An Online Approach for DNN Model Caching and Processor Allocation in Edge Computing

Zhiqi Chen[†], Sheng Zhang[†], Zhi Ma[†], Shuai Zhang[†], Zhuzhong Qian[†], Mingjun Xiao[§], Jie Wu[‡], Sanglu Lu[†]
[†]State Key Lab. for Novel Software Technology, Nanjing University, P.R. China
[§]School of Computer Science and Technology, University of Science and Technology of China, P.R. China
[‡]Center for Networked Computing, Temple University
Email: sheng@nju.edu.cn

Abstract—**Edge computing is a new computing paradigm rising gradually in recent years. Applications, such as object detection, virtual reality and intelligent cameras, often leverage Deep Neural Networks (DNN) inference technology. The traditional paradigm of DNN inference based on cloud suffers from high delay because of the limited bandwidth. From the perspective of service providers, caching DNN models on the edge brings several benefits, such as efficiency, privacy, security, etc.. The problem we concerned in this paper is how to decide the cached models and how to allocate processors of edge servers to reduce the overall system cost. To solve it, we model and study the DNN Model Caching and Processor Allocation (DMCPA) problem, which considers user-perceived delay and energy consumption with limited edge resources. We model it as an integer nonlinear programming (INLP) problem, and prove its NP-Completeness. Since it is considered as a long-term average optimization problem, we leverage the Lyapunov framework to develop a novel online algorithm DMCPA-GS-Online with Gibbs Sampling. We give the theoretical analysis to prove that our algorithm is near-optimal. In experiments, we study the performance of our algorithm and compare it with other baselines. The simulation results with the trace dataset from real world demonstrate the effectiveness and adaptiveness of our algorithm.**

Index Terms—**Edge Computing, DNN Model Caching, Proximity Inferences, Gibbs Sampling, Lyapunov Optimization**

I. Introduction

The rapid development of 5G and Artificial Intelligence (AI) in recent years [1], [2] result in the emergence of a large number of smart applications and smart devices, making people's lives more convenient and intelligent. Therefore, how to support the smart life of users with better Quality of Service (QoS) [3] has attracted more and more attention from service providers, for example, Virtual Reality (VR), intelligent cameras and so on. These applications need to transfer a large amount of data between user-side and cloud-side inevitably, which overwhelms the core network and causes congestion and blockages in it [4]. Fortunately, service providers have realized that the traditional cloud computing paradigm is falling behind modern applications and find the new network computing paradigm, which is called Edge Computing (EC) [5], [6].

Edge computing can effectively enhance the QoS at the network edge. An edge server can be regarded as a cloudlet with limited storage and computation ability, which is equipped with CPU, GPU or other computing or processing devices for deep learning inference tasks. We are able to deploy DNN models on edge servers for users to conduct "proximity inferences" [7] in virtue of the following advantages of EC: (1) Edge servers are located at the network edge, which means they are close enough to users to reduce the raw data transmission consumption during the DNN inference process. (2) DNN inference requests from users often comprise local geographic characteristics, such as similar inference tasks [8], so deploying a portion of deep models which are hotspots on edge servers in different locations can efficiently reduce communication delay and avoid excessive model deployment cost on the edge servers. (3) Users need to submit large amounts of input data to deep models for inference, however the bandwidth of the core network is usually insufficient to handle the excessive transmission demand generated from the edge. In EC paradigm [9], the data transmission demand is greatly reduced because most of them are handled by edge servers, immensely relieving the transmission pressure of the core network. (4) Edge computing permits to conduct inference on nearby servers rather than transmit sensitive data to the remote cloud, and protects user privacy effectively [10].

Since the storage and computation resources of the remote cloud can be infinite in most cases, we can assume that the cloud prepares all the deep models in advance that may be requested by users, while edge servers are limited by resources and can only undertake a subset of all these deep models. Moreover, a user usually connects to one edge server with requests to a specified deep model. If the requested deep model is deployed on the connected edge server coincidentally, the network transmission delay between the user device and the cloud is completely eliminated, which is called "edge-hit". As shown in the above, QoS experience will reach the best if the edge server is powerful and voluminous enough to accommodate all deep models which may be requested. However, edge servers are usually provided with limited storage and computation resources, and service providers are limited by service costs and cannot extend the service abilities of services infinitely. So it is not possible to cache all the deep models on edge servers, but only a subset of them can be selected for deployment limitedly, which brings a conflict between QoS experience and edge capacity. More specifically, an edge server can be equipped with several DNN processors and each one has provisioning cost and maintaining cost that the service provider needs to consider. Consequently, it brings us the first challenge: *Which and how many DNN processors need to equip on each edge server, making the cost of provisioning and maintaining DNN processor hardwares by the service provider as little as possible.* Another problem is that the computation and storage capacity of DNN processor is often described by its floating-point computation power flops and memory size, which means the number of DNN models that can be deployed on a DNN processor is limited, while the coming user request follows a certain distribution. The fact brings us the second challenge: *How to cache the DNN Model based on the known distribution of user requests to minimize the overall cost of user QoS-aware delay violations.*

The main contributions of this paper are as follows.

- We consider the DNN Model Caching and Processor Allocation problem in EC environments. The edge resources are assumed limited while the cloud is nearly unlimited but constrained by bandwidth. Besides, the characteristics of user request distribution (or user mobility) are considered for optimization.
- Under the constraints of DNN model inference delay, we formulate the problem with the purpose of reducing user perception delay and energy consumption cost with careful model caching and processor allocation strategy. The problem is an Integer Nonlinear Program (INLP) , which is proved to be NP(Non-deterministic Polynomial)-Complete.
- We propose a novel online algorithm called DMCPA-GS-Online, which can efficiently cache models and allocate processors in an online manner without future information. DMCPA-GS-Online leverages the Lyapunov framework to transform the origin problem into a series of time-stepping subproblems. These subproblems can be solved by an extended version of Gibbs Sampling Algorithm, which is proved to be near-optimal.
- We evaluate the performance of DMCPA-GS-Online through a trace dataset from the real world practically and extensively. The results demonstrate that our proposed algorithm outperforms other baselines.

The rest of this paper is organized as follows. Section II reviews related works. Section III gives the formulation of our problem. Section IV introduces our proposed algorithm DMCPA-GS-Online and gives theoretical analysis. We evaluate our model and algorithm in Section V. Finally, Section VI concludes this paper.

II. Related Work

In recent years, edge computing is attracting more and more attentions in network field. With the breakthrough and development of deep learning [11], artificial intelligence applications are developing rapidly. The massive amount of data generated by deep learning needs to be processed at the edge in a timely manner [12]. Some works investigate decouple deep structure [13]–[15] with dividing a complex network into multiple parts and deploying them on cloud and edge servers respectively, but it will inevitably bring additional network transmission overhead. Another way considers deep compressive offloading to speed neural network inference by trading edge computation for network latency [16]. Different from these works, our work focus on caching deep learning-based intelligent services to the edge network to enhance the user experience of services at the edge.

Since DNN model can be viewed as service, there has been many works considering service placement in edge computing [17]–[26]. For example, in [18], the authors consider how to deploy collaborative edge applications to achieve the best overall system performance. In [19], a heterogeneous MEC (Mobile Edge Computing) system is considered and it focuses on the problem of placing multiple services in the system to maximize the total reward. The problem of jointly optimizing access network selection and service placement for MEC is investigated with the aim of improving QoS by balancing access, switching and communication delays [20]. The problem of joint optimization of service placement and request routing in MEC-enabled multicell networks with multidimensional (storage-compute-communication) constraints is investigated [22], [23]. However, these works have no specific analysis for DNN tasks, and the impact of input data on network transmission and inference delay is not fully considered, which causes insufficient modeling.

In addition, the uncertainty of users has brought a series of studies on dynamic edge computing environments [27]–[30]. In [27], the authors consider the user request admission problem and use data mining and machine learning technology to estimate the user historical trajectory. However, it will inevitably bring estimation error and affect the system performance. In our work, we solve the user uncertainty without future information in an online manner. The dynamic service caching and task offloading problem [28] in a 5G-enabled MEC with varying user demand and processing delay is investigated [29], and an online learning algorithm is proposed for the problem using multi-armed bandit (MAB) technique. In [30], the performance optimization problem of mobile edge services under long-term cost budget constraints is investigated, and the Lyapunov optimization method is applied. Compared with our work, it only considers the migration of services between edge servers, while neglects the power of service with cloud-edge collaboration.

