(x)$ for all $x \in [x_1, x_2]$ is such that $f(x) \leq g(x)$ for all $x \in [x_1, x_2]$. It gives

$$\begin{aligned} LA_j &\leq \max_{i \in J}(a_i(s_r(v_j^c + v_j)) - a_i(s_r v_j^c)) \\ &= \max_{i \in J} \int_{v_j^c}^{v_j^c + v_j} \frac{d}{dt} a_i(s_r t) v \; dt \leq \int_{v_j^c}^{v_j^c + v_j} \max_{i \in J} \frac{d}{dt} a_i(s_r t) dt \end{aligned}$$

Let us now bound the total lost accuracy TLA. We have

$$\begin{aligned} TLA &\stackrel{\text{def}}{=} \sum_{j=1}^{n_c} LA_j \leq \sum_{j=1}^{n_c} \int_{v_j^c}^{v_j^c + v_j} \max_{i \in J} \frac{d}{dt} a_i(s_r t) dt \\ &= \int_0^{v} \max_{i \in J} \frac{d}{dt} a_i(s_r t) dt = \int_0^{\infty} \max_{i \in J} \frac{d}{dt} a_i(s_r t) dt \end{aligned}$$

where n_c is the number of tasks which were cut at the end of the algorithm.

Exponential accuracy function. Let us compute $\max_{i \in J} \frac{d}{dt} a_i(s_r t)$. We have

$$\begin{aligned} a_i'(t) = \frac{d}{dt} a_i(s_r t) &= \frac{d}{dt} \left((a_i^{\max} - a_i^{\min})(1 - e^{-s_r \theta_i t}) + a_i^{\min}\right) \\ &= (a_i^{\max} - a_i^{\min}) s_r \theta_i e^{-s_r \theta_i t}. \end{aligned}$$

We now consider the function $a'(t, \theta) \stackrel{\text{def}}{=} A s_r \theta e^{-s_r \theta t}$, with $A = \max_{i \in J} a_i^{\max} - a_i^{\min}$. We have $\max_{i \in J} a_i'(t) \leq \max_{\theta} a'(t, \theta)$.

Let us find the value of θ for which $a'(t, \theta)$ is maximum, that is the one for which $\frac{d}{d\theta} a'(t, \theta) = 0$.

$$\frac{d}{d\theta} a'(t, \theta) = A s_r e^{-s_r \theta t} - A s_r^2 t \theta e^{-s_r \theta t} = A s_r (1 - s_r t \theta) e^{-s_r \theta t}.$$

The derivative is 0 for $\theta = \frac{1}{s_r t}$. This means that the worst task efficiency θ^* for a task cut after time t on a machine r is $\theta^* = \frac{1}{s_r t}$. For a set of tasks with maximum and minimum efficiencies, respectively $\theta_{\max}$ and $\theta_{\min}$, we have:

$$\theta^*(t) = \arg\max_{\theta} a'(t, \theta) = \begin{cases} \theta_{\max} & \text{when } t \leq \frac{1}{s_r \theta_{\max}} \\ \theta_{\min} & \text{when } t \geq \frac{1}{s_r \theta_{\min}} \\ \frac{1}{s_r t} & \text{otherwise.} \end{cases}$$

Note that it means that for small values of t, the worst tasks to be cut have maximum efficiency. However, for large values of t, it is the reverse: the worst cut tasks have minimum efficiency.

Let us now bound the total lost accuracy. We have

$$\begin{aligned} TLA &\leq \int_0^{\infty} \max_{i \in J} \frac{d}{dt} a_i(s_r t) dt \leq \int_0^{\infty} a'(t, \theta^*(t)) dt \\ &= \int_0^{\frac{1}{s_r \theta_{\max}}} a'(t, \theta_{\max}) dt + \int_{\frac{1}{s_r \theta_{\max}}}^{\frac{1}{s_r \theta_{\min}}} a'\left(t, \frac{1}{s_r t}\right) dt \\ &+ \int_{\frac{1}{s_r \theta_{\min}}}^{v} a'(t, s_r \theta_{\min}) dt. \end{aligned}$$

Recall that $a'(t, \theta) = A s_r \theta e^{-s_r \theta t}$. Computing each integral and adding up, we get: $TLA \leq A + \frac{A}{e} \ln\left(\frac{\theta_{\max}}{\theta_{\min}}\right)$.

