Joint Optimization of DNN Model Caching and Request Routing in Mobile Edge Computing

Shuting Qiu, Fang Dong, *Member, IEEE,* Siyu Tan, Ruiting Zhou, *Member, IEEE,* Dian Shen, *Member, IEEE,* Patrick P. C. Lee, *Senior Member, IEEE* and Qilin Fan, *Member, IEEE,*

***Abstract*— Mobile edge computing (MEC) can pre-cache deep neural networks (DNNs) near end-users, providing low-latency services and improving users' quality of experience (QoE). However, caching all DNN models at edge servers with limited capacity is difficult, and the impact of model loading time on QoE remains underexplored. Hence, we introduce dynamic DNNs in edge scenarios, disassembling a complete DNN model into interrelated submodels for more fine-grained and flexible model caching and request routing solutions. This raises the pressing issue of jointly deciding request routing and submodel caching for dynamic DNNs to balance model inference precision and loading latency for QoE optimization. In this paper, we study the joint dynamic model caching and request routing problem in MEC networks, aiming to maximize user request inference precision under constraints of server resources, latency, and model loading time. To tackle this problem, we propose CoCaR, an offline algorithm based on linear programming and random rounding that leverages dynamic DNNs to optimize caching and routing schemes, achieving near-optimal performance. Furthermore, we develop an online variant of CoCaR, named CoCaR-OL, enabling effective adaptation to dynamic and unpredictable online request patterns. The simulation results demonstrate that the proposed CoCaR improves the average inference precision of user requests by 46% compared to state-of-the-art baselines. In addition, in online scenarios, CoCaR-OL achieves an improvement of no less than 32.3% in user QoE over competitive baselines.**

***Index Terms*—Mobile edge computing, dynamic DNN, model caching, request routing, joint optimization**

I. Introduction

RECENTLY, machine learning methods, especially deep neural networks (DNNs), have been transforming various application domains [2]–[4], such as computer vision [5], speech recognition [6], and autonomous driving [7]. To promote efficient task processing, mobile edge computing (MEC) [8]–[11], has emerged as a promising paradigm to provide end-users with low-latency computation, caching, and transmission capabilities [12]–[14]. In MEC, DNN models can be pre-cached at base stations (BSs) with edge servers, enabling rapid responses to user requests [15], [16]. However, unlike cloud servers, due to the limited resources of a BS, only a few popular DNN models can be cached simultaneously to service some users [17], [18]. Additionally, request routing is also a key factor affecting users' quality of experience (QoE) in multi-edge collaboration scenarios [19]–[21]. Therefore, routing requests to BSs with relevant cached models is essential for maximizing user coverage and system efficiency. Obviously, the model caching mechanism is tightly coupled with the request routing scheme. Thus, jointly optimizing model caching and request routing decisions is crucial for improving users' QoE in MEC.

Existing joint optimization schemes for model caching and request routing mainly focus on caching complete models at resource-limited BSs [22]–[26], leading to inefficient resource utilization and poor user QoE [27]. Fortunately, dynamic DNNs [28], [29] provide a promising solution by enabling flexible model structure adjustments. In the context of this paper, we define dynamic DNNs as a model versioning framework, where a base model is partitioned into multiple submodels of varying depth and precision for fine-grained caching. Specifically, we disassemble a complete DNN model into a set of interrelated submodels by dividing the feature extraction layer, allowing for switches between them by adjusting the number of layers. This approach enables fine-grained resource allocation in a multi-server caching environment, thereby distinguishing our method from device-edge split computing [30], [31], which partitions a single inference task across two locations for collaborative execution. Consequently, our use of dynamic DNNs allows for more flexible model caching and request routing solutions, utilizing BS resources in a more fine-grained manner.

However, incorporating dynamic DNNs complicates the joint optimization decisions, leading to two primary challenges: 1) Caching schemes for submodels of the same dynamic DNN at a BS are mutually exclusive, permitting only one submodel to be cached at a time. This necessitates determining not only which DNN models to cache but also which submodel of these dynamic DNNs to cache, thereby increasing the complexity of the variable-coupled joint optimization problem. 2) Inference requests for different DNN

This work is supported by National Natural Science Foundation of China under Grants, No. 62232004, Jiangsu Provincial Frontier Technology Research and Development Program under Grant BF2024070, Shenzhen Science and Technology Program under Grant KJZD20240903100814018, Jiangsu Provincial Key Laboratory of Network and Information Security under Grants No.BM2003201, Key Laboratory of Computer Network and Information Integration of Ministry of Education of China under Grants No.93K-9, and partially supported by Collaborative Innovation Center of Novel Software Technology and Industrialization. (Corresponding authors: Fang Dong.)

This paper is an extended version of our preliminary work presented at IEEE INFOCOM 2025 [DOI: 10.1109/INFOCOM55648.2025.11044457].

Shuting Qiu, Fang Dong, Siyu Tan, Dian Shen and Ruiting Zhou are with the School of Computer Science and Engineering, Southeast University, Nanjing 211189, China (email: qiushuting@seu.edu.cn; fdong@seu.edu.cn; sytan@seu.edu.cn; dshen@seu.edu.cn; ruitingzhou@seu.edu.cn).

Patrick P. C. Lee is with the Department of Computer Science and Engineering, The Chinese University of Hong Kong, Hong Kong (email: pclee@cse.cuhk.edu.hk).

Qilin Fan is with the School of Big Data and Software Engineering, Chongqing University, Chongqing 400044, China (email: fanqilin@cqu.edu.cn).

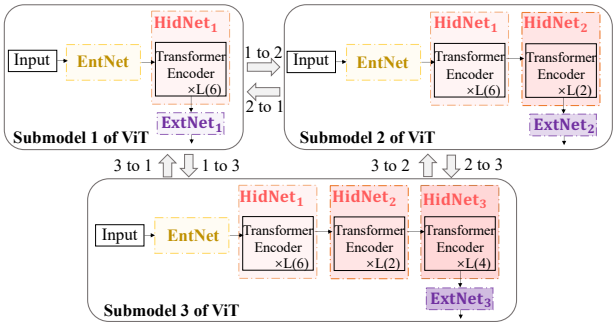

Fig. 1. Submodels division and switching of ViT. When switching from submodel 1 to submodel 2 of ViT, we only need to remove ExtNet_1 of submodel 1 and then connect HidNet_2 and ExtNet_2 to form submodel 2.

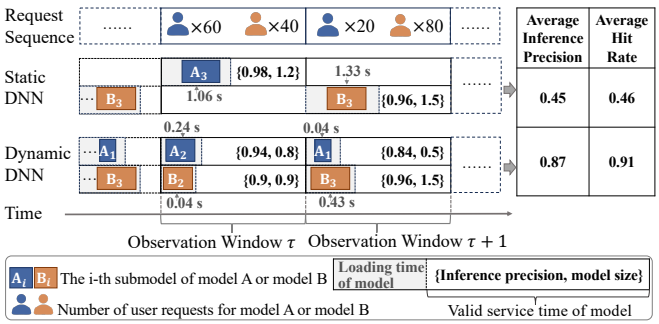

Fig. 2. Examples of static DNN and dynamic DNN schemes.

models arrive in real-time, requiring timely adjustments to caching schemes [32]. Moreover, caching models at BSs incurs loading latency [33], and different caching schemes for dynamic DNNs yield different model valid service times and inference precisions, thereby affecting users' QoE. Therefore, the adverse impact of model loading time on QoE must be considered and minimized.

Although some related studies consider model loading time [33], [34] and explore its impact on the joint optimization of model caching and request routing problem in multi-edge collaborative MEC to maximize throughput under latency and accuracy constraints, they only use model loading costs to refine the solution, without considering its impact on decision-making. Therefore, existing solutions are not efficient and flexible enough in MEC environments with limited resources and dynamically changing user requests.

In this paper, we study a **J**oint **D**ynamic model **C**aching and request **R**outing problem (JDCR) with the goal of maximizing user request inference precision. Based on linear programming (LP) and randomized rounding, we first propose a novel offline multi-edge **Co**llaborative dynamic model **Ca**ching and request **R**outing optimization algorithm (CoCaR), which provides an approximate optimal solution to the JDCR problem with theoretical guarantees. Subsequently, we extend CoCaR to CoCaR-OL to adapt to online scenarios where user requests are hard to predict. Specifically, we divide time into consecutive observation windows. When making decisions in each window, we first consider the BS resource and latency constraints, as well as the impact of model loading time on QoE due to the switching of caching schemes. Then, based on BS caching results from the previous window, we utilize the capability of rapid switching between submodels of the same dynamic DNN to flexibly adjust the caching scheme, thereby increasing the valid service time of the model. Additionally, by leveraging the fine-grained resource utilization feature of dynamic DNNs, BS resources can be used more efficiently, enabling the caching of more DNN models to satisfy a greater number of user demands. The contributions of this paper are as follows:

- We formulate the JDCR problem in MEC networks. The goal is to maximize the total inference precision for user requests while adhering to constraints on BS resources, latency, and model loading time. As this JDCR problem is a non-convex, nonlinear integer programming problem that poses significant challenges for resolution, we perform equivalent transformations and relaxations.
- We propose a novel offline algorithm, CoCaR, based on LP and random rounding to solve the JDCR problem. Specifically, CoCaR can leverage dynamic DNNs to provide a more flexible and fine-grained caching scheme. Theoretical analyses show that CoCaR has strict performance guarantees and achieves an approximation ratio of $(1-\sqrt{\frac{4\ln|\mathcal{H}|}{\mathbb{P}^\dagger}})^2$ to the optimal integer solution, where $|\mathcal{H}|$ represents the total number of submodels, and $\mathbb{P}^\dagger$ is the objective value of the optimal fractional solution.
- We further extend CoCaR to handle online scenarios where future user requests are difficult to predict in advance and both caching and routing decisions must be made in real-time. Thus, we propose CoCaR-OL, an online variant that determines dynamic DNN model caching strategies based on expected future gains estimated from historical user request patterns. It further adopts a greedy algorithm for routing decisions, enabling effective adaptation to dynamic and unpredictable demands.
- Extensive simulations validate the effectiveness of the proposed CoCaR and CoCaR-OL. The results show that CoCaR achieves at least a 46% improvement in average inference precision and a 44% increase in cache hit rate over four state-of-the-art algorithms, utilizing at least 86% of BS resources. In online scenarios, CoCaR-OL attains at least a 1.71× improvement in users' QoE and a 1.73× increase in cache hit rate over the baseline.

II. Background and Motivation

Dynamic DNN. As shown in Fig. 1, taking Vision Transformer (ViT) [35] as an example, we divide a DNN into three parts based on its network architecture: the entry network (EntNet), the hidden network (HidNet), and the exit network (ExtNet). To ensure a certain level of inference precision, the DNN is divided into multiple submodels of different sizes by partitioning HidNet, and each submodel has its own trained ExtNet. Additionally, switching between submodels of a dynamic DNN involves adding or removing the corresponding HidNet and ExtNet.

Next, we use an example to compare a scheme that caches only complete models (i.e., static DNN) with a scheme based

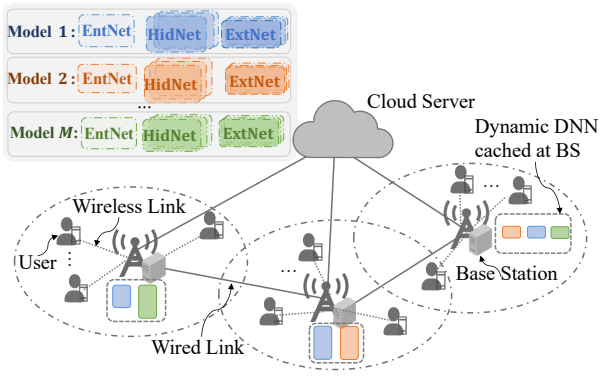

Fig. 3. System model. The same color indicates the dynamic DNN associated with a model type, while different lengths of that color represent different submodels of the dynamic DNN.

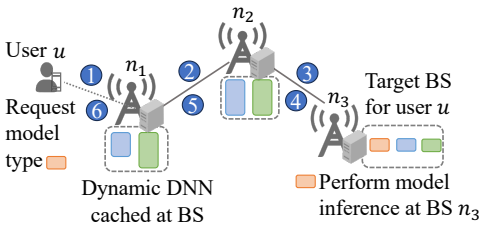

Fig. 4. Communication latency: Routing user u's request involves wireless transmission from u to home BS n_1, wired transmission from n_1 to target BS n_3, and a total of 6 hops from initiating the request to receive the inference result.

on a dynamic DNN at a BS. Apart from the specific experimental settings described below, the remaining parameters and performance results are obtained under the experimental environment and configurations described in Sec. VI-A.

Motivating Example. As shown in Fig. 2, there are two DNN model types, A and B, each divided into three submodels of different sizes. A period of 50s is divided into 10 observation windows, with 100 users randomly and uniformly initiating model inference requests in each window. The cache size is set to 2, with cached models loaded into the BS memory at the start of each window, and providing valid services once fully cached. In the observation window $\tau+1$, the static DNN scheme takes 1.33s to fully cache model B (size 1.5), but cannot fully cache model A (size 1.2) due to the limited cache size at the same time. In contrast, the dynamic DNN-based scheme utilizes cache results from a window τ, taking only 0.43s to cache model B from submodel 2 to submodel 3, providing 4.57s of valid service. Additionally, it can also cache submodel 1 of model A, achieving an inference precision of 0.84, thereby providing more services to more users. Ultimately, using a dynamic DNN improves the average inference precision and cache hit rate by at least 48% compared to a static DNN. This example underscores the significance of developing novel dynamic DNN-based model caching and request routing schemes.