III. Problem

In this section, we first introduce our DNN Model Caching and Processor Allocation problem with "edge-hit" and "cloud-hit". Then we explain our overall system model with the definition of user perception delay. Our goal is to minimize the overall system cost of delays and energy with constraint edge server capacities.

A. Overview

We consider such a caching scenario in edge as shown in Fig. 1. Base stations (BSs) are located in the area where DNN inference requests from users are raised in an online manner. Each BS is equipped with an edge server that provides computation and storage capacity. Besides, these edge servers are equipped with multiple processors with DNN computation ability for inference. These BSs only serve users in edge regions that they cover, where the edge region means the

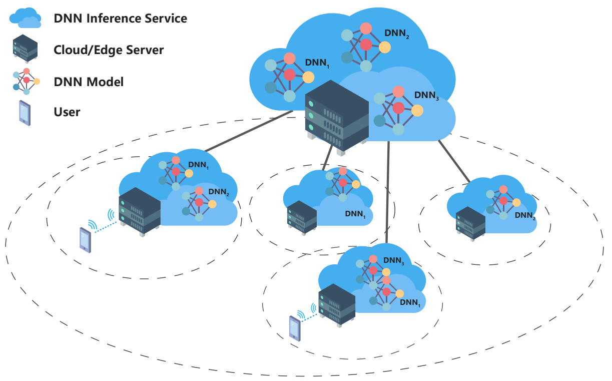


Fig. 1. An overview of our problem.

area that all user requests from which can be handled by the edge. To simplify, users prefer to access the BS with lower delay and better signal quality if they are in an overlap among several edge regions. Each BS is connected to the cloud server via the backbone network. Since edge servers have no enough computation and storage capacity to cache all kinds of DNN models, a user request will encounter two caching hit conditions: *Edge-Hit* and *Cloud-Hit*, which is determined by the case that the model is cached or not.

- **Edge-Hit**: When a user sends a DNN inference request to the nearest BS and the requested DNN model is already cached on the edge server, the request can be directly processed by the DNN model on the edge server. The inference result can be quickly returned to the user.

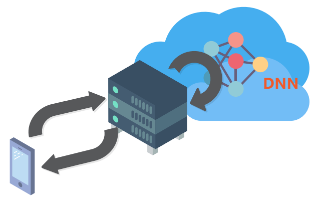


Fig. 2. Edge-Hit

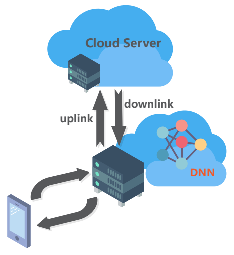


Fig. 3. Cloud-Hit

- **Cloud-Hit**: If the request arrives at the BS but the edge server does not cache the target DNN model, it will be forwarded to the cloud server, resulting in the edge-cloud data transmission delay depending on the user data size. Once the cloud server finishes the DNN inference task and returns to the edge, the edge forwards the result to the user with the cloud-edge downlink delay.

Obviously, *Edge-Hit* entails less transmission delay than Cloud-Hit. However, it demands more caching cost of DNN models provided by the edge server. How to achieve a good trade-off between QoS experience and DNN caching cost is one of our study goal.

B. The Model and Notations

We first denote the set of edge servers located in the fixed area as $\mathbb{S}$, where $S_i \in \mathbb{S}$ represents the i-th edge server and the total size is N_s. The set of users is denoted as $\mathbb{U}$, where $U_k \in \mathbb{U}$ represents the k-th user and the total size is N_u. Meanwhile, $\mathbb{M}$ represents the set of all deep models, where

TABLE I
KEY NOTATIONS FOR DMCPA PROBLEM

Notation	Definition
$\mathbb{G} = (\mathbb{S}, \mathbb{U}, \mathbb{M})$	The overall set $\mathbb{G}$ for DMCPA problem.
$\mathbb{S}$	The set of edge servers, where $S_i \in \mathbb{S}$ and $\lvert\mathbb{S}\rvert = N_s$.
$\mathbb{U}$	The set of users, where $U_k \in \mathbb{U}$ and $\lvert\mathbb{U}\rvert = N_u$.
$\mathbb{M}$	The set of models, where $M_j \in \mathbb{M}$ and $\lvert\mathbb{M}\rvert = N_m$.
$x_{i,j}(t) \in \{0, 1\}$	A binary indicator, whether DNN model M_j is cached on edge server S_i at time slot t.
$y_{i,j}(t) \in \mathbb{Z}$	An integer indicator, which indicates the number of processors allocated for Model M_j by edge server S_i at time slot t.
d_k^{input}	The input data size uploaded by user U_k for DNN inference, which obeys a certain distribution at time slot t.
$c_k^{input}(t)$	The inference computing workload of user U_k, $c_k^{input}(t) = \theta d_k^{input}(t)$, where θ is a constant that indicates the computation workload per bit.
$P_{k,i}(t)$	The distribution that the user U_k attached to the edge server S_i at time slot t.
D_j	The maximum inference delay model U_j are expected.
w_j	The resource vector of M_j that occupies on edge server.
$e_{i,j}$	The energy consumed by a single processor for model M_j on edge server S_i.
a_j	The initialization cost for DNN Model $\mathcal{M}_j$ caused by loading models on edge servers.
$L^{cloud\downarrow}$	The propagation delay of the cloud server to return the inference results.
$c_{i,j}$	The inference computing ability of a single processor for model M_j on edge server S_i.
ϕ_i	The maximum number of processors that can be provided by server S_i.
W_i	The maximum resource vector of edge server S_i can provide.
$L_{k,i}^{acc}$	The BS access delay of user U_k to edge server S_i.
B_i^{cloud}	The bandwidth between edge server S_i and the cloud server.

$M_j \in \mathbb{M}$ represents the j-th type of deep models and the total size is N_m. These can be represented by the overall set $\mathbb{G} = (\mathbb{S}, \mathbb{U}, \mathbb{M})$.

We believe that the distribution of user connection to edge servers deserves consideration. We use $P_{k,i}(t)$ to indicate the probability of a user U_k attached to edge server S_i at time slot t. The reasons behind are as follows [31]: (1) service providers can easily estimate user probabilities by statistics on edge servers; (2) user requests usually follow a certain geographic and time pattern, which can be captured as constant in one time slot; (3) the expectation of the cost under a certain policy can be calculated and we can optimize it to reduce the overall cost based on the distribution.

B_i^{cloud} denotes the bandwidth from edge server S_i to the cloud server. $L_{k,i}^{cloud\uparrow} = \frac{d_k^{input}}{B_i^{cloud}}$ denotes the transmission delay of the DNN input of user U_k submitted to S_i and sent from S_i to reach the cloud. $L^{cloud\downarrow}$ denotes the downlink delay of the inference result returned by the cloud, which we consider as a constant because the result size of DNN inference is usually small enough. D_j denotes the maximum inference delay model M_j can guarantee.

Here $c_{i,j}$ denotes the inference computation ability provided by a single DNN processor on edge server S_i for model M_j. We use $c_k^{input} = \theta d_k^{input}$ to denote the computation workload by user U_k, where θ is a constant that indicates the computation requirement per bit.

As for the decision variables, we use the binary indicator $x_{i,j}(t) \in \{0,1\}$ to indicate whether the model M_j is cached on edge server S_i at time slot t as our cache decision vector. When $x_{i,j}(t) = 1$, it means that DNN model M_j is cached on S_i and vice versa. We use the integer indicator $y_{i,j}(t) \in \mathbb{N}$ to indicate the number of DNN processors on edge server S_i as the processor allocation vector. The maximum number of processors that edge server S_i can provide is denoted as ϕ_i.

C. User Perception Delay

User perception delay consists of three parts: server access delay, network transmission delay and model inference delay.