As there are m machines, the global lost accuracy of the algorithm is bounded by $G \stackrel{\text{def}}{=} mA\left(1 + \frac{1}{e} \ln\left(\frac{\theta_{\max}}{\theta_{\min}}\right)\right)$.

The algorithm complexity is directly given by the one of Algorithm 2 and of the 3 final nested loops. □

VII. Experimental Evaluation

In this section, we evaluate the proposed approximation algorithm DSCT-APPROX described in Algorithm 3. We first show that the absolute guarantees given by Theorem 3 really correspond to a worst case scenario and that in practice DSCT-APPROX provides near optimal solutions. We then discuss the processing time of our algorithm and show that DSCT-APPROX provides near optimal solutions for medium to large instances in a limited time, even using a non optimized Python code. Last, we compare its performance with the one of state of the art solutions using no compression or a limited number of models. We show that using adjustable models allows to reach very significant gains in performance for a large range of scenarios.

Hardware Settings. During the experiments, we used a computer equipped with a CPU Intel Core i9-12900H and with a GPU NVIDIA RTX A2000.

Experimental scenarios and Problem instances. We generate synthetic sets of tasks based on the results of our experiments with OFA in Section III. We set the maximum accuracy to $a^{\max} = 0.82$, i.e., the highest value found by `ofa-resnet` on `ImageNet-1k`, and the minimum task accuracy to $a^{\min} = 1/1000$, the accuracy of a random guess. We fix the value of the minimum task efficiency $\theta_{\min} = 0.1$.

We then build scenarios with different (i) task heterogeneity and (ii) deadline tolerance levels. For (i), we vary the *task heterogeneity ratio* of a set of tasks, denoted as $\mu \stackrel{\text{def}}{=} \frac{\theta_{\max}}{\theta_{\min}}$ which measures the similarity between task efficiencies. When μ is chosen, so $\theta_{\max}$, for each task $j \in J$, we select its efficiency θ_j uniformly at random between θ_{min} and θ_{max}. Then, we compute its maximal processing time $t_j^{\max}$, in order to have $a_j(t_j^{\max}) = 0.82$. For (ii), we define the *deadline tolerance level*, denoted by ρ, as following: $\rho = \frac{m \cdot d_{\max}}{\sum_{j \in J} t_j^{\max}}$. The greater ρ is the more time is allocated for the tasks.

Baselines. We benchmark the performance of DSCT-APPROX with 3 different solutions:

- DSCT-UB, an upper bound solution provided by Algorithm 2 that solves the fractional relaxation of the problem DSCT problem on several machines.
- EDF-NoCompression: Here, compression is not allowed. Then, each task must be executed with its maximal processing time ($t_j^{\max}$), which corresponds to the neural network in its maximal configuration. The scheduling strategy used here is the EDF (Earliest Deadline First) for ordering the tasks combined to the `worst-fit` strategy [37], for deciding the machine on which the task should be executed.
- EDF-3CompressionLevels: The algorithm considers a discrete set of possible configurations for compressing the neural network. In this approach, we use 3 levels of compression, corresponding to 3 accuracy levels compared to the maximal configuration: 27%, 55% and 82%. For this algorithm, we implemented a strategy based on [27], which uses the EDF for ordering the tasks and implements a heuristic based on the accuracy for choosing the node and the compression level.