III. System Model and Problem Formulation

A. System Overview

As shown in Fig. 3, we consider an MEC system consisting of $\mathcal{N}=\{1,2,\cdots,N\}$ BSs equipped with edge servers. These BSs have caching and computing capabilities and can communicate with each other through high-speed wired networks. They can provide some of the M services for end users $\mathcal{U}=\{1,2,\cdots,U\}$. We assume that users are randomly and uniformly distributed within the BS coverage area, with the closest BS to each user u (called the home BS) denoted as $\hat{n}_u$. We focus on joint model caching and request routing in MEC networks. Let $\mathcal{M}=\{1,2,\cdots,M\}$ represent the set of different DNN model types, and $m_u\in\mathcal{M}$ the DNN model inference request generated by user u. Let $\mathcal{H}(m)=\{h_0^m,h_1^m,\cdots,h_{H(m)}^m\}$ denote the set of submodels $\{h_j^m\}_{j=1}^{H(m)}$ of the dynamic DNN associated with m, along with an empty submodel h_0^m [1]. Then, denote by $\mathcal{H}=\bigcup_{m\in\mathcal{M}}\mathcal{H}(m)$ the set of all submodels, and $|\mathcal{H}|$ the total number of submodels. For each model type m, a BS can cache at most one submodel concurrently to provide service. Furthermore, we define a partial order $(\mathcal{H},\preceq)$ over the set of submodels $\mathcal{H}$, where each $\mathcal{H}(m)$ is linearly ordered, and elements from different $\mathcal{H}(m)$ are incomparable. That is, for any $h_i,h_j\in\mathcal{H}$, we have $h_i\preceq h_j$ if and only if $h_i,h_j\in\mathcal{H}(m)$, and h_i is a submodel no larger than h_j. Similarly, $h_i\succ h_j$ holds if and only if h_i and h_j belong to the same $\mathcal{H}(m)$, and h_i is a larger submodel than h_j.

Considering the dynamic nature of the MEC network, time is divided into discrete time slots, denoted as $\mathcal{T}=\{1,2,\ldots,t,\ldots,T\}$. Multiple consecutive time slots form an observation window, denoted as $\Gamma=\{1,2,\ldots,\tau,\ldots,|\Gamma|\}$. Each observation window updates the caching and routing decisions based on user requests. If user u's request is routed to the target BS that caches the submodel of the dynamic DNN of m_u, it is called a hit, and model inference is performed to obtain a result with corresponding precision; otherwise, it is a miss, and the user u's request will be sent to the cloud with an inference precision of 0 at the edge.

Next, we focus on the joint offline caching and routing decisions within an observation window τ. For brevity, we omit τ when there is no ambiguity.

B. Model Caching Model

It is crucial to decide which DNN models to cache on resource-limited BSs. Let binary variable $x_{n,h}\in\{0,1\}$ be the model caching decision for submodel h in BS n at window τ, where $x_{nh}=1$ if submodel h has been cached at BS n, and $x_{n,h}=0$ otherwise. Each BS is required to cache a submodel of the dynamic DNN associated with each model type:

$$\sum_{h\in\mathcal{H}(m)} x_{n,h}=1,\forall n\in\mathcal{N},m\in\mathcal{M}, \tag{1}$$

where if $x_{n,h_0^m}=1$, it indicates that the empty submodel of model type m is cached (which is equivalent to not caching the model type m).

[1]Here, h_0^m is introduced for modeling convenience, with no impact on BS resource usage or model inference precision.

In addition, the total cached models cannot exceed the memory capacity of BS n:

$$\sum_{h\in\mathcal{H}} x_{n,h}\cdot r_h \le R_n, \forall n\in\mathcal{N}, \tag{2}$$

where r_h is the required memory space to cache submodel h, and R_n is the total memory capacity of BS n.

C. Request Routing Model

In general, there are many methods to pre-fetch user requests [23], [36], which can then be routed to the target BS or the cloud for inference. Let $y_{n,u}\in\{0,1\}$ be the user request routing decision variable at window τ, where $y_{n,u}=1$ if the request from user u is routed to BS n, otherwise $y_{n,u}=0$. Let d_u, ddl_u, and s_u denote the request data size of user u, the maximum perceived latency user u can tolerate, and the initiation time of user u's request in the observation window τ, respectively.

First, each user u's request is routed to at most one BS, and requests that cannot be routed are sent to the cloud:

$$\sum_{n\in\mathcal{N}} y_{n,u}\le 1, \forall u\in\mathcal{U}. \tag{3}$$

Second, users experience end-to-end inference latency from request initiation to receiving the result, which includes:

- *Communication latency*. As shown in Fig. 4, the communication latency for routing user requests can be calculated as $T_u^{\text{off}}=\sum_{n\in\mathcal{N}} y_{n,u}\cdot(\frac{d_u}{\varphi_{\hat{n}_u}}+\frac{d_u}{r_{\hat{n}_u,n}}+\lambda_{u,n}), \forall u\in\mathcal{U}$, where $\varphi_{\hat{n}_u}=W_c log_2(1+\frac{g_u E_{n,u}}{\mathbb{N}_0})$ is the wireless transmission rate, g_u is user u's transmission power, $E_{n,u}$ is the channel gain from user u to BS n, W_c is the channel bandwidth, and $\mathbb{N}_0$ is the noise power [2]; $r_{\hat{n}_u,n}$ is the average wired transmission rate between home BS $\hat{n}_u$ and target BS n; and $\lambda_{u,n}$ denotes the propagation latency of user u's request routed to BS n, measured from its initiation to the reception of the inference result, which depends on the number of hops.
- *Inference latency*. The inference latency of user u's request at the target BS n can be calculated as $T_u^{\text{infer}}=\sum_{n\in\mathcal{N}}\sum_{h\in\mathcal{H}(m_u)} y_{n,u}\cdot x_{n,h}\cdot\frac{c_h\cdot d_u}{C_n}, \forall u\in\mathcal{U}$, where c_h denotes the computational flops required for model h to process one unit of data request, and C_n denotes the maximum computational capacity that BS n can provide.

Thus, user u's end-to-end inference latency $\mathbb{T}_u$ can be calculated as $\mathbb{T}_u=T_u^{\text{off}}+T_u^{\text{infer}}, \forall u\in\mathcal{U}$. To ensure the QoE of end-users, $\mathbb{T}_u$ must not exceed the maximum perceived latency ddl_u that user u can tolerate:

$$\mathbb{T}_u\le ddl_u, \forall u\in\mathcal{U}. \tag{4}$$

Additionally, the time required to load the DNN model into the BS's memory at each observation window cannot be ignored. In offline scenarios, we can proactively pre-download the required DNN models from the cloud to the secondary storage of the BS based on the expected user requests in the upcoming window, and then load the models into the memory at the beginning of the next observation window. Let $T_{n,m}^{\text{load}}$ denote the loading latency for caching the submodel of the dynamic DNN associated with model type m at BS n. When caching submodel h of a model type that is already cached in the previous window, let $D_m^{swit}(h',h)$ denote the switching latency from submodel h' to submodel h; conversely, when caching a model that has not been cached before, let $D_m^{new}(h_0^m,h)$ represent the latency to add submodel h of model m. For the observation window τ, the model loading latency of the model type m at BS n can be calculated as:

$$\begin{aligned} T_{n,m}^{\text{load}} &= \sum_{h,h'\in\{\mathcal{H}(m)\setminus h_0^m\}} x_{n,h'}(\tau-1)\cdot D_m^{swit}(h',h)\cdot x_{n,h} \\ &\quad+\sum_{h\in\{\mathcal{H}(m)\setminus h_0^m\}} x_{n,h}\cdot D_m^{new}(h_0^m,h)\cdot x_{n,h_0^m}(\tau-1) \\ &= \mathbf{x}_{n,m}^{\mathrm{T}}\cdot D_m\cdot\mathbf{x}'_{n,m}(\tau-1), \end{aligned} \tag{5}$$

where vector $\mathbf{x}_{n,m}=[x_{n,h_0^m},x_{n,h_1^m},\cdots,x_{n,h_{H(m)}^m}]$, and D_m can be regarded as the transit time of model m, including the possible switching or addition of the model state during the transit process from the previous window to the current window.

Since DNNs must be cached before providing services, user u's request can be routed to BS n with the cached submodel of m_u for inference only if the model loading completion time is earlier than the request initiation time s_u:

$$\sum_{n\in\mathcal{N}} y_{n,u}\cdot T_{n,m_u}^{\text{load}}\le s_u, \forall u\in\mathcal{U}. \tag{6}$$

TABLE I
LIST OF NOTATIONS.

Notation	Description
$\mathcal{N},\mathcal{U}$	Set of base stations and users
$\mathcal{M},\mathcal{H}$	Set of DNN model types and submodels
$\mathcal{H}(m)$	Set of submodels associated with model type m
$\hat{n}_u$	Closest BS to user u (home BS)
m_u	User u's requested DNN model type m
$x_{n,h}$	Caching variable for submodel h at BS n in window τ
$y_{n,u}$	Routing variable for user u to BS n in window τ
d_u, ddl_u	User u's data size and maximum tolerable latency
s_u	User u's request initiation time in window τ
R_n	Memory capacity of BS n
C_n	Computation capacity of BS n
r_h	Size of submodel h
c_h	Flops of submodel h per data unit
φ_n	Wireless transmission rate of BS n
$r_{n',n}$	Wired transmission rate between BS n' and BS n
$\lambda_{u,n}$	Propagation latency for user u to BS n in window τ
T_u^{off}	User u's communication latency in window τ
T_u^{infer}	User u's inference latency in window τ
$\mathbb{T}_u$	User u's total end-to-end inference latency in window τ
$T_{n,m}^{\text{load}}$	Load latency of model type m at BS n in window τ
p_h	Expected inference precision of submodel h

[2]In this paper, we simplify the transmission model by assuming that BSs adopt a fixed transmission rate $\varphi_{\hat{n}_u}$ for all users u [37].

D. Problem Formulation

In the offline scenario where user requests can be predicted in advance, the purpose is to jointly decide on model caching and request routing variables in a multi-edge collaborative scenario to maximize the total inference precision of all user requests in each observation window, while the constraints associated with the decision variables are not violated. Let p_h be the expected inference precision of the submodel h. The problem can be formulated as follows:

$$(\mathcal{P}) \max_{x,y} \sum_{u\in\mathcal{U}} \sum_{n\in\mathcal{N}} \sum_{h\in\mathcal{H}(m_u)} x_{n,h}\cdot y_{n,u}\cdot p_h \tag{7}$$

$$\text{s.t.} \quad (1),(2),(3),(4),(6),$$
$$x_{n,h}\in\{0,1\}, y_{n,u}\in\{0,1\}, \forall n\in\mathcal{N}, u\in\mathcal{U}, h\in\mathcal{H}, \tag{8}$$

where constraint (1) ensures that each BS caches one submodel per model type; constraint (2) limits the BS memory used by cached models; constraint (3) restricts each user request to at most one BS; constraint (4) keeps the end-to-end inference latency within user tolerance; and constraint (6) requires the DNN models to be cached before executing user requests at BSs. For ease of reference, important notations are listed in Table I.

IV. The CoCaR Approach

In this section, we transform problem $\mathcal{P}$, formulated under the assumption in Sec. III-C that user requests for the upcoming window are known in advance, into a linear programming problem. The assumption allows proactive downloading of the required models from the cloud to the edge server, followed by loading them into the server's memory. Then, we propose CoCaR to solve the transformed problem. Furthermore, we provide a detailed theoretical analysis and complexity evaluation for CoCaR, and extend it to practical scenarios.

A. Problem Transformation

The term $x_{n,h}\cdot y_{n,u}$ in problem $\mathcal{P}$ makes it a nonlinear integer programming problem, which is difficult to solve directly. To address this, we introduce the virtual binary integer variable $A_{n,u,h}$ to transform problem $\mathcal{P}$ into an equivalent integer linear programming problem $\mathcal{P}1$. We define

$$A_{n,u,h} = x_{n,h}\cdot y_{n,u}, \forall n\in\mathcal{N}, u\in\mathcal{U}, h\in\mathcal{H}(m_u). \tag{9}$$

Then, we introduce three additional constraints to equivalently represent Eq. (9):

$$\begin{aligned} A_{n,u,h}\le x_{n,h}, A_{n,u,h}\le y_{n,u},\\ x_{n,h}+y_{n,u}-1\le A_{n,u,h}. \end{aligned} \tag{10}$$

Subsequently, from Eq. (9), we can observe that $A_{n,u,h}$ has practical significance, i.e., $A_{n,u,h}=1$ indicates that BS n has cached submodel h and user u's request is routed to BS n. Thus, we can eliminate y to reduce the number of variables and simplify the problem.

