QoS-Aware Placement of Deep Learning Services on the Edge with Multiple Service Implementations

Nathaniel Hudson*, Hana Khamfroush*, and Daniel E. Lucani†
*University of Kentucky, Lexington, KY, USA. Email: nathaniel.hudson@uky.edu, khamfroush@cs.uky.edu
†Aarhus University, Aarhus, Denmark. Email: daniel.lucani@eng.au.dk

***Abstract*—Mobile edge computing pushes computationally-intensive services closer to the user to provide reduced delay due to physical proximity. This has led many to consider deploying deep learning models on the edge — commonly known as edge intelligence (EI). EI services can have many model implementations that provide different QoS. For instance, one model can perform inference faster than another (thus reducing latency) while achieving less accuracy when evaluated. In this paper, we study joint service placement and model scheduling of EI services with the goal to maximize Quality-of-Servcice (QoS) for end users where EI services have multiple implementations to serve user requests, each with varying costs and QoS benefits. We cast the problem as an integer linear program and prove that it is NP-*hard*. We then prove the objective is equivalent to maximizing a monotone increasing, submodular set function and thus can be solved greedily while maintaining a $(1-1/e)$-approximation guarantee. We then propose two greedy algorithms: one that theoretically guarantees this approximation and another that empirically matches its performance with greater efficiency. Finally, we thoroughly evaluate the proposed algorithm for making placement and scheduling decisions in both synthetic and real-world scenarios against the optimal solution and some baselines. In the real-world case, we consider real machine learning models using the ImageNet 2012 data-set for requests. Our numerical experiments empirically show that our more efficient greedy algorithm is able to approximate the optimal solution with a 0.904 approximation on average, while the next closest baseline achieves a 0.607 approximation on average.**

***Index Terms*—Service Placement, Edge Intelligence, Edge Computing, Deep Learning, Optimization, Quality-of-Service**

I. Introduction

The growth of the Internet has given birth to the advent of the Internet-of-Things (IoT). This ecosystem consists of countless different devices, or things (e.g., sensors, home appliances), that can seamlessly communicate with one another. More importantly, these devices also serve to generate/collect data. In order to acquire meaningful information from these data, they must first be processed. The scale of the IoT poses a problem for processing these data in a timely fashion through a conventional, centralized approach (e.g., cloud computing). As such, a promising framework to approach this problem has been that of *mobile edge computing* (MEC) [1]–[4].

The MEC framework considers the deployment of *edge clouds* (or *edge servers*) that provide communication, compute, and storage resources closer to the end user devices to ameliorate latency incurred from physical distance. This physical proximity allows for more immediate and timely data processing for nearby devices in IoT. However, MEC is not without challenges. The hardware resources available at an individual edge cloud pales in comparison to that available at the far away central cloud server. As such, decisions related to how these resources are spent must be optimized. Regardless, MEC remains a promising framework for performing timely data processing for IoT devices, such as smart sensors.

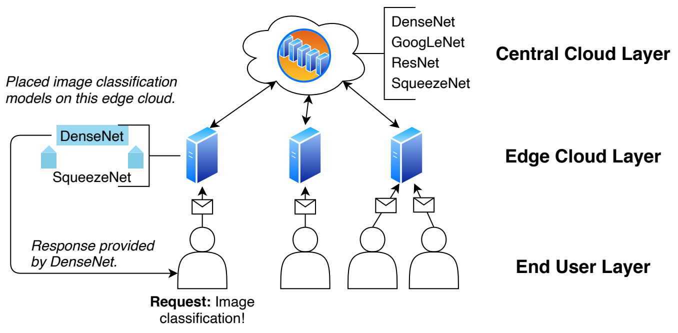
DenseNet
GoogLeNet
ResNet
SqueezeNet
Central Cloud Layer
Placed image classification models on this edge cloud.
DenseNet
SqueezeNet
Edge Cloud Layer
Response provided by DenseNet.
End User Layer
Request: Image classification!

Fig. 1: 3-tier MEC architecture where an edge cloud has two image classification models to serve requests for that service.

The popularity of *machine learning* (ML) for performing the task of data processing has skyrocketed in recent years. ML has been shown to be capable of achieving remarkable accuracy for complex tasks (e.g., image classification, video classification, speech-to-text). Due to the flexibility and performance of ML technologies, deploying such models on the edge to process IoT data is promising. Thus, the notion of *edge intelligence* (EI) has gained prominence. A notable feature of EI services, such as image classification, is that several different EI architectures (i.e., deep neural networks) can be implemented to perform inference for some input for a service. These different architectures can have varying trade-offs in terms of the time they take to perform inference, the size of input data they require, and how accurate they are in practice when performing inference. Given these observations, in this work, we adapt the well-studied problem of *service placement* in MEC to consider EI services with different model implementations. Most notably, this problem aims to maximize the Quality-of-Service (QoS) provided by the edge when services are allowed to have several different model implementations to serve user requests.

To the best of our knowledge, this is the first EI service placement work that considers each service to have multiple implementations. We summarize our contributions as follows:

- Introduce the *Placement of Intelligent Edge Services* (PIES) problem for optimal placement and scheduling of EI services with multiple implementations to maximize QoS provided by the edge.
- Prove that the PIES problem is NP-*hard* and that its objective maximizes a monotone increasing, submodular set function under matroid constraints.
- Propose two greedy algorithms: one with a $(1 - 1/e)$-approximation guarantee and another more efficient greedy algorithm that empirically matches this approximation algorithm with much greater efficiency.
- Our empirical results show that both greedy algorithms outperform their minimal approximation guarantee and each achieve an approximation of roughly 0.9 on average.

II. Related Works

Service Placement. The problem of deciding which services to place on edge clouds in MEC is commonly known as the *service placement* problem. Many works study the problem with the goal to optimize resource utilization, energy consumption [5], [6], and Quality-of-Service (QoS) [7], [8]. Some works consider a static placement decision where the placement decision is made in a single shot [9], [10]. Other works focus on dynamic placement where placement decisions are made over some timespan [11], [12]. The case where edge clouds can share their resources with one another to collaboratively serve user requests has been studied by He et al. in [13]. The existence of mobile end users has encouraged research studying placement with service migration, where a service processing a request migrates between edge clouds [14]. For a recent and comprehensive survey on service placement, please refer to [15]. *Quality-of-Service* (QoS) is very domain specific. Gao et al. in [7] study maximizing QoS, defined as a function of latency, through a joint decision problem for both service placement and network selection where users can be served by more than one edge cloud. Skarlat et al. define FSPP, a QoS-aware service placement problem, as an ILP where QoS is defined as a function of application deadlines — using the Gurobi solver to provide the optimal solution [16]. Yousefpour et al. in [17] propose a dynamic fog computing framework, called FogPlan, for dynamic placement and release of services on fog nodes in IoT infrastructure. The authors consider QoS-aware service placement where QoS is considered exclusively w.r.t. delay latency and user's delay tolerance. Wang et al. in [5] study a similar problem wherein they focus on the placement of virtual machines for software-defined data centers to maximize energy efficiency and QoS. An application placement policy using a fuzzy logic-based approach is proposed by Mahmud et al. in [8] that maximizes QoE. In this work, the authors consider QoE under a fuzzy framework comprised of sets for access rate, required resources, and processing times. Farhadi et al. in [18] study service placement and request scheduling on MEC systems for data-intensive applications. They pose the problem of service placement as a set optimization problem and provide an algorithm that demonstrates an approximation bound on optimal solutions.

