2021 IEEE International Symposium on Dynamic Spectrum Access Networks (DySPAN) | 978-1-6654-1339-8/21/$31.00 ©2021 IEEE | DOI: 10.1109/DySPAN53946.2021.9677398

Joint Compression and Offloading Decisions for Deep Learning Services in 3-Tier Edge Systems

Minoo Hosseinzadeh*, Nathaniel Hudson*, Xiaobo Zhao†, Hana Khamfroush*, and Daniel E. Lucani‡

*Department of Computer Science, University of Kentucky (Lexington, KY, United States)
†Department of Information Technology, Uppsala University (Uppsala, Sweden)
‡Department of Engineering, Aarhus University (Aarhus, Denmark)

***Abstract*—Task offloading in edge computing infrastructure remains a challenge for dynamic and complex environments, such as Industrial Internet-of-Things. The hardware resource constraints of edge servers must be explicitly considered to ensure that system resources are not overloaded. Many works have studied task offloading while focusing primarily on ensuring system resilience. However, in the face of deep learning-based services, model performance with respect to loss/accuracy must also be considered. Deep learning services with different implementations may provide varying amounts of loss/accuracy while also being more complex to run inference on. That said, communication latency can be reduced to improve overall Quality-of-Service by employing compression techniques. However, such techniques can also have the side-effect of reducing the loss/accuracy provided by deep learning-based service. As such, this work studies a joint optimization problem for task offloading decisions in 3-tier edge computing platforms where decisions regarding task offloading are made in tandem with compression decisions. The objective is to optimally offload requests with compression such that the trade-off between latency-accuracy is not greatly jeopardized. We cast this problem as a mixed integer nonlinear program. Due to its nonlinear nature, we then decompose it into separate sub-problems for offloading and compression. An efficient algorithm is proposed to solve the problem. Empirically, we show that our algorithm attains roughly a 0.958-approximation of the optimal solution provided by a block coordinate descent method for solving the two sub-problems back-to-back.**

***Index Terms*—Task Offloading, Compression, Edge Computing, Deep Learning, Network Optimization, Industrial Internet-of-Things**

I. Introduction

Exponential technological growth has been a defining quality of the 21st century. This growth led to the development of the *Industrial Internet-of-Things* (IIoT) where smart sensors, devices, digital storage units, instruments, etc. are interconnected through the Internet to aid the needs of industry [1]. Industrial sectors (e.g., manufacturing [2]) can be supplemented by the IIoT by providing comprehensive data collection, sharing, and processing across network devices. A key enabling technology for IIoT is that of *Edge Computing* (EC) [3]. Under the EC paradigm, compute resources are pushed to the network edge to provide more opportunity for IIoT devices (e.g., smart sensors) to have data they collect/generate/sense processed more immediately than if they were to solely rely on a faraway central cloud server [4]. This is done by the deployment of edge servers (or edge servers) that process requests/data closer to end users. Following the growing popularity of EC, the notion of *Edge Intelligence* (EI) has also risen in popularity. EI pushes AI services, namely *Machine Learning* (ML) and *Deep Learning* (DL) services, to the network edge. EI allows for data collected/generated/sensed at the network edge to be immediately processed without needing to relay all IIoT-generated data to be sent to a faraway central cloud [5].

The benefits of EC [6] (e.g., reduced latency, increased scalability, improved reliability), making EI all the more attractive for providing reliable intelligent services [7], [8]. However, the hardware resources (e.g., communication, computation, and storage capacities) available at the edge servers are relatively constrained when compared to central cloud servers and these limitations must be explicitly considered [9]–[11]. Many works studying problems related to EC study the optimization of the resources equipped to edge servers to optimally serve user requests. One such problem is that of request offloading (or task offloading/scheduling). Request offloading to edge server-hosted services has been widely studied for both general [9], [12] and deep learning-based services [13], [14]. The problem of request offloading aims to decide where requests should be sent in the system to be processed (i.e., which edge server should serve a given request) while being mindful of system resources and incurred latency. Typically, the primary objective of offloading problems is to minimize latency experienced by end users [9], [15].

One viable approach for reducing the latency for transmitting data across communication channels is to reduce the size of data transmitted over the communication channel through compression techniques [16]–[18]. Beyond minimizing communication latency, compression has also been considered for the purposes of minimizing energy consumption incurred from data transmission in EC systems for general service requests [19]. However, the applicability of compression techniques faces an additional challenge for deep learning-based services. It is important to note that compression techniques can largely be split into 2 groups: *lossless* and *lossy*. Lossless compression does not compromise data quality when it is uncompressed; however lossy compression provides much greater data size reduction [20]. Since the performance of DL-based services relies on data quality, compression techniques

(Minoo Hosseinzadeh and Nathaniel Hudson contributed equally to this work.)

Authorized licensed use limited to: UNIVERSITY OF KENTUCKY. Downloaded on March 01,2022 at 16:24:36 UTC from IEEE Xplore. Restrictions apply.

that harm data quality can compromise the loss/accuracy of the DL-based service. Thus, there is a trade-off between how much we compress data in service requests to reduce latency while not jeopardizing model accuracy too greatly.

This paper studies an offloading problem with the objective of optimizing the latency-accuracy trade-off for requests for DL-based services in 3-tier EC platforms while considering compression. We define this problem, ***C**ompression and **O**ffloading **D**ecisions on the **E**dge* (CODE), as a *mixed integer nonlinear program*. The CODE problem is similar to the problem studied in a prior work [13]. The novelty of the CODE problem is the joint consideration of both compression and offloading for DL-based service requests while considering more than 1 service request at a time. Further, CODE assumes edge servers can offload requests to other edge servers through device-to-device (D2D) communication channels. Finally, this paper proposes a polynomial-time algorithm for approaching the CODE problem. Our algorithm is empirically shown to achieve a 0.958-approximation of the near-optimal solution which is given by a *block coordinate descent method* [21].