Algorithm 1 CoCaR Algorithm

Input: $\{m_u, d_u, ddl_u, s_u\}, \tau, \tilde{x}_{n,h}(\tau-1), \mathcal{N}, \mathcal{M}$
Output: $\tilde{x}_{n,h}, \tilde{A}_{n,u,h}, \tilde{y}_{n,u}$

```
Solve problem P1-LR and get the optimal fractional solution x†_{n,h}, A†_{n,u,h}.
for n ∈ N, m ∈ M do
    Sample x̃_{n,m} from the multinoulli distribution Pr[x̃_{n,m} = I(h)] = x†_{n,h}, h ∈ H(m).
    Denote the sample result by x̃_{n,m} = I(ĥ).
    Set x̃_{n,ĥ} = 1, x̃_{n,h} = 0, h ∈ H(m)\{ĥ}.
end for
for u ∈ U, n ∈ N do
    for h ∈ H(m_u) do
        Set φ̃_{n,u,h} = 1 with probability A†_{n,u,h} / x†_{n,h}.
        Let Ã_{n,u,h} = x̃_{n,h} · φ̃_{n,u,h}.
    end for
    Let ỹ_{n,u} = 1(Σ_{h∈H(m_u)} Ã_{n,u,h} > 0).
end for
return x̃_{n,h}, Ã_{n,u,h}, ỹ_{n,u}
```

Therefore, problem $\mathcal{P}1$ involves only the variables $x_{n,h}$ and $A_{n,u,h}$. For each user $u\in\mathcal{U}$, there is

$$\sum_{n\in\mathcal{N}}\sum_{h\in\mathcal{H}(m_u)} A_{n,u,h} = \sum_{n\in\mathcal{N}}\sum_{h\in\mathcal{H}(m_u)} x_{n,h}\cdot y_{n,u} = \sum_{n\in\mathcal{N}} y_{n,u}, \tag{11}$$

which holds due to Eq. (1). Thus, the constraint (3) can be equivalently transformed into:

$$\sum_{n\in\mathcal{N}}\sum_{h\in\mathcal{H}(m_u)} A_{n,u,h}\le 1, \forall u\in\mathcal{U}. \tag{12}$$

Furthermore, for the T_u^{off} part of the end-to-end inference latency in Eq. (4), let $T_u^{\text{off}} = \sum_{n\in\mathcal{N}}\sum_{h\in\mathcal{H}(m_u)} x_{n,h}\cdot y_{n,u}\cdot(\frac{d_u}{\varphi_{\hat{n}_u}}+\frac{d_u}{r_{\hat{n}_u,n}}+\lambda_{u,n}), \forall u\in\mathcal{U}$, which is equivalent to the original constraint. Consequently, problem $\mathcal{P}1$ can be formulated as:

$$(\mathcal{P}1) \max_{x,A} \sum_{u\in\mathcal{U}}\sum_{n\in\mathcal{N}}\sum_{h\in\mathcal{H}(m_u)} A_{n,u,h}\cdot p_h \tag{13}$$

$$\text{s.t.} \quad (1),(2),(12),$$
$$A_{n,u,h}\le x_{n,h}, \forall n\in\mathcal{N}, u\in\mathcal{U}, h\in\mathcal{H}(m_u), \tag{14}$$
$$\sum_{n\in\mathcal{N}}\sum_{h\in\mathcal{H}(m_u)} A_{n,u,h}\cdot\Big(\big(\frac{d_u}{\varphi_{\hat{n}_u}}+\frac{d_u}{r_{\hat{n}_u,n}}+\lambda_{u,n}\big) + \frac{c_h\cdot d_u}{C_n}\Big)\le ddl_u, \forall u\in\mathcal{U}, \tag{15}$$
$$\sum_{n\in\mathcal{N}}\sum_{h,h'\in\mathcal{H}(m_u)} A_{n,u,h}\cdot x_{n,h'}(\tau-1)\cdot D_{m_u}(h',h) \le s_u, \forall u\in\mathcal{U}, \tag{16}$$
$$x_{n,h}\in\{0,1\}, \forall n\in\mathcal{N}, h\in\mathcal{H}, \tag{17}$$
$$A_{n,u,h}\in\{0,1\}, \forall n\in\mathcal{N}, u\in\mathcal{U}, h\in\mathcal{H}(m_u). \tag{18}$$

However, problem $\mathcal{P}1$ is still difficult to solve in polynomial time due to its non-convexity. To address this, we relax the

variables in $\mathcal{P}1$ and transform it into an LP problem:

$$(\mathcal{P}1\text{-LR}) \max_{x,A} \sum_{u\in\mathcal{U}}\sum_{n\in\mathcal{N}}\sum_{h\in\mathcal{H}(m_u)} A_{n,u,h}\cdot p_h \tag{19}$$

$$\text{s.t.} \quad (1),(2),(12),(14),(15),(16),$$

$$x_{n,h}\in[0,1], \forall n\in\mathcal{N}, h\in\mathcal{H}, \tag{20}$$

$$A_{n,u,h}\in[0,1], \forall n\in\mathcal{N}, u\in\mathcal{U}, h\in\mathcal{H}(m_u). \tag{21}$$

B. Approximation Algorithm

We propose the CoCaR algorithm to solve the JDCR problem using random rounding and linear programming. The details are provided below and summarized in Alg. 1.

First, by applying a linear programming solver [38] to solve problem $\mathcal{P}1$-LR, we can obtain the optimal fractional solutions $x^\dagger_{n,,h}$ and $A^\dagger_{n,u,h}$ (Line 1). Then, we round $x^\dagger_{n,h}$ and $A^\dagger_{n,u,h}$ to obtain vector $\tilde{\mathbf{x}}_{n,m}$ and intermediate variable $\tilde{\phi}_{n,u,h}$, where $\tilde{\mathbf{x}}_{n,m}$ contains the integer solutions $\tilde{x}_{n,h}$, and $\tilde{\phi}_{n,u,h}$ indicates whether user u's request attempts to route to submodel h at BS n for inference, serving to determine the integer solution $\tilde{A}_{n,u,h}$. Specifically, let the vector $\tilde{\mathbf{x}}_{n,m}$ be the indicator vector $\mathbb{I}(h)$ with probability $x^\dagger_{n,h}$ (Lines 3-5) and the rounding probability of $\tilde{\phi}_{n,u,h}$ be $\phi^\dagger_{n,u,h}$ (Lines 9-10):

$$\begin{aligned}&\Pr[\tilde{\mathbf{x}}_{n,m}=\mathbb{I}(h)] = x^\dagger_{n,h},\\ &\Pr[\tilde{\phi}_{n,u,h}=1] = \phi^\dagger_{n,u,h} = \frac{A^\dagger_{n,u,h}}{x^\dagger_{n,h}},\end{aligned} \tag{22}$$

where $\mathbb{I}(h)$ is a one-hot vector of size $|\mathcal{H}(m)|$, with the h-th element being 1 and 0 otherwise, indicating that the submodel h of model m is cached at BS n, i.e., $\tilde{x}_{n,h}=1$. Finally, let

$$\tilde{A}_{n,u,h} = \tilde{x}_{n,h}\cdot\tilde{\phi}_{n,u,h}. \tag{23}$$

Thus, we can revert $\tilde{y}_{n,u}$ based on $\tilde{A}_{n,u,h}$ (Line 12).

Then, we provide theoretical guarantees on the quality of the solutions returned by the CoCaR algorithm.

Lemma 1. *The solutions returned by CoCaR satisfy the constraints in problem $\mathcal{P}1$ in expectation.*

Proof. First, the inequality $\tilde{A}_{n,u,h}\le\tilde{x}_{n,h}$ holds according to Eq. (23), which satisfies constraint (14).

Second, for integer solutions $\tilde{x}_{n,h}$, $\sum_{h\in\mathcal{H}(m)}\tilde{x}_{n,h} = \|\tilde{\mathbf{x}}_{n,m}\|_1 = 1$ holds. Therefore, constraint (1) can be satisfied, where $\|\tilde{\mathbf{x}}_{n,m}\|_1$ is the L1 norm of vector $\tilde{\mathbf{x}}_{n,m}$.

Next, the expected amount of memory consumed by model caching at BS $n, n\in\mathcal{N}$ is given by: $\mathbb{E}[\sum_{h\in\mathcal{H}}\tilde{x}_{n,h}\cdot r_h] = \mathbb{E}[\sum_{m\in\mathcal{M}}\tilde{\mathbf{x}}_{n,m}\cdot\mathbf{r}_m] = \sum_{m\in\mathcal{M}}\sum_{h\in\mathcal{H}(m)}x^\dagger_{n,h}\cdot r_h \le R_n$, where vector $\mathbf{r}_m = [r_{h^m_0}, r_{h^m_1},\cdots,r_{h^m_{H(m)}}]$, and the last inequality holds due to constraint (2).

Then, constraint (12) is satisfied in expectation due to:

$$\begin{aligned}&\mathbb{E}[\sum_{n\in\mathcal{N}}\sum_{h\in\mathcal{H}(m_u)}\tilde{A}_{n,u,h}] = \sum_{n\in\mathcal{N}}\mathbb{E}[\tilde{\mathbf{x}}_{n,m_u}\cdot\vec{\tilde{\phi}}_{n,u}]\\ &=\sum_{n\in\mathcal{N}}\sum_{h\in\mathcal{H}(m_u)}\Pr[\tilde{\mathbf{x}}_{n,m_u}=\mathbb{I}(h)]\cdot\Pr[\tilde{\phi}_{n,u,h}=1]\\ &=\sum_{n\in\mathcal{N}}\sum_{h\in\mathcal{H}(m_u)}x^\dagger_{n,h}\cdot\frac{A^\dagger_{n,u,h}}{x^\dagger_{n,h}} = \sum_{n\in\mathcal{N}}\sum_{h\in\mathcal{H}(m_u)}A^\dagger_{n,u,h}\le 1,\end{aligned} \tag{24}$$

where vector $\vec{\tilde{\phi}}_{n,u} = [\tilde{\phi}_{n,u,h^{m_u}_0},\tilde{\phi}_{n,u,h^{m_u}_1},\cdots,\tilde{\phi}_{n,u,h^{m_u}_{H(m_u)}}]$. The first equation holds due to vector operations, and the third due to Eq. (22). The inequality holds by constraint (12).

Furthermore, the end-to-end inference latency perceived by user $u\in\mathcal{U}$ is expected to be:

$$\begin{aligned}&\mathbb{E}[\sum_{n\in\mathcal{N}}\sum_{h\in\mathcal{H}(m_u)}\tilde{A}_{n,u,h}\cdot((\frac{d_u}{\varphi_{\hat{n}_u}}+\frac{d_u}{r_{\hat{n}_u,n}}+\lambda_{u,n})+\frac{c_h\cdot d_u}{C_n})]\\ &=\mathbb{E}[\sum_{n\in\mathcal{N}}\vec{\tilde{\phi}}_{n,u}\cdot(\tilde{\mathbf{x}}_{n,m_u}\odot\vec{\tilde{\mathbb{T}}}_{n,u})]\\ &=\sum_{n\in\mathcal{N}}\sum_{h\in\mathcal{H}(m_u)}A^\dagger_{n,u,h}\cdot\hat{\mathbb{T}}_{n,u,h}\le ddl_u,\end{aligned} \tag{25}$$

where $\hat{\mathbb{T}}_{n,u,h} = ((\frac{d_u}{\varphi_{\hat{n}_u}}+\frac{d_u}{r_{\hat{n}_u,n}}+\lambda_{u,n})+\frac{c_h\cdot d_u}{C_n})$, vector $\vec{\hat{\mathbb{T}}}_{n,u} = [\hat{\mathbb{T}}_{n,u,h^{m_u}_0},\hat{\mathbb{T}}_{n,u,h^{m_u}_1},\cdots,\hat{\mathbb{T}}_{n,u,h^{m_u}_{H(m_u)}}]$ and $\odot$ represents the element-wise vector multiplication. The second equation holds due to Eq. (22), and the inequality is by constraint (15).

Finally, the model loading latency for user $u, u\in\mathcal{U}$ is expected to be:

$$\begin{aligned}&\mathbb{E}[\sum_{n\in\mathcal{N}}\sum_{h\in\mathcal{H}(m_u)}\sum_{h'\in\mathcal{H}(m_u)}\tilde{A}_{n,u,h}\cdot x_{n,h'}(\tau-1)\cdot D_{m_u}(h',h)]\\ &=\mathbb{E}[\sum_{n\in\mathcal{N}}\vec{\tilde{\phi}}_{n,u}\cdot(\tilde{\mathbf{x}}_{n,m_u}\odot\vec{\hat{D}}_{n,m_u})]\\ &=\sum_{n\in\mathcal{N}}\sum_{h,h'\in\mathcal{H}(m_u)}A^\dagger_{n,u,h}\cdot x_{n,h'}(\tau-1)\cdot\hat{D}_{n,h}\le s_u,\end{aligned} \tag{26}$$

where $\hat{D}_{n,h} = \sum_{h'\in\mathcal{H}(m_u)}x_{n,h'}(\tau-1)\cdot D_{m_u}(h',h)$, and vector $\vec{\hat{D}}_{n,m_u} = [\hat{D}_{n,h^{m_u}_0},\hat{D}_{n,h^{m_u}_1},\cdots,\hat{D}_{n,h^{m_u}_{H(m_u)}}]$. The last inequality holds due to constraint (16). □

Lemma 2. *The objective value obtained by the CoCaR is equal to that of the optimal fractional solution in expectation.*

Proof. First, let $\mathbb{P}^\dagger$ be the objective value in the optimal fractional solution, $\mathbb{P}^\dagger = \sum_{u\in\mathcal{U}}\sum_{n\in\mathcal{N}}\sum_{h\in\mathcal{H}(m_u)}A^\dagger_{n,u,h}\cdot p_h$. In expectation, the total inference precision of user requests that the CoCaR algorithm can obtain is:

$$\begin{aligned}\mathbb{E}[\tilde{\mathbb{P}}] &= \mathbb{E}[\sum_{u\in\mathcal{U}}\sum_{n\in\mathcal{N}}\sum_{h\in\mathcal{H}(m_u)}\tilde{A}_{n,u,h}\cdot p_h]\\ &=\sum_{u\in\mathcal{U}}\sum_{n\in\mathcal{N}}\mathbb{E}[\vec{\tilde{\phi}}_{n,u}\cdot(\tilde{\mathbf{x}}_{n,m_u}\odot\vec{P}_{m_u})]\\ &=\sum_{u\in\mathcal{U}}\sum_{n\in\mathcal{N}}\sum_{h\in\mathcal{H}(m_u)}A^\dagger_{n,u,h}\cdot p_h = \mathbb{P}^\dagger,\end{aligned} \tag{27}$$

where vector $\vec{P}_{m_u} = [p^{m_u}_0, p^{m_u}_1,\cdots,p^{m_u}_{H(m_u)}]$. □

The above lemmas show that CoCaR has theoretical guarantees in expectation. However, in practice, random rounding may lead to constraint violations. Next, we analyze in detail using the Chernoff Bound theorem [39].

Definition 1 (Chernoff Bound [39]). *Given I independent random variables $z_1, z_2, \ldots, z_I$, where for all z_i obey multinoulli distribution, and $z_i\in[0,1]$. Let $\mu = \mathbb{E}[\sum_{i=1}^I z_i]$. Then, it holds that $\Pr[\sum_{i=1}^I z_i \ge (1+\delta)\mu] \le exp^{\frac{-\delta^2\mu}{2+\delta}}$, $\forall\ \delta>0$ and $\Pr[\sum_{i=1}^I z_i \le (1-\delta)\mu] \le exp^{\frac{-\delta^2\mu}{2}}$, $\forall\ 0<\delta<1$.*

Theorem 1. *There is a high probability that the solution returned by the CoCaR algorithm has at least* $(1-\sqrt{\frac{4\ln|\mathcal{H}|}{\mathbb{P}^\dagger}})^2$ *approximation ratio with the optimal integer solution, where* $\mathbb{P}^\dagger$ *is the objective value obtained from the optimal fractional solution, under the assumption* $\mathbb{P}^\dagger \geq 4\ln|\mathcal{H}|$.