Edge Intelligence. The advent of pushing machine learning services to the edge led to the established field of *edge intelligence* (EI) [19]. Due to the limited resource capacities of MEC environments, a central focus of EI is the design of models that are less costly in terms of resources to run [20]. One proposed solution is to simply prune the elements comprising the deep neural network for the EI service (e.g., remove the number of neurons/units or entire layers) [21]–[23]. Another proposed idea is to consider a EI model's architecture being *split* across different tiers of the MEC architecture (e.g., one half is run on the edge and the other on the central cloud) [24]. Other works have studied optimizing the QoS provided by deep learning EI models on the edge. Zhao et al. in [25] study the trade-off between accuracy and latency for offloading decisions regarding deep learning services for provided optimal QoS on the edge through compression techniques. Hosseinzadeh et al. in [26] study the related problem of offloading and request scheduling in edge systems to maximize QoS for deep learning models under the assumption that placement of these models has been done a priori. There are few works that have formally studied the service placement problem specifically for EI models. Recently in 2021, Zehong, Bi, and Zhang study the problem of optimally placing EI services with the objective of optimizing energy consumption and completion time [27].

Our work departs from the literature in that we focus on EI service placement where each service can have multiple implementations. To the best of our knowledge from surveying the literature, this is the first work to do so. A similar work by Hung et al. in [28] considers multiple implementations of services. However, the focus of that work is on optimal query scheduling of video processing tasks rather than placement of services on edge clouds.

III. System Model

A. System Architecture Definition

We consider a 3-tier MEC architecture consisting of a central cloud, edge clouds, and end users (see Figure 1). The central cloud hosts all model implementations for each service in the environment. However, the objective of this work is to maximize expected QoS provided by the edge clouds and thus we do not closely consider the central cloud. For this work, we focus on three aspects of this environment: edge clouds, end user requests, and available EI service models. For simplicity, we consider requests processed by the cloud to be dropped.

Edge Clouds. We denote the set of edge clouds $\mathcal{E} = \{1, \cdots, E\}$ where E is the number of edge clouds. We consider edge clouds to be deployed computational devices that are associated with a wireless access point to connect to users. Each edge cloud is equipped with hardware resources to process the requests provided by end users. Specifically, we consider each edge cloud $e \in \mathcal{E}$ to have resource capacities K_e, W_e, and R_e for communication, computation, and storage capacities, respectively. For simplicity, we do not consider the possibility of edge clouds collaborating to serve user requests.

Thus, either a user's request is served by the user's covering edge cloud or is offloaded to the central cloud (dropped).

User Requests. We denote the set of user requests as $\mathcal{U} = \{1, \cdots, U\}$ where U is the number of user requests. For simplicity, we consider each user to make a single service request. For users that request multiple services, we represent this as separate user requests altogether. Each user is covered by some edge cloud, which will process their service request. We denote the edge cloud covering user u by e_u. Additionally, we denote the set of users an edge cloud e covers by $\mathcal{U}_e = \{u \in \mathcal{U} : e_u \equiv e\}$. The service some user u requests is denoted by s_u. When submitting a request, users also provide thresholds for accuracy and delay inform the MEC system for how to make decisions w.r.t. QoS. The accuracy threshold provided by user u is denoted by $\alpha_u \in [0, 1]$; the delay threshold provided by user u is denoted by $\delta_u \in [0, \delta_{\max}]$, where $\delta_{\max}$ represents the maximum possible delay. These thresholds are used to prioritize the needs of end users. For instance, some applications that use deep learning are more sensitive to inaccurate answers and others are more time-sensitive. For instance, a self-driving vehicle that uses object detection to detect nearby pedestrians would be more accuracy-sensitive and delay-sensitive than a game on a smartphone.

Service Models. We consider a set of services that are available for users to request, denoted by $\mathcal{S} = \{1, \cdots, S\}$ where S is the number of available services. We assume that there is at least 1 implementation for each service $s \in \mathcal{S}$. However, we also allow for EI services to be implemented by several different machine learning architectures. For brevity, we refer to a single service implementation as a "service model" for short. The set of implemented service models for service s is denoted by $\mathcal{M}_s = \{1, \cdots, m_s\}$ where $m_s \geq 1$ is the number of models implemented for service s. For simplicity, we also denote the set of all individual implemented service models by $\mathcal{SM} = \{(s, m) : \forall s \in \mathcal{S}, m \in \mathcal{M}_s\}$. Each service model $(s, m) \in \mathcal{SM}$ is associated with an accuracy metric $A_{sm} \in [0, 1]$. We assume this value is provided by evaluation using some test data-set (as is typical in machine learning). We denote the expected delay for performing service model (s, m) for user u by $D_{sm}(u)$ — this is defined more explicitly in §III-B2. Finally, we denote communication, computation, and storage costs for each service model $(s, m) \in \mathcal{SM}$ by k_{sm}, w_{sm}, and r_{sm}, respectively.

B. Quality-of-Service (QoS) Definition

We consider QoS for EI models to be comprised of two components: provided model accuracy and incurred delay. As mentioned earlier in §III-A, each user request submitted to the system includes thresholds for requested minimum accuracy, α_u, and requested maximum delay, δ_u. As such, we use these threshold values to compute the expected QoS a service model (s, m) can provide to user u. The formal definition is provided below in Eq. (1),

$$Q(u, s, m) \triangleq \begin{cases} \frac{1}{2}[\hat{a}_{sm}(u) + \hat{d}_{sm}(u)] & \text{if } s = s_u \\ 0 & \text{otherwise} \end{cases} \tag{1}$$

where $\hat{a}_{sm}(u)$ and $\hat{d}_{sm}(u)$ represent how much service model (s, m) satisfies user u's accuracy and delay thresholds, respectively. The summation of these two terms is multiplied by $1/2$ because the maximum possible values for both $\hat{a}_{sm}(\cdot)$ and $\hat{d}_{sm}(\cdot)$ is 1.0, thus normalizing the range to $Q(\cdot) \in [0, 1]$.

1) Accuracy Satisfaction: As stated, each user submits a minimum accuracy threshold, $\alpha_u \in [0, 1]$, which indicates the amount of accuracy needed to satisfy them. This accuracy of a service model, A_{sm}, is a metric retrieved from model evaluation (as is standard with machine learning models). We define this as a nonlinear function in Eq. (2) below,

$$\hat{a}_{sm}(u) = \begin{cases} 1 & \text{if } A_{sm} \geq \alpha_u \\ \max(0, 1 - (\alpha_u - A_{sm})) & \text{otherwise} \end{cases} \tag{2}$$

where the first case represents when a user's accuracy request has been met and the second case provides reduced satisfaction based on the difference between user-requested accuracy and the evaluated accuracy for service model (s, m).