Study of the approximation guarantee. We first compare the performance of our algorithm to the absolute performance guarantee derived in Theorem 3, i.e. $G = mA(1 + \frac{1}{e}\ln(\mu))$. We thus make μ, the task heterogeneity ratio, vary (from 1 to 20). The guarantee is derived from a worst case analysis. We thus study here the algorithm performance over a large number of experiments (1000 for each value of μ). The deadline tolerance factor is fixed $\rho = 0.35$. We used $m = 2$ machines and $n = 100$ tasks. Fig. 2 presents the distribution over the experiments of two optimality gaps: (i) the *absolute performance guarantee* G, which is actually a *worst-case optimality gap*; and (ii) the *optimality gap* provided by the difference between the fractional relaxation DSCT-UB and the approximation algorithm DSCT-APPROX solutions. For the latter, we plot the *mean*, *min* and *max* differences over all the tests for a fixed μ. Recall that $A = 0.82$ and $m = 2$ here, thus $G = 1.64$ when $\mu = 1$. We observe that the optimality gaps increase with the task heterogeneity ratio as expected, but the gap with respect to the fractional relaxation solution is well below G. In our experiments, in average, the ratio between the *mean of the optimality gap* and G was 12.36%. This is expected as G comes from very specific scenarios, in which tasks with very short processing times and the worst accuracy are cut at each time when doing rounding. These scenarios are rare, as, even with 1000 experiments, the maximal optimality gaps were far from G.

Algorithm performances and processing times. We also tested the execution time of DSCT-APPROX against DSCT-Opt, which uses the cvx-MOSEK software [38], a widely used commercial solver which allows solving Mixed-Integer Programs (MIP). The results are described in Fig. 3. We considered two scenarios: keeping the number of tasks fixed ($n = 50$), and varying the number of machines and the reverse, fixing $m = 5$. We set the solver's time limit to 60 s. We observe that DSCT-APPROX can handle hundred

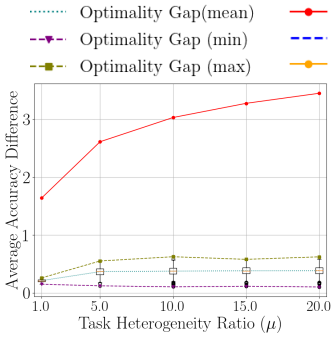
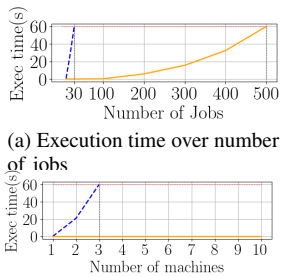


Fig. 2. Optimality gap (average accuracy difference between DSCT-UB and DSCT-APPROX) over 1000 experiments when varying the task heterogeneity ratio μ. Absolute guarantee G is given as a baseline.



(a) Execution time over number of jobs

(b) Execution time over number of machines

Fig. 3. Execution times of DSCT-APPROX vs DSCT-Opt [38] for instances with increasing (a) numbers of tasks, with $m = 5$ and (b) number of machines, with $n = 50$.

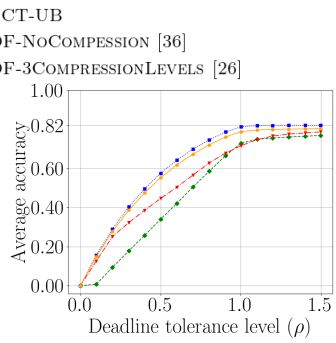
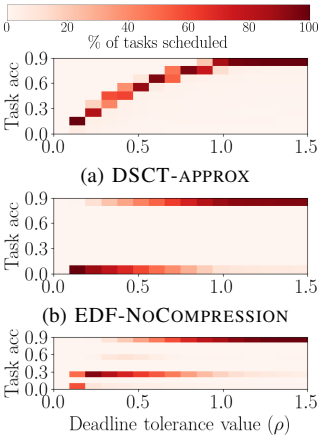


Fig. 4. Average task accuracy for DSCT-APPROX and the baselines as a function of the deadline tolerance level ρ, for $m = 10$.



(a) DSCT-APPROX

(b) EDF-NoCompression

(c) EDF-3CompressionLevels

Fig. 5. Scheduling Fairness: Heat maps represent the distribution of task accuracy values for sets of experiments with different tolerance ρ. The colors represent the percentage of tasks within an accuracy range.

of jobs before reaching the time limit. The MOSEK solver, however, can reach a maximal value of 30 jobs. Moreover, the solver manages scheduling the jobs within the time limit until 3 machines. For DSCT-APPROX, it is not impacted for the increasing number of machines, since the high complexity of the algorithm is during the fractional relaxation.