Proof. According to Eq. (22), $\mathbb{P}^\dagger$ is equivalent to $\mathbb{P}^\dagger = \sum_{u\in\mathcal{U}}\sum_{n\in\mathcal{N}}\sum_{h\in\mathcal{H}(m_u)} x^\dagger_{n,h}\cdot\phi^\dagger_{n,u,h}\cdot p_h$. Next, to satisfy the independence requirement for the Chernoff Bound, we provide a two-stage proof.

In stage I, we first keep $\phi^\dagger_{n,u,h}$ in $\mathbb{P}^\dagger$ unchanged and round $x^\dagger_{n,h}$ to $\tilde{x}_{n,h}$ to obtain $\mathbb{P}'$:

$$\begin{aligned}\mathbb{P}' &= \sum_{u\in\mathcal{U}}\sum_{n\in\mathcal{N}}\sum_{h\in\mathcal{H}(m_u)} \tilde{x}_{n,h}\cdot\phi^\dagger_{n,u,h}\cdot p_h \\ &= \sum_{n\in\mathcal{N}}\sum_{m\in\mathcal{M}}(\tilde{\mathbf{x}}_{n,m}\cdot\sum_{u\in\{u|m_u=m\}}\vec{P}_u),\end{aligned} \tag{28}$$

where $\vec{P}_u = [\frac{A^\dagger_{n,u,h_0^{m_u}}}{x^\dagger_{n,h_0^{m_u}}}\cdot p_{h_0^{m_u}},\cdots,\frac{A^\dagger_{n,u,h_{H(m_u)}^{m_u}}}{x^\dagger_{n,h_{H(m_u)}^{m_u}}}\cdot p_{h_{H(m_u)}^{m_u}}]$.

Denote $v_{n,m} = \tilde{\mathbf{x}}_{n,m}\cdot\sum_{u\in\{u|m_u=m\}}\vec{P}_u$, the determinism of $\sum_{u\in\{u|m_u=m\}}\vec{P}_u$ and the independence of $\tilde{\mathbf{x}}_{n,m}$ ensure that $v_{n,m}$ are independent. Clearly, we have $\mathbb{E}[\mathbb{P}'] = \mathbb{P}^\dagger$, and by the Chernoff Bound theorem, when $\delta_1\in[0,1]$, we have

$$\Pr[\mathbb{P}' \leq (1-\delta_1)\mathbb{P}^\dagger] = q_1 \leq exp(\frac{-\delta_1^2\mathbb{P}^\dagger}{2}). \tag{29}$$

Next, find a δ_1 value for the right side of Eq. (29) to make it very small. Specifically, we require,

$$exp(\frac{-\delta_1^2\mathbb{P}^\dagger}{2}) \leq \frac{1}{|\mathcal{H}|^2}. \tag{30}$$

This means that as the number of submodels increases, the probability bound converges to zero quickly. Eq. (30) holds when $\delta_1 \geq \sqrt{\frac{4\ln|\mathcal{H}|}{\mathbb{P}^\dagger}}$, which is true if we pick $\delta_1 = \sqrt{\frac{4\ln|\mathcal{H}|}{\mathbb{P}^\dagger}}$.

In stage II, building upon the rounded result $\tilde{x}_{n,h}$ in stage I, we round $\phi^\dagger_{n,u,h}$ to $\tilde{\phi}_{n,u,h}$, resulting in the equation $\tilde{\mathbb{P}}$: $\tilde{\mathbb{P}} = \sum_{u\in\mathcal{U}}\sum_{n\in\mathcal{N}}\sum_{h\in\mathcal{H}(m_u)}\tilde{x}_{n,h}\cdot\tilde{\phi}_{n,u,h}\cdot p_h$. Since $\tilde{x}_{n,h}\cdot p_h$ are deterministic and $\tilde{\phi}_{n,u,h}$ are independent, it follows that: $\mathbb{E}[\tilde{\mathbb{P}}] = \sum_{u\in\mathcal{U}}\sum_{n\in\mathcal{N}}\sum_{h\in\mathcal{H}(m_u)}\tilde{x}_{n,h}\cdot\Pr[\tilde{\phi}_{n,u,h}=1]\cdot p_h = \mathbb{P}'$. By the Chernoff Bound theorem, when $\delta_2\in[0,1]$, we have:

$$\Pr[\tilde{\mathbb{P}} \leq (1-\delta_2)\mathbb{P}'] = q_2 \leq exp(\frac{-\delta_2^2\mathbb{P}'}{2}). \tag{31}$$

Since stage I and stage II are independent, when the condition $\mathbb{P}' \geq (1-\delta_1)\mathbb{P}^\dagger$ in Eq. (29) is satisfied, we have:

$$\begin{aligned}q_2 &= \Pr[\tilde{\mathbb{P}} \leq (1-\delta_2)\mathbb{P}'|\mathbb{P}' \geq (1-\delta_1)\mathbb{P}^\dagger] \\ &\leq exp(\frac{-\delta_2^2\mathbb{P}'}{2}) \leq exp(\frac{-\delta_2^2(1-\delta_1)\mathbb{P}^\dagger}{2}).\end{aligned} \tag{32}$$

Next, a value of δ_2 is specified to make the right side of Eq. (32) become very small. Specifically, we require:

$$exp(\frac{-\delta_2^2(1-\delta_1)\mathbb{P}^\dagger}{2}) \leq \frac{1}{|\mathcal{H}|^{2(1-\delta_1)}}, \tag{33}$$

Eq. (33) holds when δ_2 satisfies $\delta_2 \geq \sqrt{\frac{4\ln|\mathcal{H}|}{\mathbb{P}^\dagger}}$. We select $\delta_2 = \sqrt{\frac{4\ln|\mathcal{H}|}{\mathbb{P}^\dagger}}$ to make the inequality hold.

Under the conditions specified in (30) and (33), by combining (29) and (31), we obtain:

$$\begin{aligned}&\Pr[\tilde{\mathbb{P}} \geq (1-\delta_1)(1-\delta_2)\mathbb{P}^\dagger] \\ &\geq \Pr[\mathbb{P}' \geq (1-\delta_1)\mathbb{P}^\dagger]\Pr[\tilde{\mathbb{P}} \geq (1-\delta_2)\mathbb{P}'|\mathbb{P}' \geq (1-\delta_1)\mathbb{P}^\dagger] \\ &= (1-q_1)(1-q_2) \geq (1-\frac{1}{|\mathcal{H}|^2})(1-\frac{1}{|\mathcal{H}|^{2(1-\delta_1)}}).\end{aligned} \tag{34}$$

Since $\tilde{\mathbb{P}}^* \leq \mathbb{P}^\dagger$, where $\tilde{\mathbb{P}}^*$ is the optimal integer solution, it follows that: $\Pr[\tilde{\mathbb{P}} \geq (1-\delta_1)(1-\delta_2)\tilde{\mathbb{P}}^*] \geq \Pr[\tilde{\mathbb{P}} \geq (1-\delta_1)(1-\delta_2)\mathbb{P}^\dagger] \geq (1-\frac{1}{|\mathcal{H}|^2})(1-\frac{1}{|\mathcal{H}|^{2(1-\delta_1)}})$.

In practice, as the number of user requests, BSs, and models increases, $\mathbb{P}^\dagger$ related to $u\in\mathcal{U}, n\in\mathcal{N}, h\in\mathcal{H}$ will significantly exceed $4\ln|\mathcal{H}|$ ($\mathbb{P}^\dagger \gg 4\ln|\mathcal{H}|$). Therefore, the objective value obtained by the CoCaR algorithm has an approximation ratio of at least $(1-\delta_1)(1-\delta_2) = (1-\sqrt{\frac{4\ln|\mathcal{H}|}{\mathbb{P}^\dagger}})^2$ to the optimal integer solution $\tilde{\mathbb{P}}^*$ with a high probability. □

Theorem 2. *For BS* $n\in\mathcal{N}$*, the memory consumption of the cached model returned by the CoCaR algorithm has a high probability of not exceeding its memory capacity by more than a factor of* $(\sqrt{\frac{2\ln|\mathcal{H}|}{\zeta_n^\dagger}}+\frac{1}{\sqrt{2}})^2+\frac{1}{2}$*, where* $\zeta_n^\dagger$ *is the memory consumption of BS* n *in the optimal fractional solution, under the assumption* $\zeta_n^\dagger \geq \ln|\mathcal{H}|$.

Proof. According to Lemma 1, $\mathbb{E}[\sum_{h\in\mathcal{H}}\tilde{x}_{n,h}\cdot r_h] = \mathbb{E}[\sum_{m\in\mathcal{M}}\tilde{\mathbf{x}}_{n,m}\cdot\mathbf{r}_m] = \sum_{m\in\mathcal{M}}\sum_{h\in\mathcal{H}(m)}x^\dagger_{n,h}\cdot r_h$, where $\tilde{\mathbf{x}}_{n,m}\cdot\mathbf{r}_m$ are independent random variables and can be normalized into values within [0,1]. Thus, we apply the Chernoff Bound theorem to show that for any $\delta>0$: $\Pr[\sum_{h\in\mathcal{H}}\tilde{x}_{n,h}\cdot r_h \geq (1+\delta)\sum_{m\in\mathcal{M}}\sum_{h\in\mathcal{H}(m)}x^\dagger_{n,h}\cdot r_h] \leq e^{\frac{-\delta^2\sum_{m\in\mathcal{M}}\sum_{h\in\mathcal{H}(m)}x^\dagger_{n,h}\cdot r_h}{2+\delta}}$.

Denote $\zeta_n^\dagger = \sum_{m\in\mathcal{M}}\sum_{h\in\mathcal{H}(m)}x^\dagger_{n,h}\cdot r_h$. Since $\zeta_n^\dagger \leq R_n$, it follows that: $\Pr[\sum_{h\in\mathcal{H}}\tilde{x}_{n,h}\cdot r_h \geq (1+\delta)R_n] \leq e^{\frac{-\delta^2\zeta_n^\dagger}{2+\delta}}$. Then, similar to the proof in Theorem 1, to make the right side of the inequality small, i.e., $e^{\frac{-\delta^2\zeta_n^\dagger}{2+\delta}} \leq \frac{1}{|\mathcal{H}|^2}$, δ must satisfy $\delta \geq \frac{\ln|\mathcal{H}|}{\zeta_n^\dagger}+\sqrt{\frac{\ln^2|\mathcal{H}|}{\zeta_n^{\dagger 2}}+\frac{4\ln|\mathcal{H}|}{\zeta_n^\dagger}}$.

We select $\delta = \frac{2\ln|\mathcal{H}|}{\zeta_n^\dagger}+2\sqrt{\frac{\ln|\mathcal{H}|}{\zeta_n^\dagger}}$. Since $\zeta_n^\dagger \geq \ln|\mathcal{H}|$ holds in practice, with a high probability, the memory capacity of BS n will not exceed by more than a factor of $1+\delta = (\sqrt{\frac{2\ln|\mathcal{H}|}{\zeta_n^\dagger}}+\frac{1}{\sqrt{2}})^2+\frac{1}{2}$. □

By similar proofs, we can prove the following three theorems about constraints (12), (15), and (16).

Theorem 3. *For user* $u\in\mathcal{U}$*, there is a high probability that* $\sum_{n\in\mathcal{N}}\sum_{h\in\mathcal{H}(m_u)}\tilde{A}_{n,u,h}$ *is no greater than* $(\sqrt{\frac{2\ln|\mathcal{H}|}{\eta^\dagger}}+\frac{1}{\sqrt{2}})^2+\frac{1}{2}$*, where,* $\eta^\dagger = \sum_{n\in\mathcal{N}}\sum_{h\in\mathcal{H}(m_u)}A^\dagger_{n,u,h}$.

Theorem 4. *For user* $u\in\mathcal{U}$*, CoCaR's end-to-end inference latency has a high probability of not exceeding the user's tolerable perceived latency by a factor larger than* $(\sqrt{\frac{2\ln|\mathcal{H}|}{\mathbb{T}_u^\dagger}}+\frac{1}{\sqrt{2}})^2+\frac{1}{2}$*, where* $\mathbb{T}_u^\dagger$ *is the end-to-end inference latency perceived by user* u *in the optimal fractional solution.*

Theorem 5. *For model type m_u requested by user $u, u \in \mathcal{U}$, there is a high probability that CoCaR's model loading latency will not exceed user u's request initiation time by more than a factor of $(\sqrt{\frac{2\ln|\mathcal{H}|}{T^{dep\dagger}_{m_u}}} + \frac{1}{\sqrt{2}})^2 + \frac{1}{2}$, where $T^{dep\dagger}_{m_u}$ is the model loading latency of model m_u in the optimal fractional solution.*

C. Complexity Analysis

First, we analyze the scale of the variables and constraints in problem $\mathcal{P}$1-LR. Let $h^* = \max_m |\mathcal{H}(m)|, m \in \mathcal{M}$, thus the number of variables $x_{n,h}$ and $A_{n,u,h}$ are bounded by $O(N|\mathcal{H}|)$ and $O(NUh^*)$, respectively. Therefore, the total number of decision variables is bound by $O(N|\mathcal{H}|) + O(NUh^*) \leq O(NMh^*) + O(NUh^*) \leq O(NUh^*)$, given that $M < U$ in practice. Similarly, the number of constraints is bounded by $O(NM)+O(N)+3O(U)+O(NUh^*)$. Since $M < U < Uh^*$, the total number of constraints is also $O(NUh^*)$.

Then, considering the three loops in Lines 2, 7, and 8 of Algorithm 1, the time complexity of CoCaR is $O(NM) + O(NUh^*) \leq O(NUh^*)$, as $M < U$ holds.