II. Related Work

Here, for the sake of brevity, we briefly summarize the literature related to task offloading problems in edge computing systems. Prior works studying the request offloading problem in 3-tier EC platforms typically employ optimization techniques (e.g., linear programming, stochastic modeling) to either maximize the number of requests that were served under edge server hardware constraints [9], [10] or minimize overall latency provided to end users submitting requests [22], [23]. There are few recent works that jointly study offloading and compression decisions in EC systems. Xu et al. in [19] jointly studies resource allocation, task offloading, and data compression under edge server resource constraints while aiming to minimize energy consumption. They transform a non-convex optimization model into a convex problem and apply convex optimization to solve this joint problem. Elgendy et al. in [24] consider a similar joint optimization for resource allocation, data compression, and security. They show their non-convex joint optimization is NP-*hard* and use relaxation and linearization techniques to solve a convex version of the problem. However, a limitation of these prior works is that they are for general services available in EC systems. They do not consider the loss/accuracy provided by a DL-based service. Hosseinzadeh et al. in [14] study task offloading in a 3-tier EC platform while considering the latency-accuracy trade-off as the objective. However, that work fails to consider compression to reduce latency without greatly compromising model loss/accuracy. Zhao et al. in [13] study request offloading in 3-tier EC platforms by considering a simple environment where only 1 request is consider. Additionally, only 1 edge server is considered to simplify the problem.

The main contributions of our work to the task offloading problem include: *(i)* design of a mixed integer nonlinear program that jointly considers offloading and compression decisions for requests for DL-based services that aims to optimize the latency-accuracy trade-off; *(ii)* consideration of a complex 3-tier EC platform with multiple service requests and device-to-device (D2D) communication among edge servers to collaboratively process requests; and *(iii)* design of a polynomial-time iterative algorithm for solving CODE that empirically achieves a 0.958-approximation of the optimal solution provided by directly solving CODE using a *block coordinate descent method* [21].

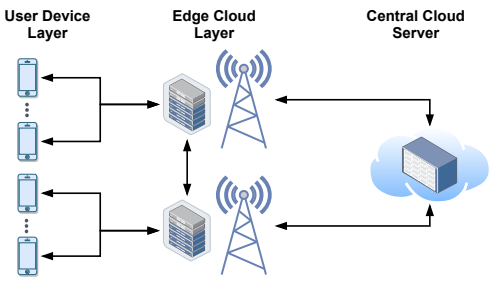

Fig. 1. Illustrative overview of our considered 3-tier architecture.

III. System Design & CODE Problem Definition

A. 3-Tier System Architecture

We consider a 3-tier edge computing system architecture. The topmost tier of this architecture is a central cloud server. The middle tier is a layer of edge servers deployed at the network edge. The bottom tier is the user device tier, comprised of IoT and IIoT devices. At any given time in the system, we consider a set of user requests submitted by the user devices in the user device layer. We denote the set of requests by $\mathcal{N}$. For this work, we assume each request $i \in \mathcal{N}$ corresponds with a single user (hence we may use "users" and "requests" interchangeably). We also denote the set of edge servers by $\mathcal{M}$. The edge server that directly covers a user device $i \in \mathcal{N}$ is denoted by w_i. The central cloud server will commonly be referred to by c throughout the paper.

Before defining the problem, we clarify two points of the system. First, edge servers and cloud servers have a maximum number of requests they are able to process in the considered time window. We refer to this as the *admission capacity*. The admission capacity for edge server j is denoted by v_j^e and the admission capacity for the central cloud server is denoted by v^c. All entities in the system (i.e., user devices, edge servers, and the central cloud server) have hardware resources associated with them. Because this paper is focused on offloading specifically, we only consider communication and computation resources. The notation for these will be presented later in §III-D.

B. Compression and Machine Learning Accuracy

In our prior work [13], we experimentally showed that image compression techniques based on DCT transforms, e.g., JPEG, reduce the expected accuracy of a machine learning model in a non-linear fashion when removing high frequency DCT coefficients, e.g., by giving them a zero value. More

Authorized licensed use limited to: UNIVERSITY OF KENTUCKY. Downloaded on March 01,2022 at 16:24:36 UTC from IEEE Xplore. Restrictions apply.

precisely, removing DCT coefficients initially has little to no effect in accuracy initially. There exists an inflexion point, dependent on the model, where additional compression results in dramatic loss of accuracy. We showed that this relationship between image compression and expected accuracy of a machine learning model can be fitted using the asymmetric Gompertz growth curve function [25], [26]:

$$g(s) = a \cdot \exp(-b \cdot \exp(-c \cdot s)) \tag{1}$$

where $a, b, c \in \mathbb{R}^+$ are parameters and $s \in [0, 1]$ is the scaling factor we use for data compression (i.e., $s = 0.95$ means the data size of the compressed image will be roughly 95% that of its original data size).

We will use the same compression method of [13] and the Gompertz approximation to control the compression level for image transmissions through the network during offloading to edge or cloud server. The goal is change the compression level to minimize the latency-accuracy metric of our system.

C. CODE Problem Definition

Here, we define the ***C**ompression and **O**ffloading **D**ecisions on the **E**dge* (CODE) problem as a *mixed integer nonlinear program* (MINLP). CODE jointly considers offloading and compression decisions. Thus, we consider 4 decision variables: 3 binary offloading decisions ($\mathbf{x}$, $\mathbf{y}$, $\mathbf{z}$) and 1 continuous compression decision ($\mathbf{s}$). First, we let decision variable $\mathbf{x} \triangleq (x_i)_{\forall i \in \mathcal{N}} = 1$ iff request i is processed locally using the requesting user's own local hardware resources, 0 otherwise. Second, we let decision variable $\mathbf{y} \triangleq (y_{ij})_{\forall i \in \mathcal{N}, j \in \mathcal{M}} = 1$ iff request i is processed by edge server j, 0 otherwise. Third, we let a set of decision variables $\mathbf{z} \triangleq (z_i)_{\forall i \in \mathcal{N}} = 1$ iff request i is processed on the remote central cloud, 0 otherwise. Finally, we consider a decision variable $\mathbf{s} \triangleq (s_i)_{\forall i \in \mathcal{N}} \in [0, 1]$ which corresponds to the compression rate used to compress the data associated with request i prior to processing (i.e., s_i represents the size of request after compression). Below, we present the definition of the CODE problem, dubbed as $\boldsymbol{\mathcal{P}1}$ for short:

$$\boldsymbol{\mathcal{P}1}: \min_{\mathbf{x},\mathbf{y},\mathbf{z},\mathbf{s}} \; L - \alpha A \tag{2}$$

$$\text{subject to } L \leq L_{\max} \tag{2a}$$

$$A \geq A_{\min} \tag{2b}$$

$$x_i + \Big(\sum_{j \in \mathcal{M}} y_{ij} \Big) + z_i = 1 \quad (\forall i \in \mathcal{N}) \tag{2c}$$

$$\sum_{i \in \mathcal{N}} y_{ij} \leq v_j^e \quad (\forall j \in \mathcal{M}) \tag{2d}$$