Comparison with benchmarks for different deadline tolerance levels. We now compare the global accuracy obtained by DSCT-APPROX with the ones reached by EDF-NoCompression [37] and EDF-3CompressionLevels [27]. To explore the different possible scenarios of usage, we vary the deadline tolerance level in a range between 0 and 1.5, with a step of 0.1. The number of machines is set to $m = 10$ and we consider $n = 100$ tasks with uniform efficiency $\theta_j = 0.1$. The average accuracy value reached by DSCT-APPROX, EDF-NoCompression, EDF-3CompressionLevels, and the fractional relaxation DSCT-UB are given in Fig. 4. Each point corresponds to an average over 1000 experiments.

We confirm that (i) when ρ is close to zero, all the methods have an average accuracy close to zero, since the space to schedule tasks is very limited; (ii) when $\rho \geq 1$, there is space enough to execute all tasks with their maximum processing time and, thus, maximum accuracy. All algorithms thus reach an accuracy of $a^{\max} = 0.82$. Between these values, we first observe that DSCT-APPROX return quasi-optimal solutions as its values are very close to the ones of the fractional relaxation. Secondly, DSCT-APPROX outperforms EDF-NoCompression [37] and EDF-3CompressionLevels [27] for each deadline tolerance value tested, as they do not use compression or only a limited number of models. In fact, the fractional relaxation DSCT-UB follows a function which is a composition of 1-exponential functions with limited supports. It thus has a close to exponentially decreasing marginal gain, when the two other algorithms have a (piecewise) linear behavior. The difference between DSCT-APPROX and EDF-NoCompression (and EDF-3CompressionLevels) performances is significant and reaches more than .18 point (.1) of accuracy, e.g. representing a gain of 57.9% (22.2%) for $\rho = 0.4$.

Fairness. A byproduct of allowing for task compression is that the allocation of processing times over tasks is *fairer* than without compression, in the sense that it allows them to reach similar accuracy values. When not using compression, a task is either executed, and reaches its maximum possible accuracy, or not, leading to a minimum one. We study this behavior in Fig. 5. For each set of experiments (corresponding to a specific value of deadline tolerance value, ρ), we report the distribution of reached accuracy values for all tasks. If we consider as an example, the case $\rho = 0.7$, we observe that more than 90% of the tasks reach an accuracy between 0.7 and 0.8 for DSCT-APPROX, and that only 2% of the tasks have an accuracy below 0.3, to be compared with 33% for EDF-3CompressionLevels and 28% for EDF-NoCompression. Thus, our algorithm allows (i) for having a better average accuracy (as seen before) and (ii) for not sacrificing tasks, which may be particularly important for scenarios in which tasks are requested by different users.

VIII. Conclusion and Discussion

We studied the problem of scheduling with compressible jobs. We analyzed and modeled the tradeoff between accuracy and processing time of a family of compressible neural networks. This technique can be used for several applications, such as real-time systems and task offloading. In this paper, we analyzed the Once-For-All architecture used for image classification.

We proposed an approximation algorithm for multiple machines scenario, which is a NP-hard problem. The algorithm proposed obtained a near-optimal value, outperforming the

baseline average accuracy by up to 57%. Also, we observed the algorithm presents smaller execution time compared to commercial MIP solvers.

We studied scheduling problems for specific jobs (classification inference tasks) and specific accuracy functions (concave, differentiable, and with exponentially decreasing marginal gains). However, we believe our work may have implications for a large range of problems.

First, the concavity of the accuracy function is a related to a very general and common phenomena which is the law of diminishing marginal utility [15].

Second, exponentially decreasing marginal gains can be found for a large variety of optimization problems. While general theoretical results on such phenomenon appear still far from the reach of current techniques for the theory of artificial neural networks, we argue that such dependency is supported by analogous phenomena in similar settings, like approximation error in Chebyshev's polynomials [34, Theorem 3.3], and classifier error of AdaBoost algorithm [35].