D. Extension to Practice

Since the CoCaR algorithm's rounded solution may violate a few constraints, we employ a heuristic strategy to convert the rounded solutions $\tilde{x}_{n,h}, \tilde{y}_{n,u}$ into feasible ones $x_{n,h}, y_{n,u}$. First, for BS $n \in \mathcal{N}$ that violates the memory capacity constraint, we evaluate the benefit of each model type $m \in \mathcal{M}$ based on user requests and inference precision p_h (where $\tilde{x}_{n,h} = 1, h \in \mathcal{H}(m)$). We remove the least beneficial submodel and try to cache smaller ones. If no suitable submodel can be cached, user requests previously routed to that BS are redirected to the cloud. This process continues until the memory constraint (2) is satisfied. Next, for users $u \in \mathcal{U}$ violating constraints on maximum tolerable perceived latency or model loading latency, requests are redirected to the cloud until constraints (15) and (16) are met. Finally, if multiple $y_{n,u} = 1$ for a user u, the request is routed to the BS with the highest inference precision. These steps ensure all caching and routing decisions satisfy the constraints.

V. Adaptation to Online Scenarios

In real-world scenarios, the assumption made in Sec. III-C, i.e., future user requests can be predicted in advance and the required models can be pre-downloaded from the cloud to edge servers, does not always hold [40], [41]. Thus, caching decisions can only be made after receiving the current user request, and the user must wait until the required model is cached before being served. To address this, we further extend the CoCaR approach to an online version, CoCaR-OL, which adjusts the dynamic model caching strategy according to real-time user request patterns.

A. System Model

As shown in Fig. 5, in the online scenario, model caching decisions are made based on historical user request patterns in each time slot. Therefore, when deciding to cache a larger

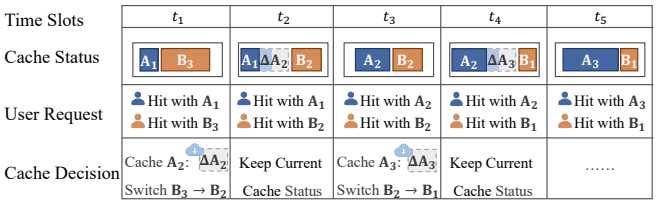

Fig. 5. Illustration of model caching in an online scenario. There are two types of models, A and B, each comprising three distinct submodels. At time slot t_1, the system decides to cache submodel A_2, which requires downloading the additional component ΔA_2 from the cloud to switch from A_1 to A_2. To satisfy the cache capacity constraint, submodel B_3 is simultaneously reduced to B_2. Since downloading ΔA_2 takes two time slots, A_2 becomes available to serve users only from time slot t_3 onward. In contrast, model eviction is fast, allowing B_2 to serve users immediately at t_2.

submodel, the components required for the submodel switching need to be downloaded from the cloud, which may not be completed within the current time slot. Since each submodel h^m_{i+1} is typically constructed by incrementally adding components to h^m_i, the system can better utilize cached models to serve users by downloading and caching the intermediate submodels in sequence. For example, if a cached submodel h^m_1 is decided to be switched to submodel h^m_3, it will first switch to submodel h^m_2, and then to submodel h^m_3 as the additional required components are downloaded. To model this online scenario, we introduce some new notations. Let W_n be the bandwidth between the cloud and base station n, and Δt be the duration of time slot t. Let $\tilde{O}^t_{n,h}$ and $O^t_{n,h}$ denote the sizes of data for submodel h to be downloaded at BS n before and after the caching decision in time slot t, respectively. Similarly, $\tilde{X}^t_{n,h}$ and $X^t_{n,h}$ represent the caching status of submodel h at BS n before and after the caching decision in time slot t, respectively.

Thus, at time slot t, the remaining data size of submodel h to be downloaded at BS n can be calculated as:

$$\tilde{O}^t_{n,h} = \begin{cases} \max(O^{t-1}_{n,h} - \bar{t} \cdot W_n, 0), & \text{if } O^{t-1}_{n,h} \neq 0, \\ 0, & \text{otherwise,} \end{cases} \tag{35}$$

where $\bar{t} = \Delta t - \min\left(\frac{\sum_{h' \prec_h} O^{t-1}_{n,h'}}{W_n}, \Delta t\right)$, meaning that each submodel to be downloaded can start downloading immediately after the previous submodel is downloaded.

Then, whether submodel h has finished downloading at time slot t can be denoted as:

$$g^t_{n,h} = \mathbf{1}\{O^{t-1}_{n,h} \neq 0 \wedge \tilde{O}^t_{n,h} = 0\}, \tag{36}$$

where $\mathbf{1}\{a\} = 1$ if condition a is true, and 0 otherwise.

Next, based on the downloading status of submodels, the caching state can be calculated as:

$$\tilde{X}^t_{n,h} = \begin{cases} 1, & \text{if } g^t_{n,h} = 1 \wedge \sum_{h' \succ_h} g^t_{n,h'} = 0, \\ 0, & \text{if } \sum_{h' \succ_h} g^t_{n,h'} \geq 1, \\ X^{t-1}_{n,h}, & \text{otherwise.} \end{cases} \tag{37}$$

Specifically, if at time slot $t-1$, submodel h at BS n has just been downloaded, and no submodel larger than h has been downloaded in this slot, then submodel h will be cached and $\tilde{X}^t_{n,h}$ is set to 1. Conversely, if at time slot t, a larger submodel

than h that has been downloaded at BS n, so that submodel h should not be cached, i.e., $\tilde{X}^t_{n,h}$ is set to 0. Otherwise, the caching state of submodel h at BS n remains unchanged.

Consequently, based on the above caching state, the inference precision that model type m can provide at BS n during time slot t can be calculated as:

$$P_{n,m}(t) = \sum_{h \in \mathcal{H}(m)} \tilde{X}^t_{n,h} \cdot p_h. \tag{38}$$

In addition, similar to Eq. (4), the total end-to-end inference latency at time slot t for routing user requests of model type m from home BS n' to target BS n for inference can be calculated as:

$$\mathbb{T}^t_m(n', n) = \frac{d_m}{\varphi_{n'}} + \frac{d_m}{r_{n',n}} + \lambda_{n'n} + \sum_{h \in \mathcal{H}(m)} \tilde{X}^t_{n,h} \cdot \frac{c_h \cdot d_m}{C_n}, \tag{39}$$

where d_m represents the input data size of model type m.

Finally, let $Q^t_m(n', n)$ be the Quality of Experience (QoE) at time slot t for routing a user request of model type m with home BS n' to target BS n for inference. To comprehensively evaluate the trade-off between inference precision and the user-perceived end-to-end inference latency, we define the QoE function as follows:

$$Q^t_m(n', n) = P_{n,m}(t) \cdot \max(0, 1 - (\mathbb{T}^t_m(n', n) - \theta) \cdot \alpha), \tag{40}$$

where θ is a normalization factor (i.e., the minimum end-to-end inference latency), and α is a smoothing factor that controls the degradation of precision due to end-to-end inference latency. Specifically, when $\mathbb{T}^t_m(n', n) > \theta$, the QoE value $Q^t_m(n', n)$ falls below the original precision $P_{n,m}(t)$, reflecting the negative impact of latency; conversely, if the inference latency is equal to θ, the QoE will not be degraded.

Accordingly, the target BS that maximizes the QoE for a user requesting model m with home BS n' at time slot t can be denoted as:

$$n^{t*}_m(n') = \arg\max_{n \in \mathcal{N}} Q^t_m(n', n). \tag{41}$$

Problem Formulation. In the online model caching and request routing optimization problem, we route user requests to the BS that maximizes their QoE. The optimization objective is to maximize the total QoE of all users by designing an effective caching policy. The problem is formulated as follows:

$$(\mathcal{P}2) \qquad \max_X \sum_{t \in \mathcal{T}} \sum_{u \in \mathcal{U}} Q^t_{m_u}(\hat{n}_u, n^{t*}_m(\hat{n}_u)) \tag{42}$$

$$\text{s.t.} \quad \sum_{h \in \mathcal{H}} X^t_{n,h} \cdot r_h \le R_n, \forall t \in \mathcal{T}, n \in \mathcal{N} \tag{43}$$

$$\mathbb{T}^t_{m_u}(\hat{n}_u, n^{t*}_{m_u}(\hat{n}_u)) \le ddl_u, \forall t \in \mathcal{T}, u \in \mathcal{U}. \tag{44}$$

B. Algorithm Design

To solve problem $\mathcal{P}2$, we propose CoCaR-OL, a heuristic caching algorithm based on predictive future gain, which enables fine-grained perception of changes in user request patterns in each time slot and allows flexible adjustment of cached models. In CoCaR-OL, the action space for enlarging cached submodels is defined as the set of all submodels from the currently cached one up to the first submodel whose additional components, relative to the cached submodels, cannot be fully downloaded within a time slot.

Algorithm 2 CoCaR-OL Algorithm

Input: $\mathcal{U}, \mathcal{N}, \mathcal{M}, round$
Output: $X^t_{n,h}, O^t_{n,h}$

```
Initialize time slot variable t = 0.
Initialize model caching and download status X^0_{n,h}, O^0_{n,h}.
while True do                          ▷ Start online monitoring and deciding
    Step to new time slot: t ← t + 1.
    Routine Update: Calculate temporary download status Õ^t_{n,h} according to Eq. (48).
    Routine Update: Calculate download-finishing flag variable g^t_{n,h} and temporary cache status X̃^t_{n,h} according to Eq. (49).
    Receive current user requests {m^t_u | u ∈ U}.
    Calculate the best routing destination n^{t*}_m(n') from Eq. (41).
    for u ∈ U do                       ▷ Route user requests and calc. total QoE
        Route use request m_u to the best destination n^{t*}_{m_u}(n̂_u).
        Calculate the corresponding QoE value Q^t_{m_u} by Eq. (40).
    end for
    Update the recent proportion of user requests f^t_{n,m} by Eq. (45).
    O^t_{n,h} := Õ^t_{n,h}, X^t_{n,h} := X̃^t_{n,h}.
    for r = 1 → round do               ▷ Make model caching decision
        Randomly choose a BS n for caching adjustment.
        Compute the future expected gain ΔR for each possible cached model switch based on O^t_{n,h}, X^t_{n,h} according to Eq. (47).
        Find the best scheme for model switching (caching and evicting) by solving a memory-constrained knapsack problem.
        Switch Update: Update download status O^t_{n,h} by Eq. (48).
        Switch Update: Update cache status X^t_{n,h} by Eq. (49).
    end for
end while
```

Specifically, we first use the user request patterns from the past ΔT^P time slots to simulate user requests for the future ΔT^F time slots. In particular, the proportion of user requests for model type m with home BS n over the past ΔT^P time slots can be denoted as:

$$f^t_{n,m} = \frac{\sum_{i=t-\Delta T^P}^{i=t} \sum_{u \in \mathcal{U}} \mathbf{1}\{m^i_u = m \wedge \hat{n}^i_u = n\}}{\sum_{i=t-\Delta T^P}^{i=t} U}. \tag{45}$$

Next, let $\pi^t_{n,m}(X^t; O^t) \in \{(h', h) \mid X^t_{n,h'} = 1, h \in \mathcal{H}(m)\}$ denote the action taken at time slot t for a cached submodel h' of model type m at BS n, under the observation of system states X^t and O^t. When $h' \succ h$, it indicates a switch from the cached submodel h' to a smaller submodel h; conversely, it indicates a switch to a larger submodel h or remaining unchanged. Then, at time slot t, the expected future reward of switching a cached submodel h' of model m at BS n to a submodel h, while keeping all other system states unchanged, can be computed as:

$$R(\pi^t_{n,m} = (h', h)) = \sum_{t'=t+1}^{\Delta T^F} \sum_{n' \in \mathcal{N}} \gamma^{(t'-t)} \cdot f^{t'}_{n',m} \cdot Q^{t'}_m\big(n', n^{t'*}_m(n')\big), \tag{46}$$

where γ^t denotes the discount factor of the user's QoE in the tth future time slot.

Therefore, the expected future gain from replacing the cached submodel h' of model m at BS n with h can be computed as:

$$\Delta R(\pi^t_{n,m} = (h', h)) = R(\pi^t_{n,m} = (h', h)) - R(\pi^t_{n,m} = (h', h')). \quad (47)$$

Finally, after making the caching decision at time slot t, if it is decided to switch from the cached submodel h' at BS n to a larger target submodel $\hat{h}^t$, the download status of submodel h can be computed as:

$$O^t_{n,h} = \begin{cases} \Delta r_h, & \text{if } \tilde{O}^t_{n,h} = 0 \wedge \sum_{h' \prec h} \tilde{X}^t_{n,h'} = 1 \wedge h \preceq \hat{h}^t, \\ \tilde{O}^t_{n,h}, & \text{otherwise,} \end{cases} \quad (48)$$

where Δr_h represents the additional data size that each submodel h between the cached submodel h' and the target submodel $\hat{h}^t$ needs to download relative to its preceding submodel. Otherwise, if the decision is to switch from the cached submodel h' at BS n to a smaller target submodel $\hat{h}^t$, the model's cache state can be updated directly as the time required for cache eviction is negligible:

$$X^t_{n,h} = \begin{cases} 1, & \text{if } h = \hat{h}^t, \\ 0, & \text{if } h = h', \\ \tilde{X}^t_{n,h}, & \text{otherwise.} \end{cases} \quad (49)$$

The procedure of the CoCaR-OL algorithm is shown in Alg. 2. At the beginning of each time slot t, the download and cache states of each submodel, $\tilde{O}^t_{n,h}$ and $\tilde{X}^t_{n,h}$, are updated through a routine process (Lines 5-6). Then, the maximum QoE for each user during the current time slot is calculated, and the historical request frequency $f^t_{n,m}$ of each model m at each BS n is updated (Lines 7-13). In each iteration, a BS n is randomly selected, and all submodels of models that are not being downloaded at BS n are traversed. For each candidate submodel, it is hypothetically enlarged, and a knapsack problem is solved under the constraint of the remaining memory capacity of BS n. This step selects a combination of cached submodels from other candidate models and their smaller submodels, such that the expected future gain of cache switching is maximized (Lines 16-18). The same process is then repeated for other candidate submodels. Finally, the model switching scheme that yields the maximum expected future gain is chosen from all the options, and the corresponding model states $O^t_{n,h}$ and $X^t_{n,h}$ are updated accordingly (Lines 19-20).