$$\sum_{i \in \mathcal{N}} z_i \leq v^c \tag{2e}$$

$$x_i, y_{ij}, z_i \in \{0, 1\} \quad (\forall i \in \mathcal{N}, j \in \mathcal{M}) \tag{2f}$$

$$0 \leq s_i \leq 1 \quad (\forall i \in \mathcal{N}) \tag{2g}$$

The pareto objective, defined in Eq. (2), aims to minimize the trade-off between average latency, L, and average accuracy, A (with α being a tunable hyperparameter). Note, both L and A are functions of offloading ($\mathbf{x}$, $\mathbf{y}$, $\mathbf{z}$) and compression decisions ($\mathbf{s}$). Constraint (2a) ensures that the average latency is less than or equal to the maximum allowed latency ($L_{\max}$). Similarly, Constraint (2b) guarantees that the average accuracy is greater than or equal to the minimum required accuracy ($A_{\min}$). Constraint (2c) guarantees that each request will be processed locally, by 1 edge server, or the central cloud. Constraints (2d) and (2e) guarantee that the admission capacity of the edge servers and the central cloud server are not exceeded. Note, requests that cannot be processed on an edge server or the central cloud due to the Constraints (2d) and (2e) will be processed locally. The last two constraints ensure that the decision variables are in the defined range.

CODE Problem Complexity. A robust proof of NP-hardness is beyond the scope of this paper. With the problem being a *mixed integer nonlinear program* (MINLP), it has been shown that both convex and non-convex MINLPs in general are NP-*hard* [27]. Hence, we assume that the CODE problem is difficult to directly solve for the optimal solution.

D. Defining Average Accuracy & Latency

1) Accuracy Definition: Each entity in the system (i.e., user devices, edge servers, and central cloud) host a ML model to perform some task (namely, image classification). However, the models can vary across these entities and thus it follows that the accuracy provided by these entities varies as well. Thus, we assume that we have fitted Gompertz functions to estimate the accuracy of the ML model hosted at each entity in the system similar to our initial results in [13]. Using these fitted functions and the compression decisions, $\mathbf{s}$, we can compute the average accuracy, A, which is used in the objective for $\boldsymbol{\mathcal{P}1}$. Its definition is found below:

$$A = \frac{1}{|\mathcal{N}|} \sum_{i \in \mathcal{N}} \left(x_i g_i^u(1) + \sum_{j \in \mathcal{M}} y_{ij} g_j^e(s_i) + z_i g^c(s_i) \right) \tag{3}$$

where $g_i^u(\cdot)$ is the Gompertz function fitted for user i's local ML model, $g_j^e(\cdot)$ is the Gompertz function fitted for the ML model hosted at edge server j's, and $g^c(\cdot)$ is the Gompertz function fitted for the ML model hosted at the central cloud. Note that no compression is ever performed if the request is processed locally (i.e., $g_i^u(1)$). The definitions of the individual Gompertz functions are based on fitted parameters and Eq. (1).

2) Latency Definitions: The latency incurred to complete a request i is the sum of both transmission/communication latency and computation latency. In $\boldsymbol{\mathcal{P}1}$, one of the goals is to minimize the *average latency*, L, by making decisions related to compression and offloading. As such, latency is a function of both compression and offloading decisions. The definition we consider for average latency can be found below:

$$L = \frac{1}{|\mathcal{N}|} \sum_{i \in \mathcal{N}} \left(x_i L^u(i) + \sum_{j \in \mathcal{M}} y_{ij} L^e(i, j) + z_i L^c(i) \right) \tag{4}$$

where $L^u(i)$ is the latency to locally process request i, $L^e(i, j)$ is the latency to process request i on edge server j, and $L^c(i)$ is the latency to process request i on the central cloud server.

Authorized licensed use limited to: UNIVERSITY OF KENTUCKY. Downloaded on March 01,2022 at 16:24:36 UTC from IEEE Xplore. Restrictions apply.

Local Latency. If a task is processed locally, then latency is only a function of local compute latency since no data transmission is needed to process the task on another device. Thus, we define local latency on the user-side for user i's request, $L^u(i)$, as the following,

$$L^u(i) = \frac{c_i}{f_i^u} \tag{5}$$

where c_i is the CPU clock cycles needed to process user i's request and f_i^u is the CPU frequency of user i's device.

User-to-Edge Latency. Tasks processed on the edge require both communication and computation to be completed. First, the user i's request must be transmitted to their covering edge server, w_i, and either processed there or offloaded to another edge server through device-to-device communication channels. Once the request is received by the final edge server, it will be processed, thus incurring additional compute latency. Thus, we define the *user-to-edge latency*, $L^e(i,j)$, as the latency for user i's request to be communicated and processed by edge server j. It is formally presented below,

$$L^e(i,j) = \begin{cases} \frac{s_i d_i}{R_i^u} + \frac{c_i}{f_j^e} & \text{if } j \equiv w_i \\ \left(\frac{s_i d_i}{R_i^u} + \frac{s_i d_i}{R^e_{w_i j}}\right) + \frac{c_i}{f_j^e} & \text{otherwise.} \end{cases} \tag{6}$$

The first case occurs when j is user i's covering edge server (w_i). In the first case, only one communication hop and processing is needed — where R_i^u is the data transmission rate from user i to its covering edge server and f_j^e is the CPU frequency on edge server j. The latter case covers the scenario where j is not user i's covering edge server, thus requiring an additional hop of communication. Here, we note that $R^e_{w_i j}$ is the bit rate between edge servers w_i and j and d_i represents the original size of data in request i.

User-to-Edge-to-Cloud Latency. Finally, we consider the *user-to-edge-to-cloud* latency, $L^c(i)$, for any task i that is offloaded to the central cloud server. It is defined below:

$$L^c(i) = \left(\frac{s_i d_i}{R_i^u} + \frac{s_i d_i}{R^c_{w_i}}\right) + \frac{c_i}{f^c}. \tag{7}$$

where $R^c_{w_i}$ is the bit-rate between user i's covering edge server and the central cloud and f^c is the CPU frequency at the central cloud.