2) Delay Satisfaction: Similarly, each user submits a maximum delay threshold, $\delta_u \in [0, \delta_{\max}]$, to indicate the amount of accuracy they are willing to tolerate. If they receive a response for their request within $\delta_{\max}$ time units, then they are satisfied; otherwise, their satisfaction will degrade. The formal definition is provided below in Eq. (3),

$$\hat{d}_{sm}(u) = \begin{cases} 1 & \text{if } D_{sm}(u) \leq \delta_u \\ \max\left(0, 1 - \frac{D_{sm}(u) - \delta_u}{\delta_{\max}}\right) & \text{otherwise} \end{cases} \tag{3}$$

where $D_{sm}(u)$ is the expected delay from processing user u's request using service model (s, m). This is defined below in Eq. (4) as the sum of two terms of the transmission delay, $D^{tran}_{sm}(\cdot)$, and the computation delay, $D^{comp}_{sm}(\cdot)$. See below for the formal definition:

$$D_{sm}(u) = D^{tran}_{sm}(u) + D^{comp}_{sm}(u). \tag{4}$$

The transmission delay is a function of the communication cost of service model (s, m) and the communication capacity of user u's covering edge cloud, e_u. Additionally, the edge cloud's bandwidth is evenly shared across all of the users it covers — see Eq. (5),

$$D^{tran}_{sm}(u) = \frac{k_{sm}}{K_{e_u}/|\mathcal{U}_{e_u}|} = \frac{k_{sm}\,|\mathcal{U}_{e_u}|}{K_{e_u}} \tag{5}$$

where k_{sm} is the communication cost for service model (s, m). Similarly, computation delay is a function of the computation cost of service model (s, m) and user u's covering edge cloud, e_u, and its computation resources — see Eq. (6),

$$D^{comp}_{sm}(u) = \frac{w_{sm}}{W_{e_u}/|\mathcal{U}_{e_u}|} = \frac{w_{sm}\,|\mathcal{U}_{e_u}|}{W_{e_u}} \tag{6}$$

where w_{sm} is the computation cost for service model (s, m). We assume that the an edge cloud's computation capacity is evenly shared across its covered users.

IV. Problem Definition

Given the system model, we now define the *Placement for Intelligent Edge Services* (PIES) problem. PIES performs service placement for EI models with multiple implementations with the goal of maximizing QoS, defined in Eq. (1). On top of making placement decisions, the PIES problem also decides *which* placed service model will serve a user's request if there are more implementations for a requested service available. For instance, given each user u makes a request for service s_u, if u's covering edge cloud e_u has had more than 1 model of service s_u placed on it, then the PIES problem will also select a placed model to process u's request.

A. PIES Formulation

The PIES problem is defined as an *integer linear program* (ILP) and consists of two types of decisions: *(i)* model placement and *(ii)* model scheduling. For the former, we consider a binary decision variable $\mathbf{x} = (x_e^{sm})_{\forall e \in \mathcal{E}, s \in \mathcal{S}, m \in \mathcal{M}_s} = 1$ if service model (s, m) is placed on edge cloud e, 0 otherwise. For the latter, we consider another binary decision variable $\mathbf{y} = (y_u^m)_{\forall u \in \mathcal{U}, m \in \mathcal{M}_{s_u}} = 1$ if user u's service request is served by its covering edge cloud e_u with service model (s_u, m), 0 otherwise. We formally define the PIES problem below:

$$\max \sum_{u \in \mathcal{U}} \sum_{m \in \mathcal{M}_{s_u}} y_u^m Q(u, s_u, m) \tag{7}$$

$$\text{s.t.} \sum_{m \in \mathcal{M}_{s_u}} y_u^m \leq 1 \qquad \forall u \in \mathcal{U} \tag{7a}$$

$$\sum_{s \in \mathcal{S}} \sum_{m \in \mathcal{M}_s} x_e^{sm} r_{sm} \leq R_e \qquad \forall e \in \mathcal{E} \tag{7b}$$

$$y_u^m \leq x_{e_u}^{s_u m} \qquad \forall u \in \mathcal{U}, m \in \mathcal{M}_{s_u} \tag{7c}$$

$$x_e^{sm} \in \{0, 1\} \qquad \forall e \in \mathcal{E}, (s, m) \in \mathcal{SM} \tag{7d}$$

$$y_u^m \in \{0, 1\} \qquad \forall u \in \mathcal{U}, m \in \mathcal{M}_{s_u} \tag{7e}$$

The objective function is defined in Eq. (7) and maximizes the expected QoS provided by the edge clouds to all users. Constraint (7a) ensures that no more than 1 model is used to process a user's request. Constraint (7b) guarantees that all edge clouds' storage capacities are not exceeded by the summation of the storage costs of their placed service models. Constraint (7c) ensures that users can only be served if their covering edge cloud has placed at least 1 implementation of their requested service. Finally, constraints (7d) and (7e) defines $\mathbf{x}$ and $\mathbf{y}$ as binary decision variables.

B. PIES Problem Complexity & Properties

Next, we provide proofs related to the hardness of the PIES problem, as well as theoretical properties that can be used to provide approximation guarantees for algorithm design.

Theorem 1. *The service model placement sub-problem of the PIES problem is NP-hard.*

Proof. We prove Theorem 1 using a reduction from the 0/1 Knapsack problem, which is one of Karp's 21 classical problems proven to be NP-*complete* [29].To review, the 0/1 Knapsack problem considers $1, \cdots, n$ items with each item having a weight cost c_i and a value v_i, as well as a maximum capacity C for the knapsack. The problem's objective is to maximize $\sum_{i=1}^n v_i x_i$ subject to $\sum_{i=1}^n w_i x_i \leq C$ and $x_i \in \{0, 1\}$. Here, $x_i = 1$ if and only if the i^{th} item is selected to be placed in the knapsack.

We can reduce the 0/1 Knapsack problem to the PIES problem as follows. Suppose that there are $|\mathcal{S}| = n$ available services in the MEC system and only 1 model per service, i.e., $|\mathcal{M}_s| = 1$ $(\forall s \in \mathcal{S})$. Each service i has v_i users requesting it — meaning there are $|\mathcal{U}| = \sum_{i=1}^n v_i$ users in total. Suppose there is $|\mathcal{E}| = 1$ edge cloud in the MEC system with storage capacity $R_1 = C$, communication capacity $K_1 = \infty$, and computation capacity $W_1 = \infty$. Let all users be covered by this 1 edge cloud, such that $e_u = 1$ $(\forall u \in \mathcal{U})$. Let the storage costs associated with each service model be $r_{s1} = c_s$ $(1 \leq s \leq n)$ and the communication/computation costs $k_{s1} = 1, w_{s1} = 1$ $(1 \leq s \leq n)$. We assume that the QoS requirements of all users are relaxed, meaning that $\alpha_u = 0.0, \delta_u = \delta_{\max}$ $(\forall u \in \mathcal{U})$. Then we claim that the 0/1 Knapsack problem is feasible if and only if we can maximize expected QoS across all users, i.e., the optimal decision variable of the constructed PIES instance equals $x_1^{i1} \equiv x_i$. First, given the optimal solution to the 0/1 Knapsack, placing the services corresponding to the decisions in x_i on the single edge cloud in our constructed instance gives the optimal solution to the PIES problem that maximizes QoS by maximizing the number of user requests served on the edge. Moreover, given an optimal solution to the PIES problem that maximizes QoS across all users, placing the corresponding items in the knapsack gives an optimal solution to the 0/1 Knapsack problem. Given the decision problem of the 0/1 Knapsack problem is NP-*complete*, it follows that the PIES problem is NP-*hard*. This concludes the proof. $\square$