VI. Performance evaluation

In this section, we conduct simulation experiments to evaluate the proposed CoCaR algorithm's performance after converting its solution into feasible ones (Sec. IV-D) as well as the performance of the CoCaR-OL algorithm in online scenarios.

A. Evaluation Setup

Following a setup similar to [22] and [25], the duration of each observation window is set to $\Delta\tau = 3$s and the number of windows $|\Gamma| = 10$, i.e., the total time is 30s. We consider an MEC network with $N = 5$ BSs and $U = 600$ users in each window. The coverage range of a BS is 150 m, and the connection between BSs is established using the Erdős–Rényi random graph model [42]. The wireless uplink transmission rate is set to $\varphi_{\hat{n}_u} = 20$ Mbps, and the transmission rate between BSs is set to $r_{\hat{n}_u n} = 100$ Mbps. The bandwidth between the cloud and the base station is set to $W_n = 800$ Mbps. The propagation time for one hop is set to 0.01s. We set the memory capacity for each BS n to $R_n = 500$MB and the maximum computing power $C_n = 70$ Gflops/s.

TABLE II
Attributes of the Three ViT Submodels

Submodel	Memory (MB)	FLOPs (GFlops)	Precision
1	174.32	5.70	0.8417
2	227.42	7.56	0.9413
3	342.05	11.29	0.9894

TABLE III
Loading Time of the Three ViT Submodels (s)

Original Submodels	Final Submodels		
	1	2	3
/	0.68860	0.87696	1.05821
1	0.00000	0.24794	0.46098
2	0.04238	0.00000	0.25082
3	0.04725	0.04242	0.00000

We use $M = 8$ different DNN model types (e.g., ViT, swintransformer [43], etc.), with each model type having three submodels. Each submodel is trained and tested on the CIFAR-10 dataset [44]. Taking ViT as an example. Tables II and III show the attributes and loading times of the ViT submodels, respectively. Table II shows the differences in memory consumption and inference precision among the ViT submodels. Table III indicates that loading submodel 2 directly from the secondary storage takes 0.877s, whereas switching from submodel 1 to submodel 2 takes only 0.248s, thereby increasing the valid service time of ViT. The popularity of model types follows a Zipf distribution with a skewness coefficient of 0.8 [45], and each user generates a request for a random model type. The data size of each request and model input is $d_u = d_m = 0.144$ MB. The maximum tolerable latency for the user is set to 0.3s. We implemented the CoCaR using Python 3.10.14 and conducted all experiments on an Intel(R) Xeon(R) Gold 6230R CPU @ 2.10 GHz with eight NVIDIA GeForce RTX 3070 GPUs.

B. Benchmarks and Evaluation Metrics

We compare CoCaR with the following algorithms.

- **Linear-Relaxation (LR).** The optimal fractional solution of problem $\mathcal{P}$1-LR, obtained via a linear programming solver, serves as an upper bound for the optimal integer solution but is typically unattainable in practice.
- **SPR**^{3} [22]. It uses random rounding to solve the joint service caching and request routing problem under memory, computation, and communication constraints.

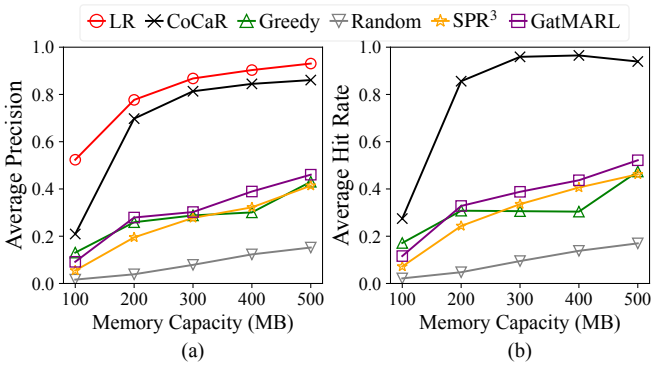

Fig. 6. Impact of different BS memory capacities: (a) Average inference precision; (b) Average hit rate.

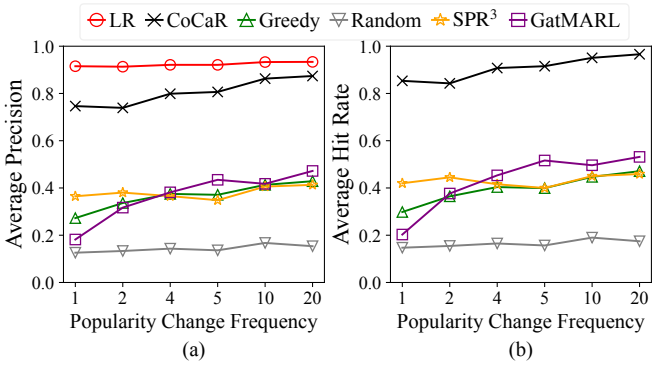

Fig. 7. Impact of popularity change frequency: (a) Average inference precision; (b) Average hit rate. The x-axis indicates how many observation windows the popularity changes once.

- **GatMARL** [46]. GatMARL constructs the MEC environment as an undirected graph, applying a graph attention-based multi-agent reinforcement learning algorithm to learn task offloading and service caching policies.
- **Greedy.** The model type is selected based on popularity, and one of its submodels is cached in descending order of precision. User requests are routed to the home BS.
- **Random.** The models cached at each BS are randomly selected from the submodels associated with each model type. User requests are randomly routed to a BS.

Note that SPR^3 and GatMARL only cache the complete service, and all benchmarks except LR ignore the impact of model loading time on decision-making.

Our evaluation is based on the following metrics:

- **Average inference precision.** It reflects the average inference precision for all user requests over all windows.
- **Average QoE of Users.** It reflects the average QoE for all user requests under the precision and latency tradeoff over all time slots, which is calculated according to Eq. (40) in Sec. V-A.
- **Average hit rate.** It reflects the ratio of requests served by the BS over all windows.
- **Average memory utilization.** It reflects the memory usage ratio of cached models at the BS over all windows.

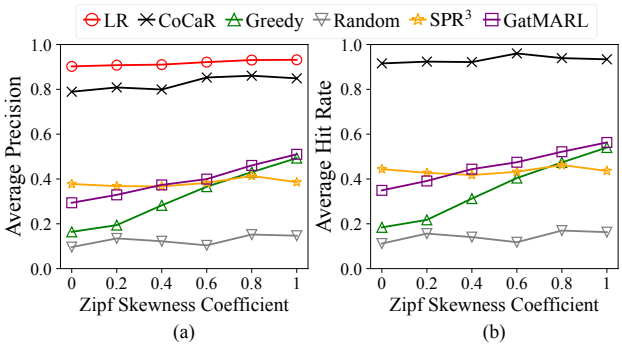

Fig. 8. Impact of different Zipf skewness coefficients:(a) Average inference precision; (b) Average hit rate.

C. Evaluation Results and Analysis of CoCaR

Impact of BS Memory Capacity. Fig. 6 shows the results of average inference precision and average hit rate as the BS memory capacity varies. As the BS memory capacity increases from 100 MB to 500 MB, the performance of all methods improves. This is because a larger memory capacity allows BSs to cache more models, thereby satisfying more user requests. When $R = 500$ MB, CoCaR's average inference precision gap with LR is only 8.1%, outperforming other baselines by at least 46.5%. Therefore, CoCaR, with its superior decision-making mechanism, can quickly adapt to changes in memory resources, consistently outperform other baselines, and steadily converge to the optimal LR algorithm.

Impact of Popularity Change Frequency. In this experiment, we set the number of observation windows to $\Gamma = 20$ and configured the other parameters to their default values (Sec. VI-A). Fig. 7 shows how often the model popularity changes, i.e., the number of observation windows in which the DNN models' popularity changes once, affecting the algorithms' performance. CoCaR consistently achieves the best results after LR, owing to its ability to leverage BS caching results from the previous observation window and balance the impact of model loading time by switching between dynamic DNN submodels, thereby quickly caching DNN models and adapting to environmental changes. In contrast, the other baselines make independent decisions for each observation window, resulting in more fluctuating results in response to popularity changes. Even when popularity changes in every window, CoCaR's average precision exceeds other baselines by more than 51.1%, second only to the LR, while maintaining the highest hit rate.

Impact of Zipf Skewness Coefficient. Fig. 8 illustrates the performance of algorithms across Zipf skewness coefficients ranging from 0 to 1. CoCaR consistently achieves higher average precision and hit rate as Zipf skewness increases, with slight fluctuations due to random rounding. Additionally, since a higher coefficient results in more frequent requests for a few highly popular model types, the upward trend in the greedy algorithm becomes more evident. When the Zipf skewness coefficient is zero, the popularity vector is uniform, indicating that user requests are completely randomized. Despite this, CoCaR achieves favourable results by leveraging a multi-

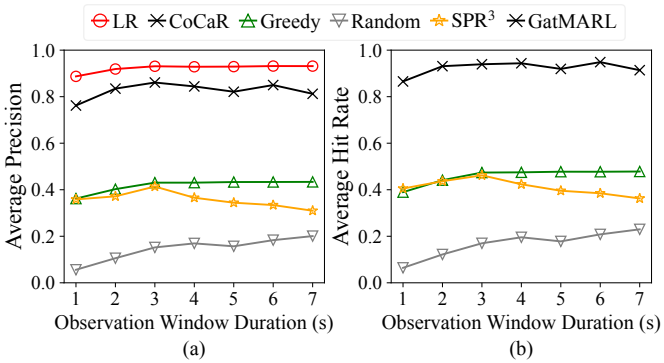

Fig. 9. Impact of different observation window duration: (a) Average inference precision; (b) Average hit rate.

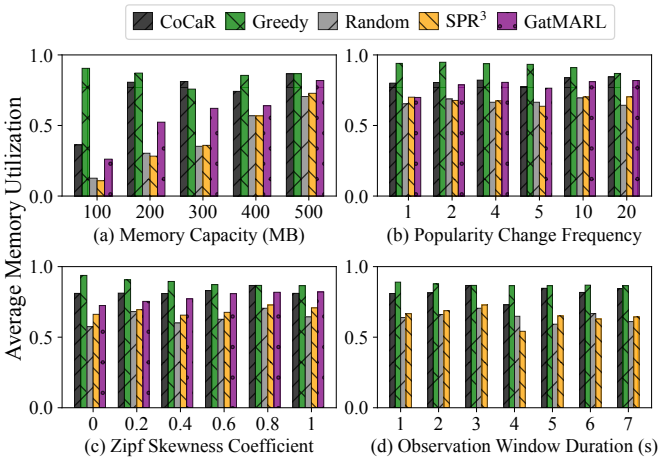

Fig. 10. Average BS memory resource utilization.

edge collaboration mechanism and quickly adjusting caching schemes via submodel switching to serve more users.

Impact of Observation Window Duration. Fig. 9 shows the impact on algorithm performance as the observation window duration is varied, while keeping the other default settings constant (Sec. VI-A). According to the parameter settings in Sec. VI-A, given the observation window duration $\Delta\tau = 1$s, there are $\Gamma = 30$ windows, each with $U = 200$ users. LR, CoCaR, and SPR^3 exhibit an initial increase followed by a decrease, achieving optimal performance at a duration of 3s. This is due to the imbalance between model loading time and observation window duration. An window that is too short significantly increases the time spent on loading models, thereby reducing the valid model service time, inference precision, and hit rate. Conversely, an overly long window may prevent the algorithm from promptly adapting to changing user requests, thus affecting performance. Additionally, fixed hyperparameters, such as the number of users during GarMARL training, limit its decision-making in this experiment.

BS Resource Utilization. Fig. 10 shows the average BS resource utilization of all algorithms under various influencing factors. Combining the experimental analysis above, CoCaR demonstrates a more effective, finer-grained utilization of BS resources, with each unit of resource usage resulting in higher

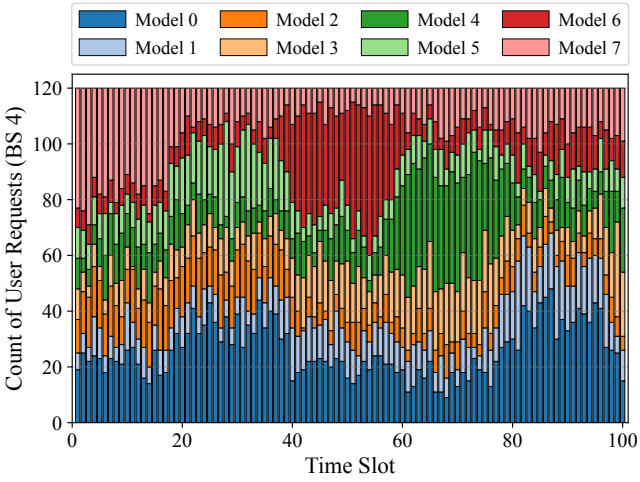

Fig. 11. Count of user requests for DNN models at BS 4 in each time.

average inference precision and hit rate, thereby improving overall service quality and user coverage. Specifically, under the experimental setting in Sec. VI-A, CoCaR can utilize at least 86% of BS memory resources. For the remaining algorithms, the coarse-grained caching scheme and the neglect of model loading time result in fewer cached DNN models and shorter valid model service time for the same BS resource utilization, ultimately leading to decreased algorithm performance.

D. Evaluation Results and Analysis of CoCaR-OL

Extended Settings. We set the duration of each time slot to $\Delta t = 0.5s$, with a total of $\mathcal{T} = 100$ time slots. In each time slot, $round = 3$ BSs are randomly selected, and a caching decision is made based on the past $\Delta T^P = 10$ user requests and the expected future gain in the next $\Delta T^F = 5$ time slots. The smoothing factor α and the discount factor γ are both set to 0.9. In each time slot, $U = 600$ users simultaneously initiate requests for model type $\mathcal{M}$. The popularity of DNN models at each BS is changed every 20 time slots, with a warm-up phase starting 5 time slots earlier to gradually adjust the popularity. Other parameters are the same as those defined in Sec. VI-A. Fig. 11 illustrates the distribution of user requests at BS 4 under the above experimental setup.