IV. Problem Decomposition & Heuristic Design

Because $\boldsymbol{\mathcal{P}1}$ (provided in §IV) is a nonlinear optimization, it is hard to solve. As such, rather than solve it directly, we instead choose to decompose it into two sub-problems, $\boldsymbol{\mathcal{P}1.1}$ and $\boldsymbol{\mathcal{P}1.2}$, that separately focus on the offloading decisions and compression decision (respectively). Further, we will solve both sub-problems back-to-back using a *block coordinate descent method* [21] to attain a near-optimal solution for $\boldsymbol{\mathcal{P}1}$. We define the decomposed sub-problems for $\boldsymbol{\mathcal{P}1}$ below:

$$\boldsymbol{\mathcal{P}1.1}: \min_{\mathbf{x},\mathbf{y},\mathbf{z}} \; L - \alpha A \tag{8}$$

$$\text{subject to Constraints (2a)-(2f)} \tag{8a}$$

Algorithm 1: Proposed PCODE Algorithm

Input : Input parameters to CODE, $\lambda \geq 2$ (int)
Output: Offloading/compression decisions ($\mathbf{x}$, $\mathbf{y}$, $\mathbf{z}$, $\mathbf{s}$).

Init decision variables $\mathbf{x}$, $\mathbf{y}$, $\mathbf{z}$, $\mathbf{s}$;
Init total latency $\sigma_L \leftarrow 0$, total accuracy $\sigma_A \leftarrow 0$;
Init total number of processed requests $\eta \leftarrow 0$;
Init $counter^c \leftarrow 0$, $counter_j^e \leftarrow 0$ ($\forall j \in \mathcal{M}$);
$\mathbf{C} \leftarrow$ row-vect. of λ evenly-spaced values from 0 to 1;
foreach *request* $i \in \mathcal{N}$ *in a random order* **do**
  `/* Consider local processing */`
  $\pi_u^* \leftarrow \mathbb{E}[\text{obj}_i^u]$;
  $x_i \leftarrow 1$, $s_i \leftarrow 1$;
  $L_i \leftarrow L^u(i)$, $A_i \leftarrow g_i^u(1)$;
  `/* Consider offloading to edge */`
  foreach $j \in \mathcal{M}$ **do**
    if $counter_j^e < v_j^e$ **then**
      $\psi_{ej}^* \leftarrow \arg\min_{s \in \mathbf{C}} \mathbb{E}[\text{obj}_{ij}^e | s]$;
      $\pi_{ej}^* \leftarrow \mathbb{E}[\text{obj}_{ij}^e | \psi_{ej}^*]$;
    else
      $\pi_{ej}^* \leftarrow \infty$;
  $j^* \leftarrow \arg\min_{j \in \mathcal{M}} \pi_{ej}^*$;
  if *edge offloading is better, i.e.,* $\pi_{ej^*}^* < \pi_u^*$ **then**
    $y_{ij^*} \leftarrow 1$, $s_i \leftarrow \psi_{ej}^*$;
    $x_i \leftarrow 0$;
    $L_i \leftarrow L^e(i, j^*)$, $A_i \leftarrow g_{j^*}^e(\psi_{ej^*}^*)$;
  `/* Consider offloading to cloud */`
  if $counter^c < v^c$ **then**
    $\psi_c^* \leftarrow \arg\min_{s \in \mathbf{C}} \mathbb{E}[\text{obj}_i^c | s]$;
    $\pi_c^* \leftarrow \mathbb{E}[\text{obj}_i^c | \psi_c^*]$;
    if *cloud offloading is best, i.e.,* $(\pi_c^* < \pi_u^*) \wedge (\pi_c^* < \pi_{ej^*}^*)$ **then**
      $z_i \leftarrow 1$, $s_i \leftarrow \psi_c^*$;
      $x_i \leftarrow 0$, $y_{ij^*} \leftarrow 0$;
      $L_i \leftarrow L^c(i)$, $A_i \leftarrow g^c(\psi_c^*)$;
  Increment *counter* variables for edge and cloud based on offloading decision;
  $\sigma_L \leftarrow \sigma_L + L_i$, $\sigma_A \leftarrow \sigma_A + A_i$, $\eta \leftarrow \eta + 1$;
return $\mathbf{x}$, $\mathbf{y}$, $\mathbf{z}$, $\mathbf{s}$;

$$\boldsymbol{\mathcal{P}1.2}: \min_{\mathbf{s}} \; L - \alpha A \tag{9}$$

$$\text{subject to Constraints (2a), (2b), (2g)} \tag{9a}$$

$$A \geq 0.368 \cdot A_{\max}^p \quad (\forall p \in \{u, e, c\}) \tag{9b}$$

where $A_{\max}^p$ represents the maximum provided accuracy by the user layer ($p = u$), edge layer ($p = e$), and cloud layer ($p = c$). Since the Gompertz function has both concave and convex parts, we introduce the constraint (9b) to guarantee that we only consider the concave part, which is convex in the negative form. Because the objective of this problem is convex, the constraints should be concave — meaning a nonlinear solver can solve $\boldsymbol{\mathcal{P}1.2}$ directly.

A. Efficient Algorithm Design

We propose an efficient algorithm to solve $\boldsymbol{\mathcal{P}1}$. This algorithm approaches $\boldsymbol{\mathcal{P}1}$ in an iterative manner by considering each user request $i \in \mathcal{N}$ and greedily makes offloading

Authorized licensed use limited to: UNIVERSITY OF KENTUCKY. Downloaded on March 01,2022 at 16:24:36 UTC from IEEE Xplore. Restrictions apply.

and compression decisions based on the expected gain to the objective of $\boldsymbol{\mathcal{P}1}$. The pseudocode for the proposed algorithm is presented in Algorithm 1. Before explaining the pseudo code line-by-line, we introduce some mathematical definitions on which PCODE is based upon. First, to adhere to the $L_{\max}$ and $A_{\min}$ constraints (i.e., constraints (2a) and (2b)), our algorithm uses a *moving average* that tracks the expected average latency and average accuracy for requests as decisions are made. For the moving average, we denote the total latency and total accuracy by σ_L and σ_A, respectively. Next, we denote the number of requests processed thus far by η.