Theorem 2. *Given a service model placement,* $\mathbf{x}$*, the optimal solution to the PIES model scheduling sub-problem is given by a greedy algorithm.*

To prove Theorem 2, we consider an auxiliary, undirected multigraph and discuss its construction. Consider a multigraph $\mathcal{G} = (V, L, f)$ where V is the set of nodes, $L \subseteq |V| \times |V|$ the set of links/edges (hereafter we will refer to as *links*), and $f : L \rightarrow \{\{u, v\} : u, v \in V\}$ is the link identifier function. Consider the node set V to be composed of 3 sets of nodes: a set containing a single *root node* V_R, a set of *user nodes* V_U, and a set of *service nodes* V_S. Thus, we say $V = V_R \cup V_U \cup V_S$. We define V_U as the set of users with at least 1 model placed on their edge cloud for their requested service, i.e., $V_U = \{u \in \mathcal{U} : \sum_{m \in M_{s_u}} x_{e_u}^{s_u m} > 0\}$. We define V_S as the set of requested services for each user $u \in V_U$ that have had at least 1 service model placed on their covering edge cloud, i.e., $V_S = \{s_u : \forall u \in V_U\}$. We note that we allow for duplicate services such that each user $u \in V_U$ has its own service node. The set of links, L, contains links between node sets V_R, V_U and V_U, V_S. First, there is 1 edge between the root node, $\phi \in V_R$, and each of the user nodes $u \in V_U$, i.e., $\{(\phi, u) : \forall u \in V_U\} \subseteq L$. Then, a user node $u \in V_U$ can have multiple links to its

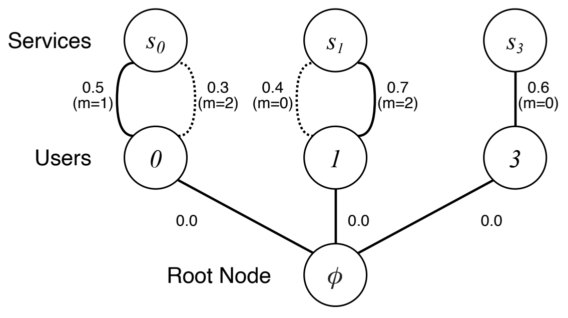

Fig. 2: Example auxiliary multigraph representing PIES scheduling sub-problem. Links between user/service nodes correspond to an model implementation for a user's requested service placed on the user's covering edge. Float values correspond with QoS weights. Solid links compose the maximum spanning tree for optimal model scheduling.

corresponding service node $s_u \in V_S$. The number of links between $u \in V_U$ and $s_u \in V_S$ is equal to $\sum_{m \in \mathcal{M}_{s_u}} x_{e_u}^{s_u m}$, which is the number of model implementations for s_u placed on u's covering edge cloud, e_u. Finally, each link is weighted. Links between the root node $\phi \in V_R$ and user nodes $u \in V_U$ are weighted by 0. Links between each user nodes $u \in V_U$ and its requested service nodes s_u are weighted by the expected $Q(u, s, m)$ where m corresponds with the model type of the respective link between u and s_u. A simple example of a multigraph constructed by this method is provided in Fig. 2.

Proof. Given a service placement decision, $\mathbf{x}$, we can prove Theorem 2 by constructing an auxiliary multigraph that represents the problem space for the PIES model scheduling sub-problem as described above and shown in Fig. 2. With this graph constructed, it is easy to verify that the optimal solution to the model scheduling sub-problem is equivalent to finding a maximum spanning tree (MST) — which can be found by applying a greedy algorithm for minimum spanning trees (e.g., Kruskal's algorithm [30]) and negating all of the edge weights. This concludes the proof. □

Proving Submodularity. In order to provide an approximation guarantee for the NP-*hard* PIES placement sub-problem, we prove the PIES service placement sub-problem is maximizing a monotone submodular set function, we rewrite our problem as a set optimization. Let $P(\mathbf{x}) \triangleq \{(e, (s, m)) \in \mathcal{E} \times \mathcal{SM} : x_e^{sm} = 1\}$ denote the set of service model placements according to decision variable $\mathbf{x}$, where $(e, (s, m))$ means service model (s, m) is placed on edge cloud e. Next, let $\sigma(P(\mathbf{x}))$ denote the optimal objective value of PIES for a given $\mathbf{x}$. By writing $P(\mathbf{x})$ as P, we can rewrite the PIES placement sub-problem as:

$$\max \ \sigma(P) \tag{8}$$

$$\text{s.t.} \sum_{(e,(s,m)) \in P \cap P_e} r_{sm} \leq R_e \qquad \forall e \in \mathcal{E} \tag{8a}$$

$$P \subseteq \mathcal{E} \times \mathcal{SM} \tag{8b}$$

where $P_e \triangleq \{e\} \times \{(s, m) \in \mathcal{SM} : r_{sm} \leq R_e\}$ is the set of all possible *single* service model placements at edge cloud e. From here, we observe the following:

- *Matroid constraint:* Let $\mathcal{I}$ be the collection of all P satisfying constraints (8a), (8b). It is then easy to verify that $\mathbb{M} = (\mathcal{E} \times \mathcal{SM}, \mathcal{I})$ is a matroid. This is known as the *partition matroid* as $\{P_e\}_{e \in \mathcal{E}}$ is a partition of the ground set $(\mathcal{E} \times \mathcal{SM})$.
- *Monotone submodular objective function:* We show that the objective function (8) has the following properties.

Theorem 3. *Function $\sigma(P)$ is monotone increasing and submodular for any feasible P.*

Proof. First, adding an element $(e, (s, m))$ to P corresponds to adding new possible users to serve or new links in the auxiliary multigraph $\mathcal{G}$ (see Fig. 2). Under either outcome, the objective value, Eq. (7), will either increase or remain unchanged. Thus it is sufficient to say that $\sigma(P)$ is monotone increasing.

The PIES service model placement sub-problem is submodular iff for every $A, B \subseteq \mathcal{E} \times \mathcal{SM}$ where $A \subseteq B$ and some $p \notin B$, the following condition holds: $\sigma(A \cup \{p\}) - \sigma(A) \geq \sigma(B \cup \{p\}) - \sigma(B)$. We can define the objective value under optimal scheduling given a set of placements P as

$$\sigma(P) \triangleq \sum_{u \in \mathcal{U}} \sigma_u(P) \tag{9}$$

where $\sigma_u(P)$ is the optimal QoS provided to user u with service model placements P. We can define $\sigma_u(P)$ as follows:

$$\sigma_u(P) \triangleq \max_{(e,(s,m)) \in P} \{Q(u, s, m) : e = e_u \wedge s = s_u\} \cup \{0\}. \tag{10}$$