Extending our results to a wide range of optimization model, such as energy efficiency and communication latency, is an interesting avenue for future work.

REFERENCES

[1] H. Hua, Y. Li *et al.*, "Edge computing with artificial intelligence: A machine learning perspective," *ACM Computing Surveys*, 2023.

[2] S. Raj, H. Gupta *et al.*, "Scheduling dnn inferencing on edge and cloud for personalized uav fleets," in *IEEE/ACM 23rd International Symposium on Cluster, Cloud and Internet Computing (CCGrid)*, 2023.

[3] A. Shakarami, M. Ghobaei-Arani *et al.*, "A survey on the computation offloading approaches in mobile edge computing: A machine learning-based perspective," *Computer Networks*, vol. 182, p. 107496, 2020.

[4] G. Drainakis, P. Pantazopoulos *et al.*, "On the distribution of ml workloads to the network edge and beyond," in *IEEE Conference on Computer Communications Workshops (INFOCOM WKSHPS)*, 2021.

[5] W. Gao, Q. Hu *et al.*, "Deep learning workload scheduling in gpu datacenters: Taxonomy, challenges and vision," *arXiv preprint arXiv:2205.11913*, 2022.

[6] J. Lin, W.-M. Chen *et al.*, "Mcunet: Tiny deep learning on iot devices," in *Annual Conference on Neural Information Processing Systems (NeurIPS)*, 2020.

[7] A. Howard, M. Sandler *et al.*, "Searching for mobilenetv3," in *Proceedings of the IEEE/CVF international conference on computer vision*, 2019, pp. 1314–1324.

[8] Y. Cheng, D. Wang *et al.*, "Model compression and acceleration for deep neural networks: The principles, progress, and challenges," *IEEE Signal Processing Magazine*, vol. 35, no. 1, 2018.

[9] J. Yu and T. Huang, "Autoslim: Towards one-shot architecture search for channel numbers," 2019. [Online]. Available: https://arxiv.org/abs/1903.11728

[10] H. Cai, C. Gan *et al.*, "Once-for-all: Train one network and specialize it for efficient deployment," in *International Conference on Learning Representations*, 2020. [Online]. Available: https://openreview.net/forum?id=HylxE1HKwS

[11] S. Javaid, N. Javaid *et al.*, "Intelligent resource allocation in residential buildings using consumer to fog to cloud based framework," *Energies*, vol. 12, no. 5, p. 815, 2019.

[12] S. Jiang, Z. Ma *et al.*, "Scylla: Qoe-aware continuous mobile vision with fpga-based dynamic deep neural network reconfiguration," in *IEEE INFOCOM 2020-IEEE Conference on Computer Communications*.

[13] B. Fang, X. Zeng *et al.*, "Nestdnn: Resource-aware multi-tenant on-device deep learning for continuous mobile vision," in *Proceedings of the 24th Annual International Conference on Mobile Computing and Networking*, 2018, pp. 115–127.

[14] V. Nigade, P. Bauszat *et al.*, "Jellyfish: Timely inference serving for dynamic edge networks," in *2022 IEEE Real-Time Systems Symposium (RTSS)*. IEEE, 2022, pp. 277–290.

[15] A. A. Cournot, *Researches into the Mathematical Principles of the Theory of Wealth*. chez L. Hachette. Translated from French by New York: Macmillan Company, 1927 [c1897], 1838.

[16] C. L. Liu and J. W. Layland, "Scheduling algorithms for multiprogramming in a hard-real-time environment," *Journal of the ACM (JACM)*, vol. 20, no. 1, pp. 46–61, 1973.

[17] B. Chen, C. N. Potts *et al.*, "A review of machine scheduling: Complexity, algorithms and approximability," *Handbook of Combinatorial Optimization: Volume1–3*, pp. 1493–1641, 1998.

[18] R. Vickson, "Two single machine sequencing problems involving controllable job processing times," *AIIE transactions*, 1980.

[19] B. Alidaee and A. Ahmadian, "Two parallel machine sequencing problems involving controllable job processing times," *European Journal of Operational Research*, vol. 70, no. 3, pp. 335–341, 1993.