- **Server Access Delay**: It represents the delay from the submission of user U_k DNN inference request to the edge server S_i, which is denoted by $L_{k,i}^{acc}$ [27].
- **Network Transmission Delay**: If model M_j is cached on the edge server S_i, network transmission delay is considered as 0; otherwise it is calculated as $L_{k,i}^{cloud} = L_{k,i}^{cloud\uparrow} + L^{cloud\downarrow} = \frac{d_k^{input}}{B_i^{cloud}} + L^{cloud\downarrow}$.
- **Model Inference Delay**: If model M_j is cached on the edge server, the model inference delay is related to the number of processor deployment as $L_{k,i,j}^{edge} = \frac{\theta d_k^{input}(t)}{y_{i,j}(t)c_{i,j}}$; if model M_j is not cached, the model inference delay depends on the inference delay of the cloud server, which can be neglected (or represented as a constant for generalization) since the cloud computation capacity is viewed as nearly unlimited.

According to the above definition, the user perception delay of user U_k accessing server S_i for requesting DNN model M_j can be calculated as:

$$L_{k,i,j}(t) = L_{k,i}^{acc} + (1 - x_{i,j}(t))L_{k,i}^{cloud} + x_{i,j}(t)L_{k,i,j}^{edge} \quad (1)$$

D. The Formulation of Problem

With the distribution of $P_{k,i}(t)$, the expectation of model average delay of model M_j can be calculated as:

$$L_j^P(t) = \frac{1}{N_u} \sum_{U_k \in \mathbb{U}} \sum_{S_i \in \mathbb{S}} P_{k,i} L_{k,i,j}(t). \quad (2)$$

To achieve a good QoS experience, we consider that the maximum inference delay D_j that the model M_j can guarantee. If the average delay exceeds D_j, then QoS experience will be degraded; otherwise it will not affect. Based on this observation, we come up with the violation metric of model delay as follows:

$$\mathcal{T}^d(t) = \sum_{M_j \in \mathbb{M}} \max\{L_j^P(t) - D_j, 0\} \quad (3)$$

Obviously, edge server S_i is limited by the maximum resource capacity, which can be described as:

$$\sum_{M_j \in \mathbb{M}} x_{i,j}(t) w_j \le W_i, \forall S_i \in \mathbb{S}, \quad (4)$$

where w_j denotes the weight vector of resources required by DNN model M_j, i.e., gflops, gpu memory. W_i denotes the maximum resource vector that edge server S_i can provide.

Since deploying processors on each server requires the additional deployment cost (pay-as-you-go model) [32], which is usually related to the purchase cost of processors and consumption cost to maintain, we do not wish to overpay this deployment cost from the service provider's point of view. Therefore we use the following equation as part of our optimization objective:

$$\mathcal{T}^e(t) = \sum_{S_i \in \mathbb{S}} \sum_{M_j \in \mathbb{M}} e_{i,j} y_{i,j} + a_j \max\{x_{i,j}(t) - x_{i,j}(t-1), 0\}, \quad (5)$$

where a_j denotes the initialization cost for DNN Model $\mathcal{M}_j$ caused by loading models from disks on edge servers. The cost is considered in the case that $x_{i,j}(t) = 1$ and $x_{i,j}(t-1) = 0$, which means M_j is just deployed on S_i at the time slot t. $e_{i,j}$ denotes the energy consumed by a single processor for model M_j on edge server S_i.

The entire problem, which is called DNN Model Caching and Processor Allocation (DMCPA) problem, can be described in formulation as follows, where α and β is the trade-off knob between user perception delay and energy consumption:

$$\mathcal{P}^0 : \min_{\forall t, x(t), y(t)} \quad \lim_{T \to \infty} \frac{1}{T} \sum_{t=0}^{T-1} \mathbb{E}[\mathcal{T}(t)] \quad (6)$$

$$\begin{aligned}
s.t. \quad & C_1 : \quad \mathcal{T}(t) = \alpha \mathcal{T}^d(t) + \beta \mathcal{T}^e(t), && (7)\\
& C_2 : \quad \sum_{M_j \in \mathbb{M}} x_{i,j}(t) w_j \le W_i, \forall S_i \in \mathbb{S}, && (8)\\
& C_3 : \quad \sum_{M_j \in \mathbb{M}} y_{i,j}(t) \le \phi_i, \forall S_i \in \mathbb{S}, && (9)\\
& C_4 : \quad x_{i,j}(t) \in \{0,1\}, \forall S_i \in \mathbb{S}, \forall M_j \in \mathbb{M}, && (10)\\
& C_5 : \quad y_{i,j}(t) \in \mathbb{Z}, \forall S_i \in \mathbb{S}, \forall M_j \in \mathbb{M}, && (11)\\
& C_6 : \quad \min\{y_{i,j}(t), 0\} \le x_{i,j}(t) \le y_{i,j}(t), && (12)\\
& C_7 : \quad \lim_{T \to \infty} \frac{1}{T} \sum_{t=0}^{T-1} \mathbb{E}[L_j^P(t)] \le D_j, \forall M_j \in \mathbb{M}. && (13)
\end{aligned}$$

C_1 gives the definition of $\mathcal{T}(t)$ in Eq. 6. C_2 ensures that the sum of the model-consuming resources hosted on edge server S_i does not exceed the maximum resources that the server can provide. C_3 ensures that the number of processors allocated to DNN services on server S_i does not exceed the total number of processors on S_i. C_4 and C_5 ensure that the final solution is feasible. C_6 is based on the observation that $x_{i,j}(t) = \min\{y_{i,j}(t), 1\}$ since allocating processors to model which is not cached is unreasonable. C_7 ensures the average user perception delay should not exceed the maximum toleration delay D_j required by M_j.

E. The Complexity Analysis of DMCPA Problem

When focusing on the one step optimization target $\mathcal{T}(t)$ without considering the long-term average target, it is an Integer Nonlinear Programming problem (INLP). We prove that Knapsack Problem can be reduced to a special case of DMCPA problem:

Theorem 1. *Knapsack problem $\leq_P$ DMCPA problem.*

Proof. First we fix all the variables of $y(t)$ of DMCPA problem. Thus the objective value is only related to $x(t)$. For the objective part, the original nonlinear function can be transformed into the linear function by assigning $\beta = 0$, $D_k = 0$, $P_{k,i}(t) = 1$, $L^{acc}_{k,i} = 0$, $c_{i,j} = \infty$ and $L^{cloud\downarrow} = 0$. Then we have the optimization target $\max \sum_{i,j,k} \frac{d_k^{input}}{B_i^{cloud}} x_{i,j}$ with constraints $\sum_{M_j \in \mathbb{M}} x_{i,j}(t) w_j \leq W_i$ and $x_{i,j}(t) \in \{0, 1\}$. It is a Multiple Knapsack Problem (MKP), which can be reduced to Knapsack Problem as proved widely [33]. So we come to the conclusion: Knapsack Problem $\leq_P$ DMCPA problem. $\square$

Lemma 2. *DMCPA problem belongs to NP-Complete.*

Proof. Firstly we prove that DMCPA problem belongs to NP. The decision version of DMCPA problem can be verified in polynomial time, so DMCPA problem belongs to NP. Since Knapsack problem belongs to NP-Complete and DMCPA problem is harder than Knapsack Problem, DMCPA problem belongs to NP-Complete. $\square$

IV. Algorithm

In this section, we introduce the optimization algorithm of DMCPA problem. The long-term model delay tolerance constraint in $\mathcal{P}^0$ makes it difficult to solve with an online schema. So we leverage the Lyapunov technique [34] by transforming the original time-exception problem into a series of minimization problems in one time slot. Then, we propose an online algorithm regardless of global information in future by solving a subproblem with Gibbs Sampling.