Next, we claim that for every user $u \in \mathcal{U}$ the following holds: $\sigma_u(A \cup \{p\}) - \sigma_u(A) \geq \sigma_u(B \cup \{p\}) - \sigma_u(B)$. We can verify this by elaborating on what these expressions represent and by exploiting their definitions. First, by definition, $\sigma_u(A \cup \{p\}) - \sigma_u(A)$ represents the increase $\{p\}$ provides to the objective. It should be noted that $\{p\}$ can only provide increase if its provided QoS for u is greater than what A could already provide. Additionally, we note $\sigma_u(A \cup \{p\}) \in \{\sigma_u(A), \sigma_u(\{p\})\}$. We claim this because $\sigma_u(A \cup \{p\}) \equiv \sigma_u(A)$ if $\sigma_u(A) \geq \sigma_u(\{p\})$, meaning an already placed model in A is scheduled to serve u's request, or $\sigma_u(A \cup \{p\}) \equiv \sigma_u(\{p\})$ if $\sigma_u(A) < \sigma_u(\{p\})$, meaning the new service model placed by the new placement $\{p\}$ is scheduled to serve u's request. These observations similarly hold for B as well. Now, we prove the following inequality $\sigma_u(A \cup \{p\}) - \sigma_u(A) \geq \sigma_u(B \cup \{p\}) - \sigma_u(B)$ directly by its possible cases:[1]

- CASE 1. $\sigma_u(A \cup \{p\}) \equiv \sigma_u(A)$ and $\sigma_u(B \cup \{p\}) \equiv \sigma_u(B)$. It is easy to verify that $\sigma_u(A \cup \{p\}) - \sigma_u(A) \geq \sigma_u(B \cup \{p\}) - \sigma_u(B)$ becomes $0 \geq 0$, which holds.
- CASE 2. $\sigma_u(A \cup \{p\}) > \sigma_u(A)$ and $\sigma_u(B \cup \{p\}) \equiv \sigma_u(B)$. It is easy to verify that the lefthand side of the

[1]Note the case that $\sigma_u(A \cup \{p\}) \equiv \sigma_u(A)$ and $\sigma_u(B \cup \{p\}) > \sigma_u(B)$ can never occur. This is due to the fact that $A \subseteq B$ and thus if $\{p\}$ provides greater QoS for user u than B, then it follows that is also true for A.

Algorithm 1: Optimal Model Scheduling (OMS)

```
Input  : Service placement ($\mathbf{x}$), Input parameters of (7)
Output: Model scheduling ($\mathbf{y}$)
1  Initialize $\mathbf{y} \leftarrow (y_u^m = 0)_{\forall u \in \mathcal{U}, m \in \mathcal{M}}$;
2  foreach user $u \in \mathcal{U}$ do
3      if $\sum_{m \in \mathcal{M}_{s_u}} x_{e_u}^{s_u m} > 0$ then
4          $m^* \leftarrow \arg\max_{m \in \mathcal{M}_{s_u}} \{Q(u, s_u, m), \forall u \in \mathcal{U}_e : x_{e_u}^{s_u m} = 1\}$;
5          $y_u^{m^*} \leftarrow 1$;
6  return $\mathbf{y}$;
```

original inequality becomes a value > 0 and the righthand side becomes 0, thus the inequality holds.

- CASE 3. $\sigma_u(A \cup \{p\}) > \sigma_u(A)$ and $\sigma_u(B \cup \{p\}) > \sigma_u(B)$. Since $A \subseteq B$, any service model placed under A is also placed under B. Intuitively, any increase to QoS for user u provided by $A \cup \{p\}$ can be matched by $B \cup \{p\}$ because B has every service model to serve u's request that A has. Additionally, B could also have service models that provide greater QoS for u than A due to it having more service models placed. Thus, it must follow that the original inequality holds because A's increase in QoS for u is always at least as large as B's increase.

Thus, $\sigma_u(\cdot)$ is submodular for every $u \in \mathcal{U}$. Since any function that is a summation of submodular functions is also submodular [32], it then follows that $\sigma(\cdot)$ is submodular. This concludes the proof. □

V. EFFICIENT ALGORITHM DESIGN

A. Scheduling Sub-Problem

In §IV-B, we proved the PIES model scheduling sub-problem can be optimally solved with a greedy solution (see Theorem 2). As such, we introduce a simple greedy algorithm, *Optimal Model Scheduling* (OMS). The pseudocode is provided in Algorithm 1. OMS works in a straightforward fashion. Given a service placement decision and the PIES input parameters, OMS iterates through each user $u \in \mathcal{U}$ and if there is at least one model of their requested service, s_u, on their covering edge cloud, e_u, then OMS selects the model that provides the greatest QoS to user u. The runtime for OMS is $O(|\mathcal{U}||\mathcal{M}_s^{\max}|)$ where $|\mathcal{M}_s^{\max}| = \max_{s \in \mathcal{S}}(|M_s|)$.

B. Placement Sub-Problem

Here, we introduce two algorithms that can be used to solve the service model placement sub-problem for PIES. The first is an approximation algorithm that exploits the theoretical properties of the PIES objective (discussed in §IV-B) to achieve an approximately optimal solution. The latter algorithm mimics some of the logic of this algorithm while reducing computational heft.

1) Approximation Algorithm: Because the PIES problem aims to maximize a monotone increasing, submodular set function under matroid constraints (see Theorem 3), a standard greedy algorithm can provide a $(1 - 1/e)$-approximation of

Algorithm 2: Approx. Greedy Placement (AGP)

```
Input  : Input parameters of (7)
Output: Service placement ($\mathbf{x}$)
1  Initialize $\mathbf{x} \leftarrow (x_e^{sm} = 0)_{\forall e \in \mathcal{E}, s \in \mathcal{S}, m \in \mathcal{M}}$;
2  $P \leftarrow \{\}$ ;          // Placement decisions $\forall e \in \mathcal{E}$.
3  foreach $e \in \mathcal{E}$ do
4      $\hat{P} \leftarrow \{\}$ ;    // Placement decisions for this $e$.
5      $\hat{R} \leftarrow R_e$;
6      repeat
7          $L \leftarrow \{(s,m) \in \mathcal{SM} \setminus \hat{P} : r_{sm} \leq \hat{R}\}$;
8          $s^*, m^* \leftarrow \arg\max_{(s,m) \in L} \sigma(P \cup \{(e,(s,m))\})$;
9          $\hat{P} \leftarrow (s^*, m^*)$;
10         $P \leftarrow P \cup \{(e,(s^*,m^*))\}$;
11         $\hat{R} \leftarrow \hat{R} - r_{s^* m^*}$;
12     until $|L \setminus \hat{P}| = 0$;
13 foreach $(e,(s,m)) \in P$ do
14     $x_e^{sm} \leftarrow 1$;
15 return $\mathbf{x}$;
```

the optimal solution [33]. As such, we introduce *Approximate Greedy Placement* (AGP) which serves as an approximation algorithm for the PIES placement sub-problem. Its pseudocode is provided in Algorithm 2. AGP iterates through each edge cloud $e \in \mathcal{E}$ and, in each iteration, it finds the set of service models $(s,m) \in \mathcal{SM}$ that can be placed on e without violating the storage capacity constraint. It then computes the objective value using optimal model scheduling via Eq. (9) to find the immediate best choice. Once there are no more legitimate choices to choose from, it moves on to the next edge cloud. Once iteration through edge clouds is finished, it converts the placement decisions (represented as a set) into the standard format for the decision variable. However, AGP's runtime is not desirable due to its need to compute optimal scheduling for each possible option at each iteration.