In the online scenario, we introduce the following online caching algorithms for comparison:

- **LFU [47].** In each time slot, $round$ BSs are randomly and independently selected. For each selected BS, model request frequencies over the past ΔT^P time slots at the BS and its one-hop neighbors are computed. The cached submodel of the most frequent model is enlarged, while the cached submodel of the least frequent model is gradually reduced until the memory constraint is met.
- **LFU-MAD [48].** It accounts for delayed hits in caching and adjusts the strategy by ranking content. In our experiment, ranks are based on weighted request frequency, with higher weights to more recent user requests. The caching decision is similar to the LFU in this paper.

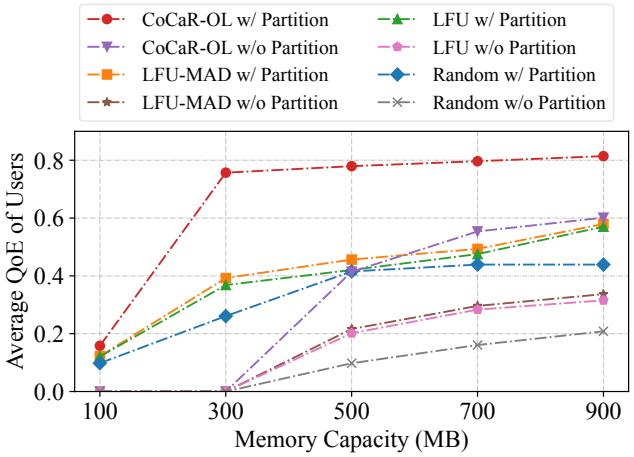

Fig. 12. Effect of BS Memory Capacity on the average user's QoE.

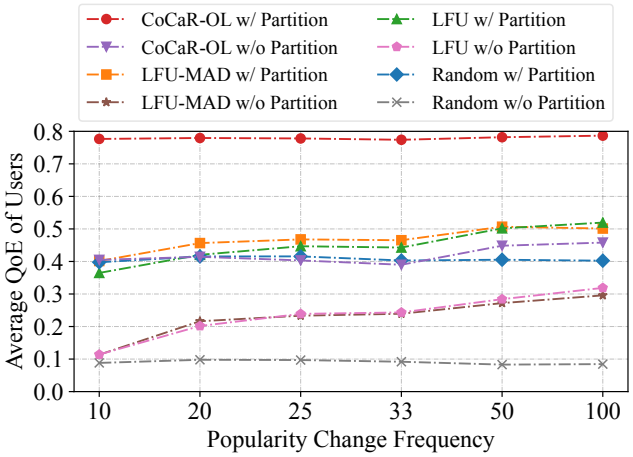

Fig. 14. Effect of popularity change frequency on the average user's QoE.

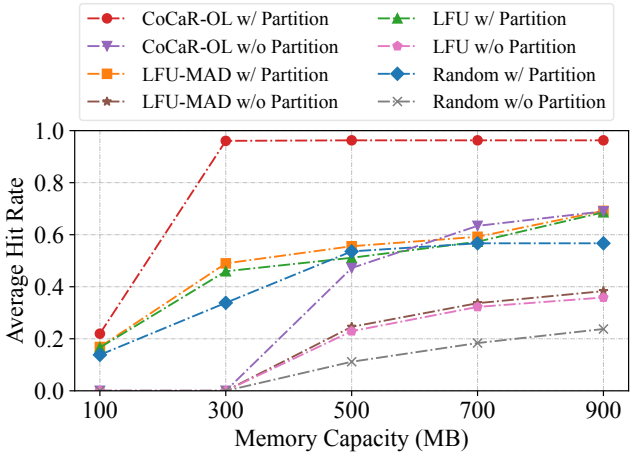

Fig. 13. Effect of BS Memory Capacity on the cache hit rate.

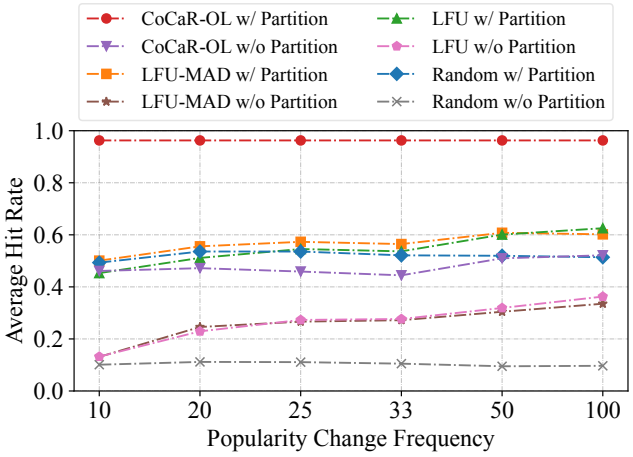

Fig. 15. Effect of popularity change frequency on the cache hit rate.

- **Random.** In each timeslot, $round$ BSs are randomly selected. At each selected BS, a cached submodel is randomly chosen and enlarged, and a combination of the remaining submodels that satisfies the memory constraint is then randomly selected for caching.

In our evaluation, for the above algorithms, only submodels of models that are not currently being downloaded can be switched, and the memory occupied by downloading submodels is considered in each decision. Additionally, we considered versions of CoCaR-OL and the above baselines without the dynamic DNN switching mechanism (i.e., model partitioning), meaning that either the model is not cached at all or the complete, original model is cached.

Effect of BS Memory Capacity. We evaluated the performance of each algorithm with cache sizes ranging from 100MB to 500 MB. As shown in Figs. 12 and 13, when the cache is small (e.g., 100 MB), only the smallest submodel of each model can be cached, resulting in low user QoE and cache hit rates across all algorithms. As the cache size increases, larger submodels can be cached, providing better service to more users, and hence improving QoE and hit rate for all algorithms. Moreover, algorithms with dynamic DNN submodel switching consistently outperform their counterparts that cache only the full original models. For instance, with a cache size of 500 MB, CoCaR-OL with model partitioning achieves a 32.3% higher user QoE than LFU-MAD with partitioning, and a 36.5% higher QoE than CoCaR-OL without partitioning. This improvement stems from our dynamic DNN-based caching mechanism, which continues serving users with current submodels while new submodels are being downloaded. These results demonstrate that the dynamic DNN caching mechanism can adapt to different caching schemes, leveraging fine-grained memory usage and enabling flexible, rapid model switching to enhance user service.

Effect of Popularity Change Frequency. Figs. 14 and 15 illustrate the impact of the frequency of DNN model popularity changes, with the popularity changing from every 10 time slots to every 100 slots. Overall, the user's QoE and cache hit rate of all algorithms gradually increase as the frequency of popularity changes decreases. Furthermore, since the proposed CoCaR-OL algorithm considers the impact of each caching decision on the global expected future gain, while others rely solely on partial information, it exhibits greater stability and consistently outperforms other algorithms regardless of popularity change

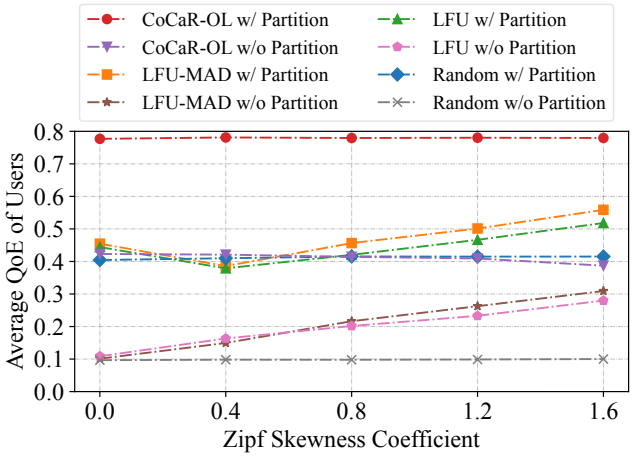

Fig. 16. Effect of the Zipf skewness coefficient on the average user's QoE.

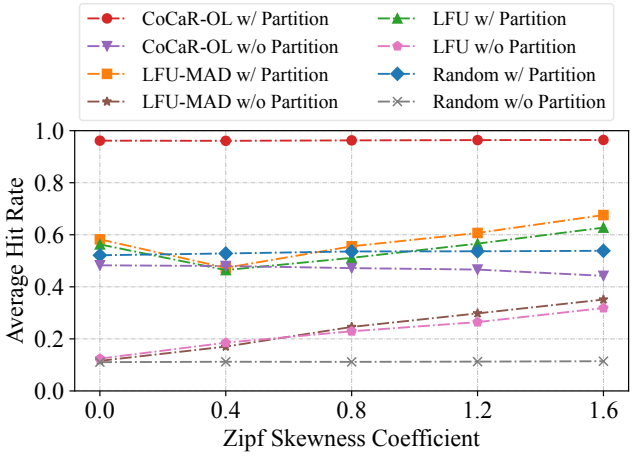

Fig. 17. Effect of the Zipf skewness coefficient on the cache hit rate.

frequency. When popularity changes every 10 time slots, LFU with model partitioning achieves 3.3% and 3.2% lower user QoE than LFU-MAD and Random, respectively. This is because LFU computes request frequencies over the past 10 time slots, so when popularity shifts abruptly, its caching decisions lag behind the new distribution. In contrast, LFU-MAD gives more weight to recent requests, enabling faster adaptation to local popularity changes.

Effect of the Zipf Skewness Coefficient. The impact of the Zipf skewness coefficient on algorithm performance is shown in Figs. 16 and 17. As the skewness coefficient increases, the proposed CoCaR-OL consistently leverages global information to generate high-quality caching decisions, maintaining stable performance. In contrast, LFU and LFU-MAD with model partitioning exhibit much weaker performance. This is because the LFU strategy relies exclusively on frequency for its decisions, disregarding other critical factors such as model size and precision. These limitations further underscore the advantages of our gain-oriented CoCaR-OL in balancing precision and latency. When the Zipf coefficient is zero, popular models change randomly, and LFU tends to switch to caching smaller submodels of different models to provide some services. As the Zipf coefficient increases to 0.4, LFU gradually caches the largest submodels of popular models, but the requests for these models may not greatly exceed those for less popular models, leading to overall performance degradation. This degradation is mitigated by a further increase in the coefficient, ultimately resulting in improved performance.

VII. Related Work

Caching & Routing in MEC. In recent years, mobile edge computing (MEC) has received increasing attention in networking. Since service caching and request routing are critical in MEC, jointly optimizing them to guarantee users' QoE remains a major challenge. Poularakis *et al.* [22] studied the joint optimization problem of service placement and request routing with multidimensional constraints and proposed an algorithm to achieve near-optimal performance. Yao *et al.* [25] proposed a graph attention-based intelligent cooperative task offloading and service caching scheme to maximize the utility of quality of service-based systems. Zhang *et al.* [23] jointly considered service caching, computation offloading, transmission, and computation resource allocation to minimize the overall computation and latency costs for all users. Yao *et al.* [33] aimed to maximize throughput under budget, accuracy, and latency constraints by jointly optimizing model caching and request routing. Chu *et al.* [19] investigated how to maximize user QoE in MEC-based networks by jointly optimizing service caching, resource allocation, and task offloading decisions. Zhao *et al.* [17] investigated how to efficiently offload dependent tasks to edge nodes with limited service caches and designed an efficient algorithm based on convex programming to solve this problem. Bi *et al.* [49] consider the problem of joint optimization of service cache locations, computation offloading decisions, and system resource allocation on a single edge server that assists mobile users in performing a series of computational tasks. However, these studies focus on caching complete services at BSs, which limits the effective utilization of BSs' resources. Additionally, they neglect the impact of service loading time on caching and routing decision-making. In practice, user requests have dynamically changing patterns and significantly impact users' QoE, and hence must be addressed.

Dynamic DNNs. Dynamic DNNs can meet various requirements under different resource constraints by adjusting the number of channels and layers. MutualNet [50] and MSDNet [51] achieve adaptive precision-latency tradeoffs by varying DNN network width and depth, respectively. Dynamic-OFA [28] provides an efficient architecture for GPUs and CPUs by using a shared backbone model that adjusts DNN width, depth, filter sizes, and input resolution. SubFlow [52] dynamically builds and executes DNN sub-networks for flexible time-bound inference and training. Smart-MOMEPS [27] leverages a multi-exit mechanism and dynamic service migration to alleviate DNN performance degradation. SN-Net [53] generates numerous new stitched networks by combining pre-trained models from a model family, enabling adaptation to different resource constraints. ESTA [54], on the other hand, uses efficient parameter fine-tuning and training-time gradient statistics

to assemble stitched models of various sizes from existing pre-trained DNNs, achieving a stable balance between precision and efficiency. Different from existing studies, we introduce dynamic DNNs into edge caching by dividing a complete DNN into multiple interrelated and switchable submodels. This approach provides a novel solution for fine-grained, resource-efficient, and flexible edge-based model caching.

VIII. Conclusion and future work

In this paper, we study the joint dynamic model caching and request routing optimization problem in MEC networks, aiming to maximize the total inference precision of user requests. Then, we perform multiple equivalent transformations and relaxation for the joint optimization problem and propose a novel random rounding and LP-based algorithm, CoCaR, to solve it. Moreover, to adapt to the online scenario where user requests are difficult to predict, we further propose CoCaR-OL as an extension of CoCaR. Theoretical analysis and experimental results show that the proposed CoCaR achieves near-optimal performance and CoCaR-OL outperforms other benchmark algorithms in online settings. Therefore, our proposed dynamic DNN-based caching mechanism effectively exploits memory resources in a fine-grained manner, flexibly adjusting caching strategies to enhance service delivery for users. In the future, we will further explore BSs' resource allocation and distributed decision-making scheme in the joint model caching and request routing problem.

Acknowledgments

The authors also thank the Big Data Computing Center of Southeast University for providing the experiment environment and computing facility. The authors are also grateful to SF Technology Co., Ltd. for their valuable discussions and technical support.

References

[1] S. Qiu, F. Dong, S. Tan, D. Shen, R. Zhou, and Q. Fan, "CoCaR: Enabling efficient dynamic DNN-based model caching and request routing in MEC," in *proc. IEEE INFOCOM*, London, United Kingdom, May, 2025, pp. 1–10.