A. Problem Transformation by Lyapunov

In the original problem $\mathcal{P}^0$, we have no knowledge of the arrival of user requests, such as c_k^{input}, making it difficult to guarantee the user delay constraint inevitably. Fortunately, we can construct virtual queues using Lyapunov optimization, which can depict the impact of this stochastic constraint for solving this problem. These queues hold the residual of the delay constraints for the current time slot. Our goal is to keep the queues stable as much as possible while optimizing the long-term objective. And for each model M_j, there is a corresponding queue $Q_j(t)$ with an initial value of 0:

$$Q_j(t+1) = \max\{Q_j(t) + L_j^P(t) - D_j, 0\}. \tag{14}$$

$Q_j(t)$ portrays the j-th queue's backlog for the t-th time slot. In fact, the $t+1$-th queue is derived by the t-th queue and its arrival/departure increments of the current time slot.

From the above equation, it follows that:

$$Q_j(t+1) \geq Q_j(t) + L_j^P(t) - D_j \tag{15}$$

$$L_j^P(t) - D_j \leq Q_j(t+1) - Q_j(t) \tag{16}$$

$$\sum_{t=0}^{T-1} L_j^P(t) - D_j \leq Q_j(T) - Q_j(0) = Q_j(T) \tag{17}$$

$$\frac{1}{T}\sum_{t=0}^{T-1} \mathbb{E}[L_j^P(t) - D_j] \leq \frac{\mathbb{E}[Q_j(T)]}{T} \tag{18}$$

Recall C_7 in $\mathcal{P}^0$, we can use Eq. 18 to convert it to the following new constraint:

$$\lim_{T\to\infty} \frac{\mathbb{E}[Q_j(T)]}{T} \leq 0 \tag{19}$$

So we have the new problem $\mathcal{P}^1$:

$$\mathcal{P}^1 : \min_{\forall t, x(t), y(t)} \quad \lim_{T\to\infty} \frac{1}{T} \sum_{t=0}^{T-1} \mathbb{E}[\mathcal{T}(t)] \tag{20}$$

$$s.t. \qquad (C_1) - (C_6), \tag{21}$$

$$C_7 : \quad \lim_{T\to\infty} \frac{\mathbb{E}[Q_j(T)]}{T} \leq 0, \forall M_j \in \mathbb{M}. \tag{22}$$

Let $\Theta(t) = [Q_1(t), \cdots, Q_{N_m}(t)]$. We define the quadratic Lyapunov function: $L(\Theta(t)) = \frac{1}{2}\sum_j Q_j(t)^2$ firstly. Then, we can use $\Delta(\Theta(t)) = L(\Theta(t+1)) - L(\Theta(t))$ to present the increment of all queues' backlog, which is called Lyapunov drift. Using Lyapunov drift, we have the drift-plus-penalty algorithm in Lyapunov optimization.

Theorem 3. *For the possible values of $Q_j(t)$ in all time slots, we have the following stable statement:*

$$\Delta(\Theta(t)) \leq B + \sum_j Q_j(t)(L_j^P(t) - D_j), \tag{23}$$

where $B = \frac{1}{2}\sum_j (L_j^P(t) - D_j)^2$ is a constant value.

Proof. According to Eq. 14, the following inequality holds:

$$Q_j(t+1)^2 \leq (Q_j(t) + L_j^P(t) - D_j)^2. \tag{24}$$

So we have the following conclusion by accumulation:

$$\begin{aligned} &\frac{1}{2}\sum_j Q_j(t+1)^2 \leq \frac{1}{2}\sum_j Q_j(t)^2 \\ &+ \frac{1}{2}\sum_j (L_j^P(t) - D_j)^2 + \sum_j Q_j(t)(L_j^P(t) - D_j). \end{aligned} \tag{25}$$

$$\begin{aligned} &\Delta(\Theta(t)) = \frac{1}{2}\sum_k Q_k(t+1)^2 - \frac{1}{2}\sum_k Q_k(t)^2 \\ &\leq \frac{1}{2}\sum_j (L_j^{P,Q}(t) - D_j)^2 + \sum_j Q_j(t)(L_j^P(t) - D_j) \\ &\leq B + \sum_j Q_j(t)(L_j^P(t) - D_j). \end{aligned} \tag{26}$$

As a result, the above theorem is proved. $\square$

After introducing Lyapunov queues, we can optimize without breaking the convention that the current time slot needs to follow, which means the control queue remains stable.

$$\mathcal{P}^2 : \min_{\forall t, x(t), y(t)} \quad \mathbb{E}[\Delta(\Theta(t)) + V \cdot \mathcal{T}(t) | \Theta(t)] \tag{27}$$

$$s.t. \qquad (C_1) - (C_6), \tag{28}$$

$$C_7: \qquad \lim_{T\to\infty} \frac{\mathbb{E}[Q_j(T)]}{T} \le 0, \forall M_j \in \mathbb{M}. \tag{29}$$

Recall Eq. 23, the new problem can be expressed as:

$$\mathcal{P}^3: \min_{\forall t, x(t), y(t)} \quad \mathbb{E}[B + V\cdot \mathcal{T}(t) + \tag{30}$$

$$\sum_j Q_j(t)(L_j^P(t) - D_j)|\Theta(t)] \tag{31}$$

$$s.t. \qquad (C_1) - (C_6), \tag{32}$$

$$C_7: \qquad \lim_{T\to\infty} \frac{\mathbb{E}[Q_j(T)]}{T} \le 0, \forall M_j \in \mathbb{M}. \tag{33}$$

B. The DMCPA-GS-Online Algorithm

We have the online algorithm for DMCPA problem as shown in Alg. 1. First we initialize $Q_j(0)$ as zero. For each time slot, we receive the user input $d_k^{input}(t)$ and current distribution P from the EC environment. With Alg. 2, get the optimal solution $x^*(t)$ and $y^*(t)$ by solving $\mathcal{P}^3$ in the current time slot. At the end of each iteration, update Q_j by Eq. 14.

Algorithm 1: The DMCPA-GS-Online Algorithm

```
Input: $Q_j(0) \leftarrow 0, x^{prev}(0) \leftarrow 0$
for $t = 0$ to $T$ do
    receive $d_k^{input}(t)$ from environment;
    update current distribution $P$ from environment;
    get $x^*(t)$, $y^*(t)$ by solving $\mathcal{P}^3$ using Alg. 2;
    $x^{prev}(t+1) \leftarrow x^*(t)$;
    for $M_j \in \mathbb{M}$ do
        $Q_j(t+1) \leftarrow \max\{Q_j(t) + L_j^P(t) - D_j, 0\}$;
    end
end
```

Theorem 4. *The time average penalty (optimization goal) of DMCPA-GS-Online is $O(\frac{1}{V})$.*

Proof. According to Eq. 26 and the fact that $x^*(t), y^*(t)$ is the optimal value of $\mathcal{P}^3$, we have:

$$\begin{aligned}\Delta(\Theta(t)) + V\cdot\mathcal{T}(t) &\le B + V\cdot\mathcal{T}(t) + \sum_j Q_j(t)(L_j^P(t) - D_j)\\ &= B + V\cdot\mathcal{T}(d^{input}(t), x^*(t), y^*(t))\\ &+ \sum_j Q_j(t)(L_j^P(d^{input}(t), x^*(t), y^*(t)) - D_j)\end{aligned} \tag{34}$$

where $d^{input}(t) = [d_1^{input}(t), \cdots, d_{N^u}^{input}(t)]$. So we have the following expectation result:

$$\begin{aligned}&\mathbb{E}[\Delta(\Theta(t)) + V\cdot\mathcal{T}(t)|\Theta(t)]\\ &\le \mathbb{E}[B + V\cdot\mathcal{T}(d^{input}(t), x^*(t), y^*(t))\\ &+ \sum_j Q_j(t)(L_j^P(d^{input}(t), x^*(t), y^*(t)) - D_j)]\\ &= B + V\cdot\mathbb{E}[\mathcal{T}(d^{input}(t), x^*(t), y^*(t))] +\\ &\sum_j \mathbb{E}[Q_j(t)|\Theta(t)]\mathbb{E}[L_j^P(d^{input}(t), x^*(t), y^*(t)) - D_j]\\ &\le B + V\cdot\mathcal{T}^*,\end{aligned} \tag{35}$$

where $\mathcal{T}^* = \min(\lim_{T\to\infty}\frac{1}{T}\sum_{t=0}^{T-1})\mathbb{E}[\mathcal{T}(t)]$.