2) Efficient Algorithm: Due to the heavy runtime complexity of AGP, there is a need for a more efficient algorithm that can relatively match AGP's performance w.r.t. approximating the optimal solution without the large computational cost. Thus, we introduce the *Efficient Greedy Placement* (EGP) algorithm. EGP iteratively places models by keeping a record of anticipated benefit of placing any given service model on an edge cloud. It does this without computing optimal scheduling, thus reducing its computational cost. EGP's pseudocode is provided in Algorithm 3. In line 1, the placement decision variable is initialized. Then, on line 2, we begin to iterate through each edge cloud to decide which service models should be placed on the current edge cloud. Line 3 initializes an empty hash-map and lines 4-6 compute the total QoS each service model relevant for the current edge cloud can provide towards the objective function. This data structure will be updated as decisions are made. Lines 7-9 initialize some supporting variables for EGP's logic. $\mathcal{A}$ records service models that have been considered for placement at some point for the current edge cloud; $\mathcal{B}$ keeps track of the set of users who can be provided maximum QoS; and $\hat{R}$ tracks the remaining storage capacity. Lines 10-20 find the service

Algorithm 3: Efficient Greedy Placement (EGP)

```
Input  : Input parameters of (7)
Output: Service placement $(\mathbf{x})$
Initialize $\mathbf{x} \leftarrow (x_e^{sm} = 0)_{\forall e \in \mathcal{E}, s \in \mathcal{S}, m \in \mathcal{M}}$;
foreach edge $e \in \mathcal{E}$ do
    Initialize hash-map $\mathbf{v}$ with default values of 0.0;
    foreach user $u \in \mathcal{U}_e$ do
        foreach model type $m \in \mathcal{M}_{s_u}$ do
            $\mathbf{v}_{s_u m} \leftarrow \mathbf{v}_{s_u m} + Q(u, s_u, m)$;
    $\mathcal{A} \leftarrow \{\}$ ;              // Considered service models.
    $\mathcal{B} \leftarrow \{\}$ ;                        // Satisfied users.
    $\hat{R} \leftarrow R_e$ ;                      // Remaining storage.
    repeat
        $s^*, m^* \leftarrow \arg\max_{(s,m) \in \text{keys}(\mathbf{v}) \setminus \mathcal{A}} \{\mathbf{v}_{sm}\}$;
        if $r_{s^* m^*} \leq \hat{R}$ then
            $x_e^{s^* m^*} \leftarrow 1$;
            $\hat{R} \leftarrow \hat{R} - r_{s^* m^*}$;
            foreach $m \in \mathcal{M}_{s^*}$ where $(s^*, m) \notin \mathcal{A}$ do
                $\mathbf{v}_{sm} \leftarrow \sum_{u \in \mathcal{U}_e \setminus \mathcal{B}} Q(u, s^*, m) - Q(u, s^*, m^*)$;
        $\mathcal{A} \leftarrow \mathcal{A} \cup \{(s^*, m^*)\}$;
        foreach $u \in \mathcal{U}_e$ do
            if $Q(u, s^*, m^*) = 1$ then $\mathcal{B} \leftarrow \mathcal{B} \cup \{u\}$ ;
    until $(\hat{R} = 0) \vee (|\mathcal{U}_e| = |\mathcal{B}|) \vee (|\mathcal{A}| = |keys(\mathbf{v})|)$;
return $\mathbf{x}$;
```

model (s^*, m^*) that provide the maximum QoS among the service models we have yet to consider from our hash-map. If (s^*, m^*) can be placed without violating the storage constraint, then it will be placed (line 13) and remaining storage will be reduced (line 14). Then, in lines 15-16 we recalculate the benefit of each other model implementation for s^* by computing $\sum_{u \in \mathcal{U}_e \setminus \mathcal{B}} Q(u, s^* m) - Q(u, s^*, m^*)$. Since the newly placed model degrades the benefit from picking other implementations of the same service, we sum the difference between these other models and the newly placed model to reevaluate the benefit of placing them. Lines 10-20 repeat until either there is no more storage space, all of the edge cloud's user's achieve maximum QoS, or we have considered all models relevant for the edge cloud (according to the keys recorded in the hash-map). The runtime complexity of EGP is $O(|\mathcal{U}| + |\mathcal{U}||\mathcal{M}_s^{\max}|)$ where $|\mathcal{M}_s^{\max}| = \max_{s \in \mathcal{S}}(|M_s|)$.

VI. Experimental Design & Results

We consider numerical simulations and a real-world implementation with real image data and ML models. Algorithms are implemented in Python 3.8 and experiments are largely run on a macOS 64 bit machine with a 3.2 GHz quad-core Intel Core i5 processor and 32 GB 1600 MHz DDR3 memory.

A. Baselines

We compare AGP and EGP to the ILP defined in Eq. (7) which is solved using the PuLP Python library [34] and the CBC solver [35] (referred to as "OPT"). We also adapt the standard dynamic programming algorithm for the 0/1 Knapsack problem for the PIES problem (referred to as "SCK"). SCK considers the individual service models as the separate

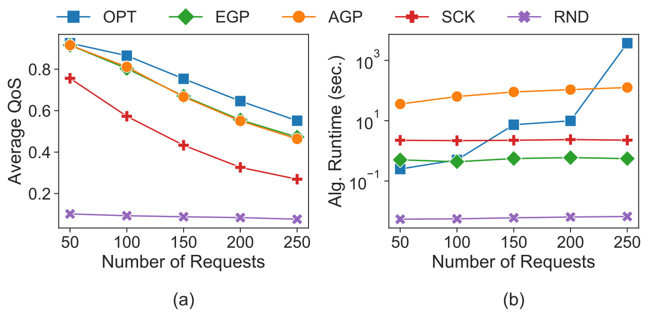

Fig. 3: **Validation Test.** For these experiments, we consider $|\mathcal{U}| = [50, 100, 150, 200, 250]$, with 10 trials each. We perform this validation test to confirm the efficacy of EGP relative to the optimal solution and the approximation algorithm, AGP.

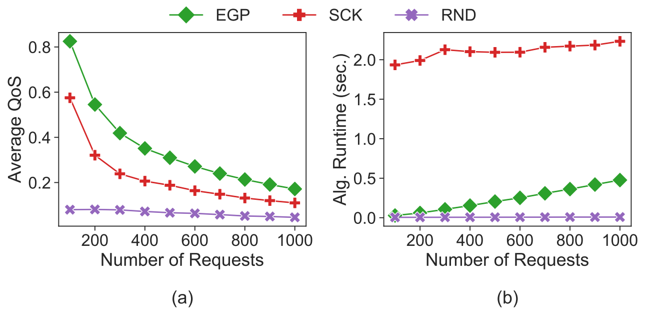

Fig. 4: **Numerical Results.** Experiments with more user requests $|\mathcal{U}| = [100, 200, \cdots, 1000]$, with 100 trials each.

items — with their storage costs serving as their weights and Eq. (1) as their values. SCK will use Alg. 1 for scheduling. We also consider a random placement and scheduling heuristic (referred to as "RND") as our other baseline.