1) Approximating objective values: The proposed algorithm relies on approximating how an offloading and compression decision for a single user will affect the global objective using a moving average technique. As such, we define how these are approximated. Requests can be processed in 3 ways: local processing, edge-based processing, and cloud-based processing. First, the estimated objective if a request i is processed locally is computed by

$$\mathbb{E}[\text{obj}_i^u] \triangleq \frac{\sigma_L + L^u(i)}{\eta+1} - \alpha \cdot \frac{\sigma_A + g_i^u(1)}{\eta+1} \tag{10}$$

where σ_L is the summation of latency across all requests processed thus far based on prior offloading/compression decisions, σ_A is similar but for the summation of accuracies for previously processed requests, and η is the number of previously processed requests. Next, we approximate the global objective if a request i is offloaded to edge server j using compression s by

$$\mathbb{E}[\text{obj}_{ij}^e|s] \triangleq \begin{cases} \frac{\sigma_L + L^e(i,j|s_i=s)}{\eta+1} - \alpha \cdot \frac{\sigma_A + g_j^e(s_i=s)}{\eta+1} & I^e(i,j) \\ \infty & \text{otherwise} \end{cases} \tag{11}$$

where $I^e(i,j)$ is an indicator function such that $I^e(i,j) = 1$ if and only if $\left(\frac{\sigma_L + L^e(i,j|s_i=s)}{\eta+1} \leq L_{\max}\right) \wedge \left(\frac{\sigma_A + g_j^e(s_i=s)}{\eta+1} \leq A_{\min}\right)$, 0 otherwise. Finally, we approximate the global objective if a request i is offloaded to the central cloud using compression s by

$$\mathbb{E}[\text{obj}_i^c|s] \triangleq \begin{cases} \frac{\sigma_L + L^c(i|s_i=s)}{\eta+1} - \alpha \cdot \frac{\sigma_A + g^c(s_i=s)}{\eta+1} & I^c(i) \\ \infty & \text{otherwise} \end{cases} \tag{12}$$

where $I^c(i)$ is an indicator function such that $I^c(i) = 1$ if and only if $\left(\frac{\sigma_L + L^c(i|s_i=s)}{\eta+1} \leq L_{\max}\right) \wedge \left(\frac{\sigma_A + g^c(s_i=s)}{\eta+1} \leq A_{\min}\right)$, 0 otherwise.

2) Stepping through the Algorithm: PCODE's input includes the system parameters and an integer $\lambda \geq 2$ which is used to generate a $(1 \times \lambda)$-row vector ($\mathbf{C}$) of evely-spaced values from 0 to 1 (e.g., if $\lambda = 3$ then $\mathbf{C} = [0.0, 0.5, 1.0]$). $\mathbf{C}$ is used as a quantized compression value search space. Larger λ values increase PCODE's complexity but provides more space to find good compression decisions for PCODE. With that said, PCODE starts by initializing decision variables and other supplemental variables (lines 1-5). Line 6 starts a loop that iterates through each request $i \in \mathcal{N}$ in a random order. At the start of the loop, lines 7-9 approximate the gain towards the objective for processing request i locally. Then, lines 10-16 iteratively do the same process but for each of the edge servers $j \in \mathcal{M}$ and identifies the edge server that maximizes the approximated objective value for request i (line 16). On lines 17-20, PCODE determines if processing request i on the edge is better for the objective than local processing. If so, then offloading and compression decisions are changed accordingly (lines 18-20). Then, lines 21-27 do the same for cloud-based processing and change offloading and compression decisions if processing request i is shown to be the best choice (lines 24-27). Lines 28 and 29 then update the variables maintained throughout the loop to approximate the objective and respect admission capacities.

Proposition 1. *PCODE is a polynomial-time algorithm with an asymptotic runtime complexity of* $O(\lambda \cdot |\mathcal{N}| \cdot |\mathcal{M}|)$.

Proof. This can be seen simply be first noting that the outer loop (lines 6-29) takes place exactly $|\mathcal{N}|$ times (i.e., number of requests). For each iteration through this outer loop, PCODE also iterates through each of the edge servers (i.e., a multiplicative of $|\mathcal{M}|$) to consider edge processing. Finally, on line 12 (within the loop iterating through $\mathcal{M}$), $\arg\min(\cdot)$ iterates over the compression space which has λ elements (as per its definition). Since the considering local and cloud processing only occurs once in each iteration through the main outer loop, their complexity is constant and does not contribute to the overall runtime. Thus, the runtime is $O(\lambda \cdot |\mathcal{N}| \cdot |\mathcal{M}|)$. This concludes the proof. □

The complexity of optimal solution due to having a continuous decision variable ($\mathbf{s}$) cannot be exactly calculated. We have observed some cases of 9.5 hours running time, while our proposed PCODE algorithm runs in 0.7 seconds, for the same test.

V. Experimental Design

A. Benchmark Algorithms/Heuristics

We demonstrate the efficacy of our proposed algorithm (presented in §IV-A1) against the following benchmarks:

*1) Optimal (*OPT*):* It solves $\boldsymbol{\mathcal{P}1.1}$ and $\boldsymbol{\mathcal{P}1.2}$ back-to-back over several iterations using *block coordinate descent* [21] method which minimizes the objective with respect to one block at a time while the other block is fixed.

*2) Optimal_2 (*OPT_2*):* Its approach is similar to OPT with this difference that it first solves $\boldsymbol{\mathcal{P}1.2}$ and then $\boldsymbol{\mathcal{P}1.1}$. For both OPT and OPT_2, we stop running each $\boldsymbol{\mathcal{P}1.1}$ and $\boldsymbol{\mathcal{P}1.2}$ back-to-back when either the results in terms of objective converges or a threshold of back-to-back running passes which is set to 10, here, due to computationally expensive cost of running these two algorithms. So, the solver might not be able to find the optimal solution in some cases.

3) No-Compression (NC): It provides optimal offloading decision assuming that no compression is allowed. It uses the solver to solve $\boldsymbol{\mathcal{P}1.1}$ with $s_i = 1$ $(\forall i \in \mathcal{N})$ only one round and finds the optimal decision for offloading part of the problem.

Authorized licensed use limited to: UNIVERSITY OF KENTUCKY. Downloaded on March 01,2022 at 16:24:36 UTC from IEEE Xplore. Restrictions apply.

4) Fixed-Compression-50 (FC): It solves the offloading problem with the assumption that compression is allowed only at a fixed size of $s_i = 0.5$ $(\forall i \in \mathcal{N})$.

*5) Random-Serving-Optimal-Compression (*RSOC*):* It provides near-optimal compression solution using similar approach to PCODE by searching compression space assuming random offloading decision. It randomly selects a server (user/edge/cloud) to serve the requests. If the server is either edge server or the cloud server and it has enough admission capacity, it selects the minimum possible data size which minimizes the objective; else, it selects another server randomly and repeats the process again.

*6) Random-Serving-Random-Compression (*RSRC*):* It randomly selects one of the servers (user/edge/cloud). If the server is either edge server or cloud server and it has enough admission capacity, it selects a random size of compression from the range of $(0, 1]$; else, it selects another server randomly and repeats the process again.

7) Worst-case (W): This algorithm provides a worst case bound for the objective value. It works similar to PCODE algorithm with this difference that it always selects the maximum objective value, i.e., maximizing the average latency and minimizing the average accuracy.