Accumulate the above equation on all time slots, we have:

$$\begin{aligned}&(B + V\cdot\mathcal{T}^*)T \ge \sum_{t=0}^{T-1}\mathbb{E}[\Delta(\Theta(t)) + V\cdot\mathcal{T}(t)|\Theta(t)] =\\ &\mathbb{E}[L(\Theta(T))] + V\cdot\sum_{t=0}^{T-1}\mathbb{E}[\mathcal{T}(t)|\Theta(t)] \ge V\cdot\sum_{t=0}^{T-1}\mathbb{E}[\mathcal{T}(t)|\Theta(t)].\end{aligned} \tag{36}$$

Therefore, we can draw the final conclusion:

$$\lim_{T\to\infty}\frac{1}{T}\sum_{t=0}^{T-1}\mathbb{E}[\mathcal{T}(t)|\Theta(t)] \le \mathcal{T}^* + \frac{B}{V}. \tag{37}$$

As long as V is set large enough, the solution obtained by executing the drift-plus-penalty algorithm can approach the optimization goal $\mathcal{T}^*$ of $\mathcal{P}^1$, which obeys $O(\frac{1}{V})$. □

Theorem 5. *Using DMCPA-GS-Online, all queues keep mean rate stable and the constraint C_7 in $\mathcal{P}^3$ can be neglected. In other words, the long-term optimization problem can be addressed by solving a series of subproblems $\mathcal{P}^3$ without C_7 in each time slot.*

Proof. $\forall M_j \in \mathbb{M}$, suppose we have the solution $x(t), y(t)$ which is not optimal satisfied:

$$\exists\epsilon > 0, \mathbb{E}[L_j^P(d^{input}(t), x^\star(t), y^\star(t)) - D_j] \le -\epsilon. \tag{38}$$

Let denote $\mathcal{T}_{min}$ and $\mathcal{T}_{max}$ which satisfies $\mathcal{T}_{min} \le \mathcal{T}(d^{input}(t), x(t), y(t)) \le \mathcal{T}_{max}$, so we have the conclusion, where $B' = B + V(\mathcal{T}_{max} - \mathcal{T}_{min})$:

$$\begin{aligned}&\Delta(\Theta(t)) + V\cdot\mathcal{T}(t) \le B + V\cdot\mathcal{T}(d^{input}(t), x^\star(t), y^\star(t))\\ &+ \sum_j Q_j(t)(L_j^P(d^{input}(t), x^\star(t), y^\star(t)) - D_j)\end{aligned} \tag{39}$$

$$\begin{aligned}&\Delta(\Theta(t)) + V\cdot\mathcal{T}_{min} \le B + V\cdot\mathcal{T}_{max}\\ &+ \sum_j Q_j(t)(L_j^P(d^{input}(t), x^\star(t), y^\star(t)) - D_j)\end{aligned} \tag{40}$$

$$\begin{aligned}&\mathbb{E}[\Delta(\Theta(t))] + V\cdot\mathcal{T}_{min} \le B + V\cdot\mathcal{T}_{max}\\ &+ \sum_j \mathbb{E}[Q_j(t)|\Theta(t)]\mathbb{E}[(L_j^P(d^{input}(t), x^\star(t), y^\star(t))\\ &- D_j)] \le B + V\cdot\mathcal{T}_{max} - \epsilon\sum_j\mathbb{E}[Q_j(t)|\Theta(t)]\end{aligned} \tag{41}$$

$$\mathbb{E}[L(\Theta(t+1))] - \mathbb{E}[L(\Theta(t))] \le B' - \epsilon\sum_j\mathbb{E}[Q_j(t)|\Theta(t)] \tag{42}$$

$$\mathbb{E}[L(\Theta(T))] \le B'\cdot T - \epsilon\sum_{t=0}^{T-1}\sum_j\mathbb{E}[Q_j(t)|\Theta(t)] \le B'\cdot T. \tag{43}$$

Recall the definition of Lyapunov function that $L(\Theta(t)) = \frac{1}{2}\sum_j Q_j(t)^2$, we have $\frac{1}{2}\sum_j\mathbb{E}[Q_j(T)^2] \le B'\cdot T$.

According to Cauchy Inequality [35] $(\sum_i x_i y_i)^2 \le (\sum_i x_i^2)(\sum_i y_i^2)$, we have:

$$(\sum_j\mathbb{E}[Q_j(T)])^2 \le N_m\sum_j\mathbb{E}[Q_j(T)^2] \le 2N_m B'\cdot T. \tag{44}$$

Square on both sides and the limit is taken:

$$\lim_{T\to\infty}\frac{\sum_j \mathbb{E}[Q_j(T)]}{T} \le \lim_{T\to\infty}\sqrt{\frac{2N_m B'}{T}} = 0. \quad (45)$$

Note the fact that $Q_j(t) \ge 0$, so we come to the conclusion that:

$$\lim_{T\to\infty}\frac{\mathbb{E}[Q_j(T)]}{T} = 0, \forall M_j \in \mathbb{M}. \quad (46)$$

Therefore we can ignore C_7 in $\mathcal{P}^3$ and solve it directly using the DMCPA-GS algorithm and the theorem is proved. $\square$

Lemma 6. *The time average queue size of DMCPA-GS-Online is $O(V)$, determining convergence speed of queues' backlog.*

Proof. As for the time average queue size of DMCPA-GS-Online, recall Eq. 43 and we find:

$$\begin{aligned}\frac{1}{T}\sum_{t=0}^{T-1}\sum_j \mathbb{E}[Q_j(t)|\Theta(t)] &\le \frac{B'}{\epsilon} - \mathbb{E}[L(\Theta(T))] \\ &\le \frac{B + V(\mathcal{T}_{max} - \mathcal{T}_{min})}{\epsilon}.\end{aligned} \quad (47)$$

Since $\mathbb{E}[L(\Theta(T))] \ge 0$. Note V is the only changeable parameter which determines the size of queues' backlog and convergence speed. So the lemma is proved. $\square$

Consequently, how to solve $\mathcal{P}^3$ becomes the key of our online problem, which will be discussed in the following. It belongs to NP-Hard as well. Therefore, a novel algorithm based on Gibbs Sampling is considered to solve it.

C. The DMCPA-GS Algorithm for $\mathcal{P}^3$

The proposed algorithm DMCPA-GS based on Gibbs Sampling [36], [37] for $\mathcal{P}^3$ is shown in Alg. 2. First initialize y randomly each time. Then compute the optimal caching strategy x based on y, and the optimal target value $T(x, y)$ which can be abbreviated as T. Suppose there is another feasible tuple (x', y') and the optimal value is $T(x', y')$, which is abbreviated as T'. Then we can establish the following conditional migration probability from (x, y) to (x', y'):

$$\Pr(y'|y) = \frac{1}{1 + e^{(T'-T)/\omega}}, \quad (48)$$

where $\omega > 0$ as a control parameter. In the sampling process of DMCPA-GS, it is necessary to determine the neighbor state for sampling. In Alg. 2, M empty sets are first initialized. Then each time ϕ_i processors on the server are assigned to these M sets with a random number. The assigned quantity of a set can be 0. In this way, the search of neighbor state will become very random, and it is easy to jump out even if it falls into local optimization. Therefore, it performs well in searching the optimal solution in global.

In the following, we demonstrate the convergence of DMCPA-GS using Thm. 7.