B. Numerical Simulations

We sample uniformly random integer values for edge capacities using the following distributions $K_e, W_e \in [300, 600]$, and $R_e \in [100, 200]$ $(\forall e \in \mathcal{E})$. Service model storage costs are similarly uniformly sampled integer values where $k_{sm}, w_{sm} \in [15, 30]$ and $r_{sm} \in [10, 20]$ $(\forall (s,m) \in \mathcal{SM})$. Service models' cached accuracy, A_{sm}, are sampled from a Gaussian distribution with a mean of 0.65 and a standard deviation of 0.1 (sampled values are clipped to the range $[0, 1]$). We assume users request service types from $\mathcal{S}$ uniformly at random. User accuracy thresholds, α_u, are set to $1-\epsilon$ where ϵ is sampled from an exponential distribution clipped to the range $[0, 1]$ with $\lambda = 0.125$. User delay thresholds, δ_u, are set to a sampled value (clipped to range $[0, \delta_{\max}]$ where $\delta_{\max} = 10$ seconds) from an exponential distribution where $\lambda = 1.5$. Finally, we consider $|\mathcal{E}| = 10$ edge clouds, $|\mathcal{S}| = 100$ services with each service having a random number of implementations in the range of $[1, 10]$ (sampled uniformly). We increase the number of users for experiments.

First, we compare our proposed algorithms (EGP and AGP) to the optimal solution. Due to the hardness of the PIES problem, we consider a validation case to demonstrate EGP performance relative to the optimal solution[2] and AGP. In Fig. 3a, we see that the AGP and EGP algorithms perform well for approximating the optimal solution provided by the solver. More importantly, we find that EGP is able to match AGP's performance even though it has a proven approximation bound. Specifically, we find that, on average, AGP and EGP achieve an approximation ratio of 0.900 and 0.904, respectively.[3] In Fig. 3b, we see that EGP greatly outclasses both Optimal and AGP in terms of efficiency. The excessive cost associated with AGP's runtime is due to its reliance on performing optimal model scheduling for each candidate service model at each selection step. EGP also manages to best SCK. When considering more requests, we see in Fig. 4 that our EGP solution achieves roughly 50% more QoS than SCK while still managing to be more efficient.

C. Real-World Implementation

We consider a simple real-world setup using set of 2 Nvidia Jetson Nano and 1 Raspberry Pi 3B+ nodes as IoT devices and a 2013 Apple iMac serving as the edge cloud. Each IoT device hosts roughly a third of the 2012 ImageNet dataset [36] via non-overlapping subsets. Each IoT device submits 100 requests with randomly sampled images from these data. Requests are submitted wirelessly for image classification service models hosted on the edge cloud. Here we focus on the multi-implementation aspect of the PIES placement sub-problem where multiple implementations for image classification can be placed and how this affects the QoS in the real-world case. Using PyTorch [**?**], we evaluate pre-trained image classification models to record their accuracy metric over the ImageNet 2012 data and record the average time needed for each model to perform inference, see Table I. The edge cloud can place $|R_e| = 1$ model where each ML model is associated with $r_{sm} = 1$ storage cost. Since all the models accept the same data size, we fix the communication costs $w_{sm} = 1$ for all the ML models. The communication and computation capacities for the edge cloud are robustly tuned to match the real-world computation and communication delay. Similarly to the numerical simulation setup, each request's α_u is sampled from $1 - \epsilon$ where ϵ is sampled from an exponential distribution clipped to the range $[0.0, 1.0]$ with a rate parameter $\lambda = 0.0625$. Each sample's delay threshold δ_u is sampled from a Gaussian distribution with a mean of 0.5 and a standard deviation of 0.125, clipped to the range $[0, \delta_{\max}]$ where $\delta_{\max} = 1.0$ second. The QoS for each request is calculated using Eq. (1) using the real-time incurred latency (in seconds) and the evaluated model accuracy.

In Fig. 5a, we see that all considered algorithms but random are able to match the optimal solution — with all non-random algorithms exclusively placing MobileNet in Fig. 5b. In Fig. 5a, the QoS distribution of the non-random algorithms are much more concentrated on the upper end when compared to random. In this setup, random does better than in Figs. 3 ,4 because a request will never be dropped (i.e., there is always an image classification available to provide *some* QoS). Thus, future real-world implementations must consider various service types. For the meantime, these results show promise but could be improved and expanded upon by considering a more robust real-world setup with more EI service types (e.g., speech-to-text, video classification).

[2]It is worth noting in some larger scenarios we ran, it took over 20 hours for the optimal solver to complete.

[3]These values are computed by taking the QoS for each algorithm in one experiment trial and dividing it by the QoS provided by the optimal solution.

TABLE I: Image classifications models used for the real-world implementation with model accuracy metrics and average computational delay from evaluation with ImageNet 2012 data.

Models	Accuracy, A_{sm}	Avg. Comp. Delay (sec.)
AlexNet [37]	56.52%	0.04
DenseNet [38]	77.14%	0.47
GoogLeNet [39]	69.78%	0.13
MobileNet [40]	71.88%	0.06
ResNet [41]	69.76%	0.08
SqueezeNet [42]	58.09%	0.07

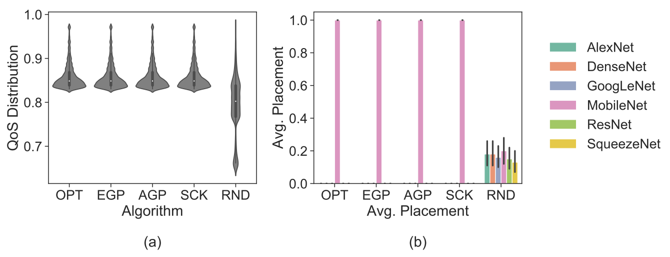

Fig. 5: **Real world implementation.** (5a) QoS distribution achieved by each of the algorithms. (5b) Average placement decision for each image classification model across 100 trials.

VII. Conclusions

The PIES problem, to the best of our knowledge, is the first service placement and scheduling problem that explicitly considers the case of EI services having multiple implementations available for the same service. We proved that the PIES problem is NP-*hard* and prove a greedy set optimization algorithm can provide a $(1 - 1/e)$-approximation guarantee of the optimal solution. We then introduce a streamlined greedy algorithm that empirically matches this algorithm's performance with much greater efficiency. While these results are preliminary, they serve as a foundational first step towards this breed of service placement. For future work, we plan to consider more dynamic extension of this work where service placement decisions are made over a time horizon rather than all at once. Additionally, we will expand the real-world setup to include more EI services (e.g., video classification) with multiple implementations for placement and scheduling.

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] ETSI, "Mobile edge computing - introductory technical white paper," Sept. 2014.

[2] W. Shi, J. Cao, Q. Zhang, Y. Li, and L. Xu, "Edge computing: Vision and challenges," *IEEE internet of things journal*, vol. 3, no. 5, 2016.

[3] Y. Mao, C. You, J. Zhang, K. Huang, and K. B. Letaief, "A survey on mobile edge computing: The communication perspective," *IEEE Communications Surveys & Tutorials*, vol. 19, pp. 2322–2358, 2017.

[4] 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.

[5] S.-H. Wang, P. P.-W. Huang, C. H.-P. Wen, and L.-C. Wang, "EQVMP: Energy-efficient and QoS-aware virtual machine placement for software defined datacenter networks," in *IEEE ICOIN*.

[6] I. Althamary, C.-W. Huang, P. Lin, S.-R. Yang, and C.-W. Cheng, "Popularity-based cache placement for fog networks," in *2018 14th IEEE IWCMC*, pp. 800–804, IEEE, 2018.