[20] R. Wu, W. Bao *et al.*, "Online task assignment with controllable processing time," *arXiv preprint arXiv:2305.04453*, 2023.

[21] D. Shabtay and M. Kaspi, "Parallel machine scheduling with a convex resource consumption function," *European Journal of Operational Research*, vol. 173, no. 1, pp. 92–107, 2006.

[22] Q. Liu and Z. Fang, "Learning to schedule tasks with deadline and throughput constraints," in *IEEE INFOCOM*, 2023, pp. 1–10.

[23] Y. Huang, F. Wang *et al.*, "Deepar: A hybrid device-edge-cloud execution framework for mobile deep learning applications," in *IEEE INFOCOM 2019-IEEE Conference on Computer Communications Workshops (INFOCOM WKSHPS)*. IEEE, 2019, pp. 892–897.

[24] L. Angelelli, A. A. Da Silva *et al.*, "Towards a multi-objective scheduling policy for serverless-based edge-cloud continuum," in *2023 IEEE/ACM 23rd International Symposium on Cluster, Cloud and Internet Computing (CCGrid)*. IEEE, 2023, pp. 485–497.

[25] F. Giroire, N. Huin *et al.*, "When network matters: Data center scheduling with network tasks," in *IEEE Conference on Computer Communications (INFOCOM)*. IEEE, 2019, pp. 2278–2286.

[26] M. S. Aslanpour, A. N. Toosi *et al.*, "Energy-aware resource scheduling for serverless edge computing," in *2022 22nd IEEE International Symposium on Cluster, Cloud and Internet Computing (CCGrid)*. IEEE.

[27] D. Lee and M. Song, "Quality-oriented task allocation and scheduling in transcoding servers with heterogeneous processors," *IEEE Transactions on Circuits and Systems for Video Technology*, vol. 32, no. 3, pp. 1667–1680, 2021.

[28] H. Zhang, G. Ananthanarayanan *et al.*, "Live video analytics at scale with approximation and {Delay-Tolerance}," in *14th USENIX Symposium on Networked Systems Design and Implementation (NSDI 17)*, 2017, pp. 377–392.

[29] T. S. Salem, G. Castellano *et al.*, "Towards inference delivery networks: Distributing machine learning with optimality guarantees," in *2021 19th Mediterranean Communication and Computer Networking Conference (MedComNet)*. IEEE, 2021, pp. 1–8.

[30] Y. Jin, L. Jiao *et al.*, "Provisioning edge inference as a service via online learning," in *2020 17th Annual IEEE International Conference on Sensing, Communication, and Networking (SECON)*. IEEE.

[31] S. S. Ogden and T. Guo, "{MODI}: Mobile deep inference made efficient by edge computing," in *USENIX Workshop on Hot Topics in Edge Computing (HotEdge 18)*, 2018.

[32] J. Deng, W. Dong *et al.*, "Imagenet: A large-scale hierarchical image database," in *2009 IEEE Conference on Computer Vision and Pattern Recognition*, 2009, pp. 248–255.

[33] K. He, X. Zhang *et al.*, "Deep residual learning for image recognition," in *Proceedings of the IEEE conference on computer vision and pattern recognition*, 2016, pp. 770–778.

[34] S. Sachdeva, "Faster Algorithms via Approximation Theory," *Foundations and Trends® in Theoretical Computer Science*, vol. 9, no. 2, pp. 125–210, 2013.

[35] R. E. Schapire and Y. Freund, *Boosting: Foundations and Algorithms*, ser. Adaptive Computation and Machine Learning Series. Cambridge, MA: MIT Press, 2012.

[36] S. P. Boyd and L. Vandenberghe, *Convex optimization*. Cambridge university press, 2004.

[37] Y.-W. Zhang, R.-K. Chen *et al.*, "Energy-aware partitioned scheduling of imprecise mixed-criticality systems," *IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems*, 2023.

[38] M. ApS, *The MOSEK optimization toolbox for Python manual. Version 10.0.*, 2022. [Online]. Available: https://docs.mosek.com/latest/pythonapi/index.html