Theorem 7. *With the decrease of ω, the probability of convergence to global optimal solution increases gradually. In particular, the algorithm converges to the global optimal value with a probability of close to 1 when $\omega \to 0$.*

Algorithm 2: The DMCPA-GS Algorithm for One Time Slot Problem $\mathcal{P}^3$

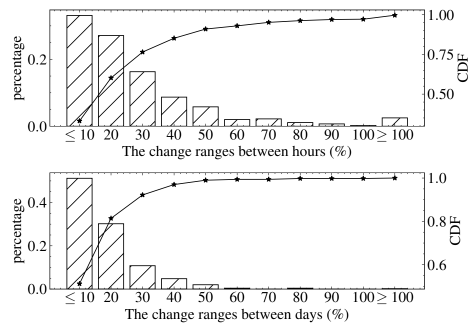
```
Input: $\omega, N, y^0, x(t-1), method$
Result: $x(t), y(t), T(t)$
$y^{cur} \leftarrow y^0$;
$x^{cur} \leftarrow \begin{cases} 1, & \text{if } y^{cur} \ge 1 \\ 0, & o.w. \end{cases}$;
$T^{cur} \leftarrow \mathcal{T}(x^{cur}, y^{cur})$;
for $S_i \in \mathbb{S}$ do
    /* N is the max iteration times. */
    for $n = 1$ to $N$ do
        repeat
            $j \leftarrow RandomInt(0, M)$;
            flip a coin with fair probability;
            if the coin comes up with heads then
                $y^{next}_{i,j} \leftarrow y^{next}_{i,j} + Rand(0, \phi_i - \sum_{j'} y_{i,j'})$;
            else
                $y^{next}_{i,j} \leftarrow y^{next}_{i,j} - Rand(0, y_{i,j})$;
            end
        until $y^{next}$ is feasible;
        $x^{next} \leftarrow \begin{cases} 1, & \text{if } y^{next} \ge 1 \\ 0, & o.w. \end{cases}$;
        $T^{next} \leftarrow \mathcal{T}(x^{next}, y^{next})$;
        $p_i \leftarrow \frac{1}{1+e^{(T^{next}-T^{cur})/\omega}}$;
        flip a coin with probability $p_i$ of heads;
        if the coin comes up with heads then
            $y^{cur} \leftarrow y^{next}$; $T^{cur} \leftarrow T^{next}$;
        end
    end
end
return $x^{cur}, y^{cur}, T^{cur}$;
```

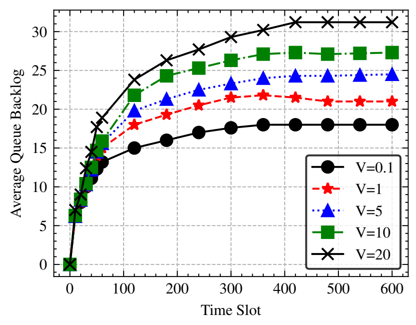
Proof. For each server S_i, suppose the overall decision space of processor allocation is $Y = \{y_1, \cdots, y_M\}$. In DMCPA-GS, the DNN model y_j is selected in each iteration and the decision space of y_j is a ϕ-dimension Markov Chain, where each dimension represents the number of processors are deployed on S_i for the model y_j. For the convenience of the following description, we only consider two DNN models, whose decision space is $Y = \{y_1, y_2\}$. During each iteration, due to the random selection of y_j jumping from the current allocation strategy to another, we have:

$$\begin{aligned}\Pr(\{y_1', y_2\}|\{y_1, y_2\}) &= \frac{e^{-T(\{y_1', y_2\})/\omega}}{e^{-T(\{y_1', y_2\})/\omega} + e^{-T(\{y_1, y_2\})/\omega}} \\ \Pr(\{y_1, y_2'\}|\{y_1, y_2\}) &= \frac{e^{-T(\{y_1, y_2'\})/\omega}}{e^{-T(\{y_1, y_2'\})/\omega} + e^{-T(\{y_1, y_2\})/\omega}}.\end{aligned} \quad (49)$$

We denote $\pi(\{y_1, y_2\})$ as the stable probability distribution of the allocation strategies $\{y_1, y_2\}$. According to the smooth distribution property of Markov Chain, we have

$$\begin{aligned}&\pi(\{y_1, y_2\}) \Pr(\{y_1', y_2'\}|\{y_1, y_2\}) \\ &= \pi(\{y_1', y_2'\}) \Pr(\{y_1, y_2\}|\{y_1', y_2'\}).\end{aligned} \quad (50)$$


(a) The probability variation.


(b) The impact of V.

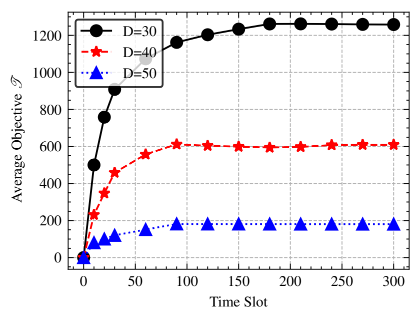

(c) The impact of D_j.

Fig. 4. The impact of Lyapunov parameters.

Substitute into Eq. 49 and we have

$$\begin{aligned}&\pi(\{y_1,y_2\})\times\frac{e^{-T(\{y_1^{'},y_2^{'}\})/\omega}}{e^{-T(\{y_1^{'},y_2^{'}\})/\omega}+e^{-T(\{y_1,y_2\})/\omega}}\\&=\pi(\{y_1^{'},y_2^{'}\})\times\frac{e^{-T(\{y_1,y_2\})/\omega}}{e^{-T(\{y_1^{'},y_2^{'}\})/\omega}+e^{-T(\{y_1,y_2\})/\omega}}\end{aligned}\tag{51}$$

According to the symmetry, we find that the equation holds when $\pi(\{y_1,y_2\})=\gamma e^{-T(\{y_1,y_2\})/\omega}$, where γ is a constant. Let Φ is the decision space of the processor allocation and $\sum_{\{y_1,y_2\}\in\Phi}\pi(\{y_1,y_2\})=1$ holds, the stable probability distribution of $\pi(\{y_1,y_2\})$ meets:

$$\begin{aligned}\pi(\{y_1,y_2\})&=\frac{e^{-T(\{y_1,y_2\})/\omega}}{\sum_{\{\hat{y_1},\hat{y_2}\}\in\Phi}e^{-T(\{\hat{y_1},\hat{y_2}\})/\omega}}\\&=\frac{1}{\sum_{\{\hat{y_1},\hat{y_2}\}\in\Phi}e^{(T(\{y_1,y_2\})-T(\{\hat{y_1},\hat{y_2}\}))/\omega}}\end{aligned}\tag{52}$$

Then for the global optimal strategy $\{y_1^*,y_2^*\}$, it is natural to have $T(\{y_1^*,y_2^*\})\le T(\{\hat{y_1},\hat{y_2}\})$ by the definition of optimality. Thus, we have $\pi(\{y_1^*,y_2^*\})\to 1$ when $\omega\to 0$. □

V. EVALUATION

In this section, we perform a series of experiments to verify the effectiveness of our algorithm.

A. Evaluation Settings

We set up a typical edge computing scenario where there are 5 edge servers, and for each edge server, a total of 10 types of DNN models can be deployed to these servers. We assume that there will be at least 200 users existing in this edge computing scenario and submitting tasks to the edge servers for "proximity inference". For the maximum inference delay model M_j is expected, D_j is set as 40 ms. As for the probability P, we assume that it follows the trace of a dataset from the real world. L_{acc} refers to the delay from each user to the nearest BS, which we consider to be 5 ms. The input data size of user d_k^{input} follows the Poisson distribution with an expectation of 100 mb and the ratio of computing consumption for one bit θ is 100. The bandwidth from the edge server to the cloud, which we consider to be 1000 mb/s in stability, and the delay to return the processing results from the cloud is about 100 ms. $c_{i,j}$ is the inference computing ability of a single processor on edge server S_i for model M_j and is assumed to be about 500 cycles/ms. In addition, we use ϕ_i to indicate the maximum number of processors that can be provided by server S_i, which is set as 20. W_i is the upper bound of the total size of DNN models that edge server S_i can handle, which is set as 300. In the meanwhile, the value of w_j follows the uniform distribution with expectation about 10 and variance about 3. As for the values of α and β, they can be determined by the service providers' preference, we set both to 0.5 by default.

B. Motivation Validation

As shown in Fig. 4(a), we study the variation of P collected from the dataset. One of our observations is that the value of P is more stable when the observation time span is longer. The horizontal axes of the two subgraphs depict the change ranges in hours and days respectively. The vertical axes depict the distribution of the change range. We can see that the cases with small change between days is more intensive by the comparison of the two subgraphs.