B. Environment Setup

We use Python language to simulate the problem. We use Pyomo to solve the optimization problem with glpk and ipopt solvers [28]. We consider a simulation environment with the following setup. We simulate the accuracy of DL models using a Gompertz function for the cloud server, edge servers, and users' devices. These is used to predict the DL model's accuracy based on compression decisions (see §III-B). The Gompertz arguments used for the DL model placed on the cloud server, $a = 0.95$, $b = 20$, and $c = 1$; for the DL model placed on the edge servers, $a = 0.70$, $b = 6$, and $c = 18$; and for the DL model placed on the user devices, $a = 0.45$, 50, and $c = 20$. These are similar to the curves fitted for state-of-the-art DL models in our prior work [13]. Next, the CPU clock cycles to process request i are uniformly sampled from $c_i \in [150, 156]$ $(\forall i \in \mathcal{N})$. The size of request i's data (in bytes) is uniformly sampled from $d_i \in [150, 156]$ $(\forall i \in \mathcal{N})$. The data transmission rate (bytes/sec) from each user i to their covering edge server, w_i, is uniformly sampled from $R_i^u \in [150, 175]$ $(\forall i \in \mathcal{N})$. Then, the CPU frequency of user i's device is uniformly sampled from $f_i^u \in [5, 8]$ $(\forall i \in \mathcal{N})$. Data transmission rate (bytes/sec) for D2D communication between edge servers j and j' is uniformly sampled from $R_{jj'}^e \in [200, 206]$ $(\forall j, j' \neq j \in \mathcal{M})$ and each edge server has a CPU frequency uniformly sampled from $f_j^e \in [500, 506]$ $(\forall j \in \mathcal{M})$. Further, admission capacities for edge servers (i.e., number of requests that can be served in an instance of the problem) are randomly sampled from $v_j^e \in [5, 7]$ $(\forall j \in \mathcal{M})$. Finally, data transmission rates (btyes/sec) from the edge servers to the cloud server are uniformly sampled from $R_j^c \in [4, 6]$ $(\forall j \in \mathcal{M})$, the CPU frequency of the cloud server is uniformly sampled from $f^c \in [10000, 10007]$ in each trial. The cloud's admission capacity is fixed at $v^c = 60$.

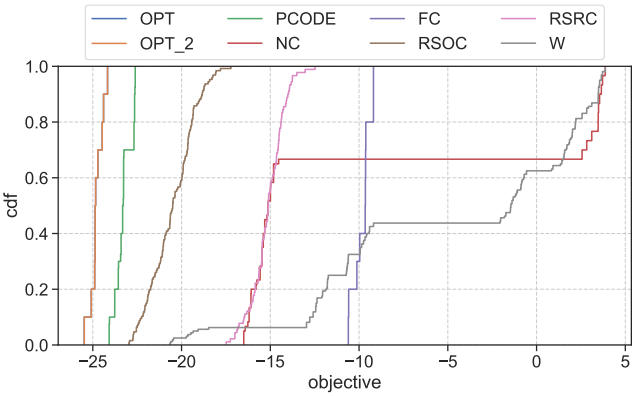

Fig. 2. CDF of Objectives (OPT and OPT_2 are overlapping and are the leftmost curves)

We fix the seed and repeat each test 10 times to remove the effect of randomness. The total number of user requests, $|\mathcal{N}|$, is set to 150 where each of requests is associated with one user and they are randomly connected to the edge servers. The number of edge servers, $|\mathcal{M}|$, is set to 10. We fix the α to 50, $L_{max} = 50$, and $A_{min} = 0.5$, unless otherwise stated. We set the λ, the size of vector C of generated compressed data, for both our proposed algorithm and RSOC equal to 11.

VI. Results

A. Impact of $A_{\min}$ & $L_{\max}$ on the Objective Value

We change the $A_{\min}$ from 40% to 70% and & $L_{\max}$ from 20 to 50 ms to evaluate the effect of QoS metrics– average latency and average accuracy– on the objective value. We set the α equal to 50. Fig. 2 represents the Cumulative Distribution Function (CDF) of the objective value of the proposed problem obtained by each algorithm when A_{min} and L_{max} are changed. The OPT and OPT_2 converge to the same results. Therefore, they show same values on the plot. As shown in Fig. 2, the proposed algorithm provides near optimal results in terms of objective value even when A_{min} and L_{max} are changed. While the performance of baseline algorithms decreases when A_{min} and L_{max} are changed.

B. Impact of α on Compression Decision (s) and the Objective Value

We fix the $L_{\max}$ to 50 and $A_{\min}$ to 0.5 and change the α in the range of 0.1 to 500 to evaluate the effect of α on compression decision and the objective value. The objective value of all algorithms based on different α values are provided in Table. I. The results imply that PCODE achieves near optimal solutions. Given the results presented in Table. I, our proposed PCODE algorithm can empirically achieve in average 0.958-approximation of the optimal solution provided by OPT and OPT_2.

Authorized licensed use limited to: UNIVERSITY OF KENTUCKY. Downloaded on March 01,2022 at 16:24:36 UTC from IEEE Xplore. Restrictions apply.