[2] X. Hu, L. Liang, S. Li, L. Deng, P. Zuo, Y. Ji, X. Xie, Y. Ding, C. Liu, T. Sherwood *et al.*, "DeepSniffer: A DNN model extraction framework based on learning architectural hints," in *proc. ACM 25th ASPLOS*, New York, USA, Mar, 2020, pp. 385–399.

[3] T. Wang, L. Shen, Q. Fan, T. Xu, T. Liu, and H. Xiong, "Joint admission control and resource allocation of virtual network embedding via hierarchical deep reinforcement learning," *IEEE Transactions on Services Computing*, vol. 17, no. 03, pp. 1001–1015, 2024.

[4] Z. Wang, R. Zhang, S. Zhang, W. Cheng, W. Wang, and Y. Cui, "Edge-assisted adaptive configuration for serverless-based video analytics," *IEEE Transactions on Networking*, vol. 33, no. 3, pp. 1144–1159, 2025.

[5] A. Krizhevsky, I. Sutskever, and G. E. Hinton, "Imagenet classification with deep convolutional neural networks," *Communications of the ACM*, vol. 60, no. 6, pp. 84–90, 2017.

[6] G. Saon, Z. Tüske, D. Bolanos, and B. Kingsbury, "Advancing RNN transducer technology for speech recognition," in *proc. IEEE ICASSP*, Toronto, Canada, Jun, 2021, pp. 5654–5658.

[7] Z. Cao, S. Xu, H. Peng, D. Yang, and R. Zidek, "Confidence-aware reinforcement learning for self-driving cars," *IEEE Transactions on Intelligent Transportation Systems*, vol. 23, no. 7, pp. 7419–7430, 2021.

[8] Z. Ning, H. Hu, X. Wang, L. Guo, S. Guo, G. Wang, and X. Gao, "Mobile edge computing and machine learning in the internet of unmanned aerial vehicles: a survey," *ACM Computing Surveys*, vol. 56, no. 1, pp. 1–31, 2023.

[9] B. Huang, X. Liu, Y. Xiang, D. Yu, S. Deng, and S. Wang, "Reinforcement learning for cost-effective IoT service caching at the edge," *Journal of Parallel and Distributed Computing*, vol. 168, pp. 120–136, 2022.

[10] R. Zhang, R. Zhou, Y. Wang, H. Tan, and K. He, "Incentive mechanisms for online task offloading with privacy-preserving in UAV-assisted mobile edge computing," *IEEE/ACM Transactions on Networking*, vol. 32, no. 3, pp. 2646–2661, 2024.

[11] X. Wang, J. Ye, and J. C. Lui, "Mean field graph based D2D collaboration and offloading pricing in mobile edge computing," *IEEE/ACM Transactions on Networking*, vol. 32, no. 1, pp. 491–505, 2023.

[12] T. Ren, Z. Hu, H. He, J. Niu, and X. Liu, "FEAT: Towards fast environment-adaptive task offloading and power allocation in MEC," in *proc. IEEE INFOCOM*, New York City, USA, May, 2023, pp. 1–10.

[13] Z. Xu, L. Zhou, S. C.-K. Chau, W. Liang, H. Dai, L. Chen, W. Xu, Q. Xia, and P. Zhou, "Near-optimal and collaborative service caching in mobile edge clouds," *IEEE Transactions on Mobile Computing*, vol. 22, no. 7, pp. 4070–4085, 2023.

[14] H. Tan, Y. Wang, C. Zhang, G. Li, H. Du, Z. Han, S. H.-C. Jiang, and X.-Y. Li, "Asymptotically tight approximation for online file caching with delayed hits and bypassing," *IEEE Transactions on Networking*, vol. 33, no. 4, pp. 1886–1899, 2025.

[15] R. Zhou, X. Wu, H. Tan, and R. Zhang, "Two time-scale joint service caching and task offloading for UAV-assisted mobile edge computing," in *proc. IEEE INFOCOM*, London, United Kingdom, May, 2022, pp. 1189–1198.

[16] M. Li, Z. Han, C. Zhang, R. Zhou, Y. Liu, and H. Tan, "Dynamic resource allocation for deep learning clusters with separated compute and storage," in *proc. IEEE INFOCOM*, New York City, USA, May, 2023, pp. 1–10.

[17] G. Zhao, H. Xu, Y. Zhao, C. Qiao, and L. Huang, "Offloading tasks with dependency and service caching in mobile edge computing," *IEEE Transactions on Parallel and Distributed Systems*, vol. 32, no. 11, pp. 2777–2792, 2021.

[18] L. Qin, H. Lu, Y. Lu, C. Zhang, and F. Wu, "Joint optimization of base station clustering and service caching in user-centric MEC," *IEEE Transactions on Mobile Computing*, vol. 23, no. 5, pp. 6455–6469, 2023.

[19] W. Chu, X. Jia, Z. Yu, J. C. Lui, and Y. Lin, "Joint service caching, resource allocation and task offloading for MEC-based networks: A multi-layer optimization approach," *IEEE Transactions on Mobile Computing*, vol. 23, no. 4, pp. 2958–2975, 2024.

[20] J. Ren, W. Zhang, H. Wang, D. Yang, S. Wang, H. Zhang, and S. Cui, "Toward deterministic wide-area networks via deadline-aware routing and scheduling," *IEEE Transactions on Networking*, vol. 33, no. 4, pp. 1762–1778, 2025.

[21] T. Wang, Q. Fan, C. Wang, L. Ding, N. J. Yuan, and H. Xiong, "FlagVNE: A flexible and generalizable RL framework for network resource allocation," in *proc. 33rd IJCAI*, 2024.

[22] K. Poularakis, J. Llorca, A. M. Tulino, I. Taylor, and L. Tassiulas, "Service placement and request routing in MEC networks with storage, computation, and communication constraints," *IEEE/ACM Transactions on Networking*, vol. 28, no. 3, pp. 1047–1060, 2020.

[23] G. Zhang, S. Zhang, W. Zhang, Z. Shen, and L. Wang, "Joint service caching, computation offloading and resource allocation in mobile edge computing systems," *IEEE Transactions on Wireless Communications*, vol. 20, no. 8, pp. 5288–5300, 2021.

[24] M. Tang and V. W. Wong, "Deep reinforcement learning for task offloading in mobile edge computing systems," *IEEE Transactions on Mobile Computing*, vol. 21, no. 6, pp. 1985–1997, 2020.

[25] Z. Yao, S. Xia, Y. Li, and G. Wu, "Cooperative task offloading and service caching for digital twin edge networks: A graph attention multi-agent reinforcement learning approach," *IEEE Journal on Selected Areas in Communications*, vol. 41, no. 11, pp. 3401–3413, 2023.

[26] K. Peng, L. Wang, J. He, C. Cai, and M. Hu, "Joint optimization of service deployment and request routing for microservices in mobile edge computing," *IEEE Transactions on Services Computing*, vol. 17, no. 3, pp. 1016–1028, 2024.

[27] P. Wang, T. Ouyang, G. Liao, J. Gong, S. Yu, and X. Chen, "Edge intelligence in motion: Mobility-aware dynamic DNN inference service migration with downtime in mobile edge computing," *Journal of Systems Architecture*, vol. 130, p. 102664, 2022.

[28] W. Lou, L. Xun, A. Sabet, J. Bi, J. Hare, and G. V. Merrett, "Dynamic-OFA: Runtime DNN architecture switching for performance scaling on heterogeneous embedded platforms," in *proc. IEEE/CVF CVPR*, Nashville, USA, Jun, 2021, pp. 3110–3118.

[29] C. Singhal, Y. Wu, F. Malandrino, M. Levorato, and C. F. Chiasserini, "Distributing inference tasks over interconnected systems through dy-

namic DNNs," *IEEE Transactions on Networking*, vol. 33, no. 4, pp. 1717–1730, 2025.
[30] Z. Liu, H. Du, J. Lin, Z. Gao, L. Huang, S. Hosseinalipour, and D. Niyato, "Dnn partitioning, task offloading, and resource allocation in dynamic vehicular networks: A lyapunov-guided diffusion-based reinforcement learning approach," *IEEE Transactions on Mobile Computing*, vol. 24, no. 3, pp. 1945–1962, 2025.
[31] H. Li, X. Li, Q. Fan, Q. He, X. Wang, and V. C. Leung, "Distributed dnn inference with fine-grained model partitioning in mobile edge computing networks," *IEEE Transactions on Mobile Computing*, vol. 23, no. 10, pp. 9060–9074, 2024.
[32] S. Qiu, Q. Fan, X. Li, X. Zhang, G. Min, and Y. Lyu, "OA-Cache: Oracle approximation-based cache replacement at the network edge," *IEEE Transactions on Network and Service Management*, vol. 20, no. 3, pp. 3177–3189, 2023.
[33] M. Yao, L. Chen, Y. Wu, and J. Wu, "Loading cost-aware model caching and request routing in edge-enabled wireless sensor networks," *The Computer Journal*, vol. 66, no. 10, pp. 2409–2425, 2023.
[34] M. Yao, L. Chen, J. Zhang, J. Huang, and J. Wu, "Loading cost-aware model caching and request routing for cooperative edge inference," in *proc. IEEE ICC*, Seoul, Korea, Republic of, May, 2022, pp. 2327–2332.
[35] A. Dosovitskiy, L. Beyer, A. Kolesnikov, D. Weissenborn, X. Zhai, T. Unterthiner, M. Dehghani, M. Minderer, G. Heigold, S. Gelly, J. Uszkoreit, and N. Houlsby, "An image is worth 16x16 words: Transformers for image recognition at scale," in *proc. 9th ICLR*. Virtual Event: OpenReview.net, May, 2021.
[36] T. Ouyang, K. Zhao, G. Hong, X. Zhang, Z. Zhou, and X. Chen, "Dynamic edge-centric resource provisioning for online and offline services co-location via reactive and predictive approaches," *IEEE Transactions on Networking*, vol. 33, no. 3, pp. 1388–1403, 2025.
[37] L. Chen, C. Shen, P. Zhou, and J. Xu, "Collaborative service placement for edge computing in dense small cell networks," *IEEE Transactions on Mobile Computing*, vol. 20, no. 2, pp. 377–390, 2019.
[38] Y. T. Lee and A. Sidford, "Efficient inverse maintenance and faster algorithms for linear programming," in *proc. IEEE 56th FOCS*, Berkeley, USA, Oct, 2015, pp. 230–249.
[39] M. Mitzenmacher and E. Upfal, *Probability and computing: Randomization and probabilistic techniques in algorithms and data analysis*. Cambridge university press, 2017.
[40] P. Wang, S. Li, Y. Han, F. Ye, and Q. Zhang, "Fast-response edge caching scheme for graph data," *IEEE Transactions on Networking*, vol. 33, no. 4, pp. 1962–1975, 2025.
[41] Z. Tang, F. Mou, J. Lou, W. Jia, Y. Wu, and W. Zhao, "Multi-user layer-aware online container migration in edge-assisted vehicular networks," *IEEE/ACM Transactions on Networking*, vol. 32, no. 2, pp. 1807–1822, 2023.
[42] P. Erdős, A. Rényi *et al.*, "On the evolution of random graphs," *Publ. math. inst. hung. acad. sci*, vol. 5, no. 1, pp. 17–60, 1960.
[43] Z. Liu, Y. Lin, Y. Cao, H. Hu, Y. Wei, Z. Zhang, S. Lin, and B. Guo, "Swin Transformer: Hierarchical vision transformer using shifted windows," in *proc. IEEE/CVF ICCV*, Virtual Event, Oct, 2021, pp. 10 012–10 022.
[44] A. Krizhevsky, G. Hinton *et al.*, "Learning multiple layers of features from tiny images," *Master's thesis, Department of Computer Science, University of Toronto*, 2009.
[45] K. Shanmugam, N. Golrezaei, A. G. Dimakis, A. F. Molisch, and G. Caire, "FemtoCaching: Wireless content delivery through distributed caching helpers," *IEEE Transactions on Information Theory*, vol. 59, no. 12, pp. 8402–8413, 2013.
[46] Z. Yao, S. Xia, Y. Li, and G. Wu, "Cooperative task offloading and service caching for digital twin edge networks: A graph attention multi-agent reinforcement learning approach," *IEEE Journal on Selected Areas in Communications*, vol. 41, no. 11, pp. 3401–3413, 2023.
[47] J. Dilley and M. Arlitt, "Improving proxy cache performance: Analysis of three replacement policies," *IEEE Internet Computing*, vol. 3, no. 6, pp. 44–50, 2002.
[48] N. Atre, J. Sherry, W. Wang, and D. S. Berger, "Caching with delayed hits," in *proc. ACM SIGCOMM*, Virtual Event USA, Jul, 2020, pp. 495–513.
[49] S. Bi, L. Huang, and Y. J. A. Zhang, "Joint optimization of service caching placement and computation offloading in mobile edge computing systems," *IEEE Transactions on Wireless Communications*, vol. 19, no. 7, pp. 4947–4963, 2020.
[50] T. Yang, S. Zhu, C. Chen, S. Yan, M. Zhang, and A. Willis, "MutualNet: Adaptive convnet via mutual learning from network width and resolution," in *proc. ECCV*. Glasgow, UK: Springer, Aug, 2020, pp. 299–315.
[51] G. Huang, D. Chen, T. Li, F. Wu, L. van der Maaten, and K. Weinberger, "Multi-scale dense networks for resource efficient image classification," in *proc. ICLR*. Vancouver, Canada: OpenReview.net, Apr, 2018.
[52] S. Lee and S. Nirjon, "Subflow: A dynamic induced-subgraph strategy toward real-time DNN inference and training," in *proc. IEEE RTAS*, Sydney, Australia, Apr, 2020, pp. 15–29.
[53] Z. Pan, J. Cai, and B. Zhuang, "Stitchable neural networks," in *proc. IEEE/CVF CVPR*, Vancouver, Canada, Jun, 2023, pp. 16 102–16 112.
[54] H. He, Z. Pan, J. Liu, J. Cai, and B. Zhuang, "Efficient stitchable task adaptation," in *proc. IEEE/CVF CVPR*, Seattle, USA, Jun, 2024, pp. 28 555–28 565.