C. The Impact of Lyapunov Parameters

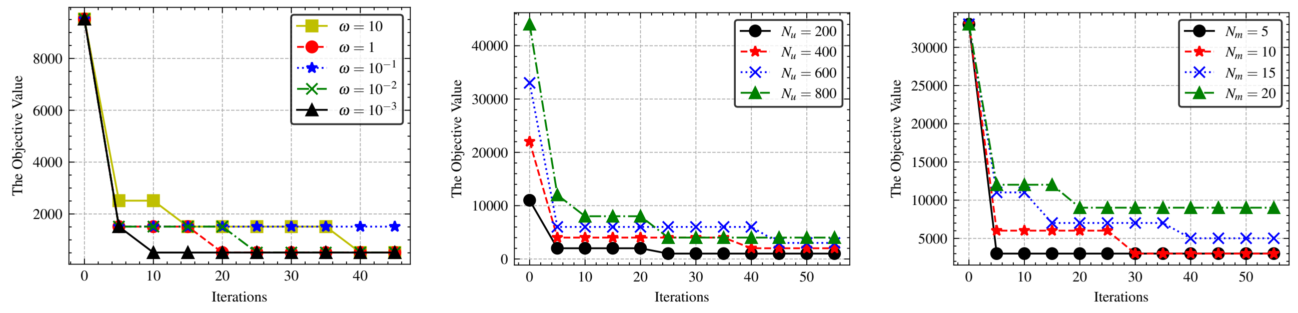
1) The Trade-off by Tuning V: Fig. 4(b) shows the role of the parameter V in DMCPA-GS-Online algorithm. We can find that after many iterations, the final algorithm will converge to a fixed value, which is controlled by V. When V is larger, the convergence speed of the algorithm is $O(V)$, so it needs more iteration rounds to converge, and the corresponding solution is closer to that of the original problem. On the contrary, when V is smaller, the algorithm converges faster and the delay constraint keeps more stable.

2) The Impact of the Model Delay Constraint D_j: Fig. 4(c) shows the convergence of the long-term optimization target $\mathcal{T}$ with different settings of D_j. For this purpose, we set three different values 30, 40 and 50 as D_j. We find that the convergence value of $\mathcal{T}$ is related to D_j. With the smaller the value of D, the tolerable inference delay is smaller, which results in the higher the delay penalty, and so the larger $\mathcal{T}$. On the contrary, the greater the value of D_j, the delay penalty is smaller, even reaching 0 if D_j is large enough.

D. The Impact of Problem Scale for DMCPA-GS

In this part, we validate the convergence of DMCPA-GS and then explore the effect of different problem scales on the convergence speed.

In Theorem. 7, DMCPA-GS Algorithm converges to the optimal value with probability close to 1 as $w\to 0$. Therefore,



(a) The impact of ω.
(b) The impact of user number.
(c) The impact of model number.

Fig. 5. The impact of different parameters for DMCPA-GS.

we investigate the effect of different values of w on the convergence in this experiment part. As shown in Fig. 5(a), we can find that as the number of iterations increases, the objective value is approaching the optimal value. And the smaller w brings faster convergence speed. In other words, it takes less iterations times to converge to the optimal value. We also note that if w is small, the convergence speed may be different, but eventually will converge to the optimal value. However, if w is too large (greater than 1), the algorithm will be too difficult to converge, which means the objective value will oscillate continuously. Therefore we verify the correctness of the theorem.

By setting different values for the number of users, we can see from Fig. 5(a) that different convergence curves are obtained by setting w to 200, 400, 600 and 800 respectively. The default value of w is 0.001. When the number of users increases, we can see that it takes more iteration times to converge relatively. However, since Gibbs Sampling algorithm belongs to random algorithms, the initial state and random neighborhood state selection will affect the convergence procedure with a certain probability of faster or slower convergence. Different convergence trajectories are obtained for each run, and only a certain run of the convergence procedure is shown in Fig. 5(b). Based on a similar method of experimental setting, we design experiments with different number of types of DNN models, which are 5,10,15,20 respectively in Fig.5(c). In these cases, the convergence of Gibbs Sampling algorithm with iteration times is studied. We can see that with the scale of the problem increases, the convergence speed will slow down, but it will converge eventually. In practical scenarios, the setting of the number of iteration times will affect the final results of the algorithm, which includes the optimality of the solution and the running time of the algorithm.

E. Comparision of Different Algorithms

Fig. 6 compares our proposed algorithm DMCPA-GS-Online with three different baselines. These baseline algorithms are described as follows:

- **Cloud-Only:** At each time, only consider forwarding all DNN inference requests to cloud and edge does not undertake any inference tasks.
- **Edge-Average:** Each time, each server selects as many acceptable DNN models as possible, and evenly allocates processors to these models.
- **Edge-Random:** Each time, several DNN models are randomly selected on each server and deployed on the edge side, and the random number of processors is allocated.

From Fig. 6(a) and Fig. 6(b), we can see that DMCPA-GS-Online is the best compared with the other three baseline algorithms under different w settings. Cloud-Only performs the worst due to its large transmission delay. Edge-Random and Edge-Average perform almost the same, but worse than DMCPA-GS-Online because they do not consider the difference in computing requirements of user request distribution. From Fig. 6(c), we can see that with the increase of cloud-edge bandwidth B_i^{cloud}, the long-term optimization target $\mathcal{T}$ also decreases. Specially, Cloud-Only is most affected by this parameter, while other algorithms are less.

VI. Conclusion

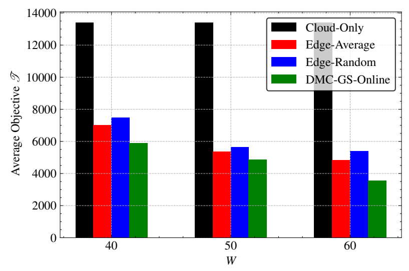
Nowadays more and more applications leverage DNN models for better service. In edge computing environments, caching models on edge servers greatly enhances the user QoS experience with benefits of efficiency, privacy, reliability and security. In this paper, we consider the DNN Model Caching and Processor Allocation (DMCPA) problem. Under limited resources of edge and model guaranteed delays, our goal is to minimize the user perception delay and energy consumption with careful model caching and processor allocation strategy. We formulate it as an Integer Nonlinear Program (INLP) and proves its NP-Completeness. The novel online algorithm DMCPA-GS-Online is proposed without future information and the theoretical analysis is given. Experiments based on trace dataset from the real world demonstrate our algorithm outperforms other baselines.

Acknowledgments

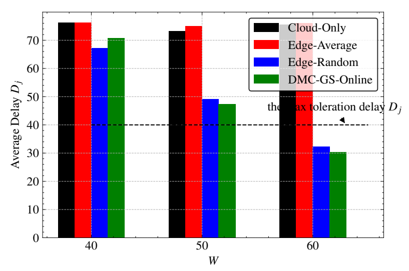
This work was supported in part by NSFC (61872175, 61832008), and Collaborative Innovation Center of Novel Software Technology and Industrialization.

References

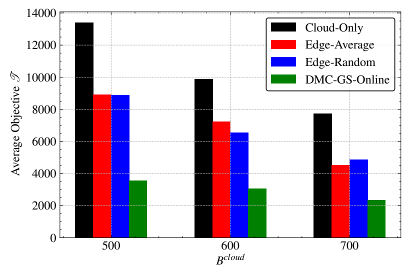
[1] X. Foukas, G. Patounas, A. Elmokashfi, and M. K. Marina, “Network slicing in 5g: Survey and challenges,” *IEEE Communications Magazine*, vol. 55, no. 5, pp. 94–100, 2017.
[2] Y. Lu, “Artificial intelligence: a survey on evolution, models, applications and future trends,” *Journal of Management Analytics*, vol. 6, no. 1, pp. 1–29, 2019.
[3] M. Jelassi, C. Ghazel, and L. A. Saïdane, “A survey on quality of service in cloud computing,” in *2017 3rd International Conference on Frontiers of Signal Processing (ICFSP)*. IEEE, 2017, pp. 63–67.