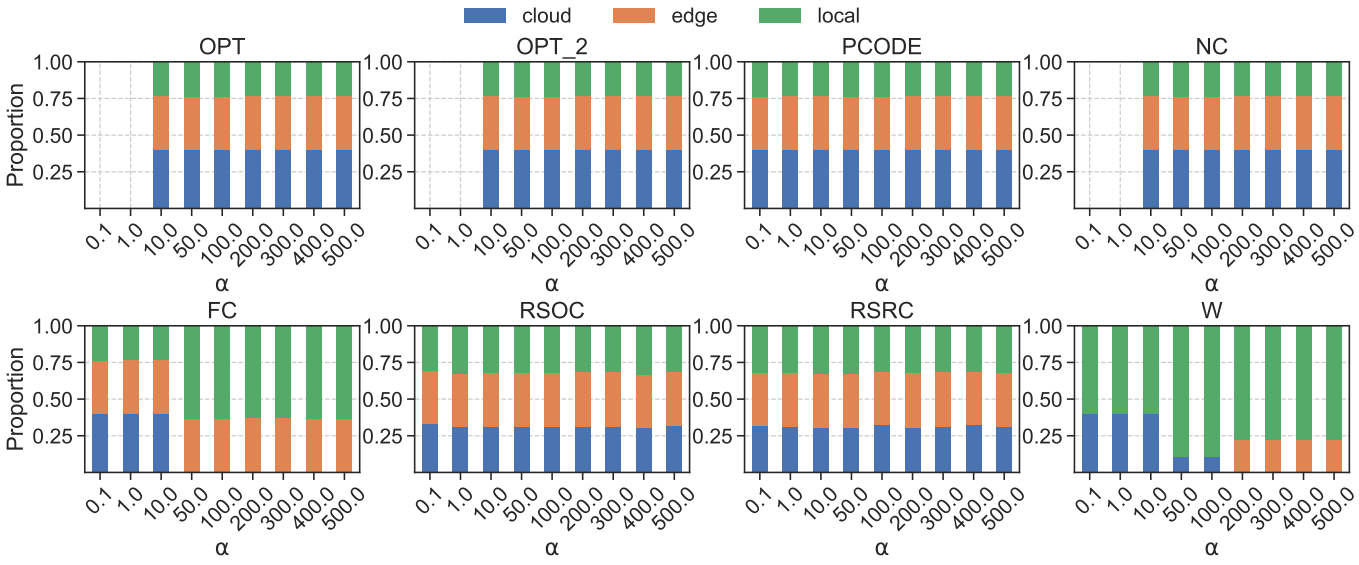

Fig. 3. Impact of α on the proportion of served requests by each layer

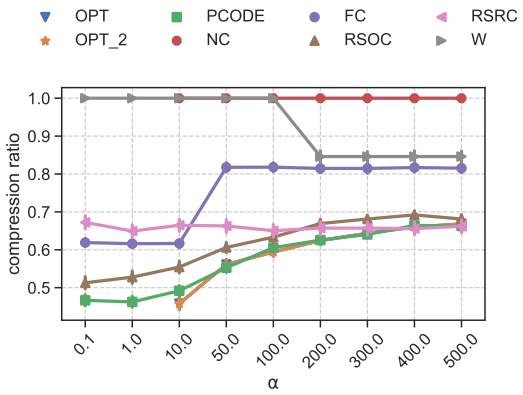

Fig. 4. Impact of α on the compression decision

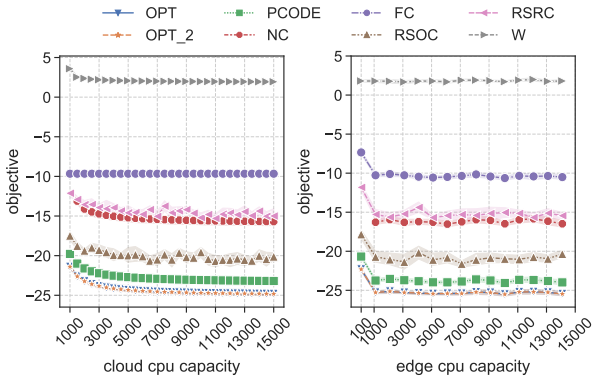

Fig. 5. Impact of edge servers and cloud server CPU on objective value

The effect of α on the compression decision of requests is shown in Fig. 4 where the y-axis represents the average compression ratio of all requests and the x-axis represents different α values. When α increases, the accuracy dominates the latency and the compression ratio, s_i, is increased which is intuitive given that larger data size provides better accuracy. Fig. 3 represents the portion of requests served by each layer– cloud, edge, and users. As shown, the optimal solutions and the proposed algorithm provide a trade-off between the amount of resources used by each layer in the network while most of baseline algorithms do not. Although the RSOC and RSRC provide a trade-off between a portion of consumed resources in the networks, their objective value is worse than our proposed algorithm, PCODE, based on Table. I. Additionally, the solver cannot find a solution for OPT and OPT_2 when $\alpha < 10$ because we limited the number of back-to-back running of $\boldsymbol{\mathcal{P}1.1}$ and $\boldsymbol{\mathcal{P}1.2}$. While our proposed algorithm can find a solution for these cases in less than a second.

TABLE I
THE AVERAGE OBJECTIVE VALUE OF ALL ALGORITHMS BASED ON DIFFERENT α VALUES (NOTE THAT "-" MEANS THAT ALGORITHM DID NOT FIND VALID SOLUTION FOR THAT CASE.)

	α Values				
Algorithm	1.0	10	50	100	200
OPT	-	3.231	-24.779	-61.231	-135.037
OPT_2	-	3.231	-24.779	-61.231	-135.037
PCODE	10.631	4.921	-23.260	-59.705	-133.419
RSOC	11.959	6.411	-20.301	-54.546	-124.137
RSRC	14.263	9.170	-14.752	-44.173	-102.950
NC	-	14.056	-15.547	-52.739	-127.165
FC	13.408	6.483	-9.820	-36.756	-91.287
W	29.063	24.167	1.750	-23.202	-79.416

C. Impact of Hardware Resources

We changed the CPU capacity of both the cloud layer and the edge layer. We fixed the edge CPU capacity across all edge servers. Fig. 5 represents the effect of cloud server

Authorized licensed use limited to: UNIVERSITY OF KENTUCKY. Downloaded on March 01,2022 at 16:24:36 UTC from IEEE Xplore. Restrictions apply.

CPU and edge servers CPU on the objective value of all algorithms. When either the cloud server CPU or the edge servers CPU increases, the objective improves. Fig. 5 implies that the objective value of our proposed PCODE algorithm is close to OPT and OPT_2 in all cases of different CPU capacities for both edge servers and the cloud server.

VII. Conclusions

This paper studies joint offloading and compression decisions in a 3-tier user-edge-cloud system considering limited communication and computation capacity of the devices. We proposed a Mixed Integer Nonlinear Program (MINLP) to solve the proposed problem. We decompose the proposed MINLP into two sub-problems: one that solves the offloading sub-problem and another that solves the the compression sub-problem with one set of continues decision variables. The MINLP is NP-hard. Therefore, we propose a heuristic algorithm, namely PCODE, to solve the problem. We compare our proposed PCODE algorithm against several baselines in terms of objective value and the final decisions of each algorithm regarding both the compression and offloading problems using extensive simulation. Our results show that PCODE can match the solution found by the optimization solver by an 0.958-approximation on average. In future works, we will focus on a real-world implementation of this work.

Acknowledgements

This material is based upon work supported by the National Science Foundation under grant no. CSR-1948387. This work was also partially funded by Cisco Systems Inc. under the research grant number 1215519250. We thank both our sponsors for their generous support.