[7] B. Gao, Z. Zhou, F. Liu, and F. Xu, "Winning at the starting line: Joint network selection and service placement for mobile edge computing," in *IEEE INFOCOM*, pp. 1459–1467, IEEE, 2019.

[8] R. Mahmud, S. N. Srirama, K. Ramamohanarao, and R. Buyya, "Quality of experience (QoE)-aware placement of applications in fog computing environments," *Journal of Parallel and Distributed Computing*, 2019.

[9] T. He, N. Bartolini, H. Khamfroush, I. Kim, L. Ma, and T. La Porta, "Service placement for detecting and localizing failures using end-to-end observations," in *2016 IEEE ICDCS*, pp. 560–569, IEEE, 2016.

[10] M. Turner and H. Khamfroush, "Meeting users' QoS in a edge-to-cloud platform via optimally placing services and scheduling tasks," in *2020 IEEE ICNC*, pp. 368–372, IEEE, 2020.

[11] S. Wang, R. Urgaonkar, T. He, K. Chan, M. Zafer, and K. K. Leung, "Dynamic service placement for mobile micro-clouds with predicted future costs," *IEEE Transactions on Parallel and Distributed Systems*, vol. 28, no. 4, pp. 1002–1016, 2016.

[12] T. Ouyang, R. Li, X. Chen, Z. Zhou, and X. Tang, "Adaptive user-managed service placement for mobile edge computing: An online learning approach," in *IEEE INFOCOM*, 2019.

[13] 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 *2018 IEEE ICDCS*.

[14] T. Ouyang, Z. Zhou, and X. Chen, "Follow me at the edge: Mobility-aware dynamic service placement for mobile edge computing," *IEEE Journal on Selected Areas in Communications*, vol. 36, 2018.

[15] F. A. Salaht, F. Desprez, and A. Lebre, "An overview of service placement problem in fog and edge computing," *ACM Computing Surveys (CSUR)*, vol. 53, no. 3, pp. 1–35, 2020.

[16] O. Skarlat, M. Nardelli, S. Schulte, and S. Dustdar, "Towards QoS-aware fog service placement," in *2017 IEEE ICFEC*, IEEE, 2017.

[17] A. Yousefpour, A. Patil, G. Ishigaki, I. Kim, X. Wang, H. C. Cankaya, Q. Zhang, W. Xie, and J. P. Jue, "FogPlan: a lightweight QoS-aware dynamic fog service provisioning framework," *IEEE Internet of Things Journal*, vol. 6, no. 3, pp. 5080–5096, 2019.

[18] 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.

[19] D. Xu, T. Li, Y. Li, X. Su, S. Tarkoma, and P. Hui, "A survey on edge intelligence," *arXiv preprint arXiv:2003.12172*, 2020.

[20] X. Wang, Y. Han, V. C. Leung, D. Niyato, X. Yan, and X. Chen, "Convergence of edge computing and deep learning: A comprehensive survey," *IEEE Communications Surveys & Tutorials*, 2020.

[21] F. Manessi, A. Rozza, S. Bianco, P. Napoletano, and R. Schettini, "Automated pruning for deep neural network compression," in *2018 24th International Conference on Pattern Recognition (ICPR)*, IEEE, 2018.

[22] T.-J. Yang, Y.-H. Chen, and V. Sze, "Designing energy-efficient convolutional neural networks using energy-aware pruning," in *Proceedings of the IEEE CVPR*, pp. 5687–5695, 2017.

[23] P. S. Chandakkar, Y. Li, P. L. K. Ding, and B. Li, "Strategies for re-training a pruned neural network in an edge computing paradigm," in *2017 IEEE EDGE*, IEEE, 2017.

[24] 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, no. 8, 2019.

[25] 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.

[26] M. Hosseinzadeh, A. Wachal, H. Khamfroush, and D. E. Lucani, "Optimal accuracy-time trade-off for deep learning services in edge computing systems," *arXiv preprint arXiv:2011.08381*, 2020.

[27] Z. Lin, S. Bi, and Y.-J. A. Zhang, "Optimizing AI service placement and resource allocation in mobile edge intelligence systems," *arXiv preprint arXiv:2011.05708*, 2021.

[28] C.-C. Hung, G. Ananthanarayanan, P. Bodik, L. Golubchik, M. Yu, P. Bahl, and M. Philipose, "Videoedge: Processing camera streams using hierarchical clusters," in *2018 IEEE/ACM SEC*, IEEE, 2018.

[29] L. Caccetta and A. Kulanoot, "Computational aspects of hard knapsack problems," *Nonlinear Analysis*, vol. 47, no. 8, pp. 5547–5558, 2001.

[30] J. B. Kruskal, "On the shortest spanning subtree of a graph and the traveling salesman problem," *Proceedings of the American Mathematical society*, vol. 7, no. 1, pp. 48–50, 1956.

[31] N. Hudson, H. Khamfroush, and D. E. Lucani, "QoS-aware placement of deep learning services on the edge with multiple service implementations," *arXiv preprint arXiv:2011.08381*, 2021.

[32] G. L. Nemhauser, L. A. Wolsey, and M. L. Fisher, "An analysis of approximations for maximizing submodular set functions—I," *Mathematical programming*, vol. 14, no. 1, pp. 265–294, 1978.

[33] G. Calinescu, C. Chekuri, M. Pál, and J. Vondrák, "Maximizing a submodular set function subject to a matroid constraint," in *International Conference on Integer Programming and Combinatorial Optimization*, pp. 182–196, Springer, 2007.

[34] S. Mitchell, S. M. Consulting, and I. Dunning, "PuLP: A linear programming toolkit for Python," 2011.

[35] johnjforrest, S. Vigerske, H. G. Santos, T. Ralphs, L. Hafer, B. Kristjansson, jpfasano, EdwinStraver, M. Lubin, rlougee, jpgoncal1, h-i gassmann, and M. Saltzman, "coin-or/Cbc: Version 2.10.5," Mar. 2020.

[36] O. Russakovsky, J. Deng, H. Su, J. Krause, S. Satheesh, S. Ma, Z. Huang, A. Karpathy, A. Khosla, M. Bernstein, A. C. Berg, and L. Fei-Fei, "ImageNet Large Scale Visual Recognition Challenge," *International Journal of Computer Vision (IJCV)*, 2015.

[37] A. Krizhevsky, "One weird trick for parallelizing convolutional neural networks," *arXiv preprint arXiv:1404.5997*, 2014.

[38] G. Huang, Z. Liu, L. Van Der Maaten, and K. Q. Weinberger, "Densely connected convolutional networks," in *Proceedings of the IEEE CVPR*, pp. 4700–4708, 2017.

[39] C. Szegedy, W. Liu, Y. Jia, P. Sermanet, S. Reed, D. Anguelov, D. Erhan, V. Vanhoucke, and A. Rabinovich, "Going deeper with convolutions," in *Proceedings of the IEEE CVPR*, pp. 1–9, 2015.

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

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

[42] F. N. Iandola, S. Han, M. W. Moskewicz, K. Ashraf, W. J. Dally, and K. Keutzer, "Squeezenet: Alexnet-level accuracy with 50x fewer parameters and¡ 0.5 mb model size," *arXiv preprint arXiv:1602.07360*, 2016.