(a) Average Objective $\mathcal{T}$.



(b) Average Delay D_j.



(c) Bandwidth B_i^{cloud}.

Fig. 6. The performance of different algorithms.

[4] W. Yu, F. Liang, X. He, W. G. Hatcher, C. Lu, J. Lin, and X. Yang, "A survey on the edge computing for the internet of things," *IEEE access*, vol. 6, pp. 6900–6919, 2017.

[5] W. Shi and S. Dustdar, "The promise of edge computing," *Computer*, vol. 49, no. 5, pp. 78–81, 2016.

[6] W. Z. Khan, E. Ahmed, S. Hakak, I. Yaqoob, and A. Ahmed, "Edge computing: A survey," *Future Generation Computer Systems*, vol. 97, pp. 219–235, 2019.

[7] E. Li, L. Zeng, Z. Zhou, and X. Chen, "Edge ai: On-demand accelerating deep neural network inference via edge computing," *IEEE Transactions on Wireless Communications*, vol. 19, no. 1, pp. 447–457, 2019.

[8] J. Chen and X. Ran, "Deep learning with edge computing: A review." *Proc. IEEE*, vol. 107, no. 8, pp. 1655–1674, 2019.

[9] A. Marchisio, M. A. Hanif, F. Khalid, G. Plastiras, C. Kyrkou, T. Theocharides, and M. Shafique, "Deep learning for edge computing: Current trends, cross-layer optimizations, and open research challenges," in *2019 IEEE Computer Society Annual Symposium on VLSI (ISVLSI)*. IEEE, 2019, pp. 553–559.

[10] C. Shi, L. Chen, C. Shen, L. Song, and J. Xu, "Privacy-aware edge computing based on adaptive dnn partitioning," in *2019 IEEE Global Communications Conference (GLOBECOM)*. IEEE, 2019, pp. 1–6.

[11] A. Shrestha and A. Mahmood, "Review of deep learning algorithms and architectures," *IEEE Access*, vol. 7, pp. 53 040–53 065, 2019.

[12] 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, pp. 1738–1762, 2019.

[13] H. Li, C. Hu, J. Jiang, Z. Wang, Y. Wen, and W. Zhu, "Jalad: Joint accuracy-and latency-aware deep structure decoupling for edge-cloud execution," in *2018 IEEE 24th International Conference on Parallel and Distributed Systems (ICPADS)*. IEEE, 2018, pp. 671–678.

[14] H.-J. Jeong, H.-J. Lee, C. H. Shin, and S.-M. Moon, "Ionn: Incremental offloading of neural network computations from mobile devices to edge servers," in *Proceedings of the ACM Symposium on Cloud Computing*, 2018, pp. 401–411.

[15] S. Zhang, S. Zhang, Z. Qian, J. Wu, Y. Jin, and S. Lu, "Deepslicing: Collaborative and adaptive cnn inference with low latency," *IEEE Transactions on Parallel and Distributed Systems*, vol. 32, no. 9, pp. 2175–2187, 2021.

[16] S. Yao, J. Li, D. Liu, T. Wang, S. Liu, H. Shao, and T. Abdelzaher, "Deep compressive offloading: Speeding up neural network inference by trading edge computation for network latency," in *Proceedings of the 18th Conference on Embedded Networked Sensor Systems*, 2020, pp. 476–488.

[17] S. Wang, J. Xu, N. Zhang, and Y. Liu, "A survey on service migration in mobile edge computing," *IEEE Access*, vol. 6, pp. 23 511–23 528, 2018.

[18] L. Wang, L. Jiao, T. He, J. Li, and H. Bal, "Service placement for collaborative edge applications," *IEEE/ACM Transactions on Networking*, vol. 29, no. 1, pp. 34–47, 2020.

[19] S. Pasteris, S. Wang, M. Herbster, and T. He, "Service placement with provable guarantees in heterogeneous edge computing systems," in *IEEE INFOCOM 2019-IEEE Conference on Computer Communications*. IEEE, 2019, pp. 514–522.

[20] 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 2019-IEEE Conference on Computer Communications*. IEEE, 2019, pp. 1459–1467.

[21] 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 38th International Conference on Distributed Computing Systems (ICDCS)*. IEEE, 2018, pp. 365–375.

[22] K. Poularakis, J. Llorca, A. M. Tulino, I. Taylor, and L. Tassiulas, "Joint service placement and request routing in multi-cell mobile edge computing networks," in *IEEE INFOCOM 2019-IEEE Conference on Computer Communications*. IEEE, 2019, pp. 10–18.

[23] V. Farhadi, F. Mehmeti, T. He, T. F. La 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*, vol. 29, no. 2, pp. 779–792, 2021.

[24] Z. Ma, S. Zhang, Z. Chen, T. Han, Z. Qian, M. Xiao, N. Chen, J. Wu, and S. Lu, "Towards revenue-driven multi-user online task offloading in edge computing," *IEEE Transactions on Parallel and Distributed Systems*, vol. 33, no. 5, pp. 1185–1198, 2021.

[25] Z. Chen, S. Zhang, C. Wang, Z. Qian, M. Xiao, J. Wu, and I. Jawhar, "A novel algorithm for nfv chain placement in edge computing environments," in *2018 IEEE Global Communications Conference (GLOBECOM)*. IEEE, 2018, pp. 1–6.

[26] Y. Chen, S. Zhang, M. Xiao, Z. Qian, J. Wu, and S. Lu, "Multi-user edge-assisted video analytics task offloading game based on deep reinforcement learning," in *2020 IEEE 26th International Conference on Parallel and Distributed Systems (ICPADS)*, 2020, pp. 266–273.

[27] Y. Ma, W. Liang, and S. Guo, "Mobility-aware delay-sensitive service provisioning for mobile edge computing," in *IEEE INFOCOM 2019-IEEE Conference on Computer Communications Workshops (INFOCOM WKSHPS)*. IEEE, 2019, pp. 270–276.

[28] T. X. Tran, K. Chan, and D. Pompili, "Costa: Cost-aware service caching and task offloading assignment in mobile-edge computing," in *2019 16th Annual IEEE International Conference on Sensing, Communication, and Networking (SECON)*. IEEE, 2019, pp. 1–9.

[29] Z. Xu, S. Wang, S. Liu, H. Dai, Q. Xia, W. Liang, and G. Wu, "Learning for exception: Dynamic service caching in 5g-enabled mecs with bursty user demands," in *2020 IEEE 40th International Conference on Distributed Computing Systems (ICDCS)*. IEEE, 2020, pp. 1079–1089.

[30] 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, no. 10, pp. 2333–2345, 2018.

[31] A. Noulas, S. Scellato, N. Lathia, and C. Mascolo, "A random walk around the city: New venue recommendation in location-based social networks," in *2012 International Conference on Privacy, Security, Risk and Trust and 2012 International Confernece on Social Computing*. Ieee, 2012, pp. 144–153.

[32] X. Xia, F. Chen, Q. He, J. C. Grundy, M. Abdelrazek, and H. Jin, "Cost-effective app data distribution in edge computing," *IEEE Transactions on Parallel and Distributed Systems*, vol. 32, no. 1, pp. 31–44, 2020.

[33] D. Pisinger and P. Toth, "Knapsack problems," in *Handbook of combinatorial optimization*. Springer, 1998, pp. 299–428.

[34] C. Wang, S. Zhang, Y. Chen, Z. Qian, J. Wu, and M. Xiao, "Joint configuration adaptation and bandwidth allocation for edge-based real-time video analytics," in *IEEE INFOCOM 2020-IEEE Conference on Computer Communications*. IEEE, 2020, pp. 257–266.

[35] G. Buskes and A. Van Rooij, "Almost f-algebras: commutativity and the cauchy-schwarz inequality," *Positivity*, vol. 4, no. 3, p. 227, 2000.

[36] A. Gelman, J. B. Carlin, H. S. Stern, and D. B. Rubin, *Bayesian data analysis*. Chapman and Hall/CRC, 1995.

[37] C. P. Robert, G. Casella, and G. Casella, *Monte Carlo statistical methods*. Springer, 2004, vol. 2.