References

[1] H. Xu, W. Yu, D. W. Griffith, and N. Golmie, "A survey on industrial internet of things: A cyber-physical systems perspective," *IEEE Access*, vol. 6, pp. 78238–78259, 2018.

[2] X. Xu, M. ran Han, S. M. Nagarajan, and P. Anandhan, "Industrial internet of things for smart manufacturing applications using hierarchical trustful resource assignment," *Comput. Commun.*, vol. 160, 2020.

[3] 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.

[4] N. Abbas, Y. Zhang, A. Taherkordi, and T. Skeie, "Mobile edge computing: A survey," *IEEE Internet of Things Journal*, vol. 5, no. 1, pp. 450–465, 2017.

[5] Z. Zhou, X. Chen, E. Li, L. Zeng, K. Luo, and J. Zhang, "Edge intelligence: Paving the last mile of artificial intelligence with edge computing," *Proceedings of the IEEE*, vol. 107, pp. 1738–1762, 2019.

[6] W. Shi, J. Cao, Q. Zhang, Y. Li, and L. Xu, "Edge computing: Vision and challenges," *IEEE Internet of Things Journal*, vol. 3, 2016.

[7] D. Xu, T. Li, Y. Li, X. Su, S. Tarkoma, T. Jiang, J. Crowcroft, and P. Hui, "Edge intelligence: Architectures, challenges, and applications," *arXiv: Networking and Internet Architecture*, 2020.

[8] N. Hudson, M. J. Hossain, M. Hosseinzadeh, H. Khamfroush, M. Rahnamay-Naeini, and N. Ghani, "A framework for edge intelligent smart distribution grids via federated learning," *IEEE ICCCN*, 2021.

[9] T. He, H. Khamfroush, S. Wang, T. La Porta, and S. Stein, "It's hard to share: joint service placement and request scheduling in edge clouds with sharable and non-sharable resources," in *IEEE 38th ICDCS*, 2018.

[10] V. Farhadi, F. Mehmeti, T. He, T. La Porta, H. Khamfroush, S. Wang, and K. S. Chan, "Service placement and request scheduling for data-intensive applications in edge clouds," in *IEEE INFOCOM*, 2019.

[11] N. Hudson, H. Khamfroush, and D. E. Lucani, "QoS-aware placement of deep learning services on the edge with multiple service implementations," in *2021 IEEE ICCCN*, pp. 1–8, 2021.

[12] S. Hu and G. Li, "Dynamic request scheduling optimization in mobile edge computing for iot applications," *IEEE Internet of Things Journal*, vol. 7, pp. 1426–1437, 2020.

[13] X. Zhao, M. Hosseinzadeh, N. Hudson, H. Khamfroush, and D. E. Lucani, "Improving accuracy-latency trade-off of edge-cloud computation offloading for deep learning services," in *IEEE Globecom Workshop on Edge Learning over 5G Networks and Beyond*, 2020.

[14] M. Hosseinzadeh, A. Wachal, H. Khamfroush, and D. E. Lucani, "Optimal accuracy-time trade-off for deep learning services in edge computing systems," in *IEEE International Conference on Communications*, 2021.

[15] V. Farhadi, F. Mehmeti, T. He, T. F. L. Porta, H. Khamfroush, S. Wang, K. S. Chan, and K. Poularakis, "Service placement and request scheduling for data-intensive applications in edge clouds," *IEEE/ACM Transactions on Networking*, pp. 1–14, 2021.

[16] T. Ma, M. Hempel, D. Peng, and H. Sharif, "A survey of energy-efficient compression and communication techniques for multimedia in resource constrained systems," *IEEE Communications Surveys & Tutorials*, vol. 15, no. 3, pp. 963–972, 2012.

[17] X. Zou, X. Xu, C. Qing, and X. Xing, "High speed deep networks based on discrete cosine transformation," in *IEEE ICIP*, 2014.

[18] D. Fu and G. Guimaraes, "Using compression to speed up image classification in artificial neural networks," 2016. Technical report.

[19] D. Xu, Q. Li, and H. Zhu, "Energy-saving computation offloading by joint data compression and resource allocation for mobile-edge computing," *IEEE Communications Letters*, vol. 23, pp. 704–707, 2019.

[20] A. J. Hussain, A. Al-Fayadh, and N. Radi, "Image compression techniques: A survey in lossless and lossy algorithms," *Neurocomputing*, vol. 300, pp. 44–69, 2018.

[21] P. Tseng, "Convergence of a block coordinate descent method for nondifferentiable minimization," *Journal of optimization theory and applications*, vol. 109, no. 3, pp. 475–494, 2001.

[22] W. Feng, H. Liu, Y. Yao, D. Cao, and M. Zhao, "Latency-aware offloading for mobile edge computing networks," *IEEE Communications Letters*, vol. 25, pp. 2673–2677, 2021.

[23] S. Huang, C. Yang, S. Yin, Z. Zhang, and Y. Chu, "Latency-aware task peer offloading on overloaded server in multi-access edge computing system interconnected by metro optical networks," *Journal of Lightwave Technology*, vol. 38, pp. 5949–5961, 2020.

[24] I. A. Elgendy, W.-Z. Zhang, Y. Zeng, H. He, Y.-C. Tian, and Y. Yang, "Efficient and secure multi-user multi-task computation offloading for mobile-edge computing in mobile iot networks," *IEEE Transactions on Network and Service Management*, vol. 17, pp. 2410–2422, 2020.

[25] R. Yuan, B. Yang, Y. Liu, and L. Huang, "Modified gompertz sigmoidal model removing fine-ending of grain-size distribution," *Open Geosciences*, vol. 11, no. 1, pp. 29–36, 2019.

[26] B. Gompertz, "XXIV. On the nature of the function expressive of the law of human mortality, and on a new mode of determining the value of life contingencies. in a letter to francis baily, esq. frs &c," *Philosophical transactions of the Royal Society of London*, no. 115, pp. 513–583, 1825.

[27] S. Burer and A. N. Letchford, "Non-convex mixed-integer nonlinear programming: A survey," *Surveys in Operations Research and Management Science*, vol. 17, no. 2, pp. 97–106, 2012.

[28] W. E. Hart, C. D. Laird, J.-P. Watson, D. L. Woodruff, G. A. Hackebeil, B. L. Nicholson, J. D. Siirola, *et al.*, *Pyomo-optimization modeling in python*, vol. 67. Springer, 2017.

Authorized licensed use limited to: UNIVERSITY OF KENTUCKY. Downloaded on March 01,2022 at 16:24:36 UTC from IEEE Xplore. Restrictions apply.