TrimCaching: Parameter-Sharing Edge Caching for AI Model Downloading

Guanqiao Qu, *Graduate Student Member, IEEE*, Zheng Lin, *Graduate Student Member, IEEE*, Qian Chen, *Member, IEEE*, Jian Li, *Senior Member, IEEE*, Fangming Liu, *Senior Member, IEEE*, Xianhao Chen, *Member, IEEE*, and Kaibin Huang, *Fellow, IEEE*

***Abstract*—Next-generation mobile networks are expected to facilitate fast AI model downloading to end users. By caching models on edge servers, mobile networks can deliver models to end users with low latency, resulting in a paradigm of *edge model caching*. In this paper, we develop a novel model placement framework, called parameter-sharing model caching (TrimCaching). TrimCaching exploits the key observation that a wide range of AI models, such as convolutional neural networks or large language models, can share a significant proportion of parameter blocks containing reusable knowledge, thereby improving storage efficiency. To this end, we formulate a parameter-sharing model placement problem to maximize the cache hit ratio in multi-edge wireless networks by balancing the fundamental tradeoff between storage efficiency and service latency. We show that the formulated problem is a submodular maximization problem with submodular constraints, for which no polynomial-time approximation algorithm exists. To tackle this challenge, we study an important special case, where a small fixed number of parameter blocks are shared across models, which often holds in practice. In such a case, a polynomial-time algorithm with a $(1-\epsilon)/2$-approximation guarantee is developed. Subsequently, we address the original problem for the general case by developing a greedy algorithm. Simulation results demonstrate that the proposed TrimCaching framework significantly improves the cache hit ratio compared with state-of-the-art content caching without exploiting shared parameters in AI models.**

***Index Terms*—Edge AI model caching, edge computing, edge intelligence, 6G, model downloading.**

Received 23 April 2024; revised 7 June 2025, 10 January 2026, and 6 March 2026; accepted 14 April 2026; approved by IEEE TRANSACTIONS ON NETWORKING Editor C. Brinton. Date of publication 30 April 2026; date of current version 20 May 2026. This work was supported in part by the Research Grants Council of Hong Kong under Grant 27213824 and Grant CRS HKU702/2. The work of Fangming Liu was supported in part by the Major Key Project of the Peng Cheng Laboratory (PCL) under Grant PCL2025A10 and Grant PCL2024A06 and in part by Shenzhen Science and Technology Program under Grant RCJC20231211085918010. The work of Kaibin Huang was supported in part by the Research Grants Council of Hong Kong Special Administrative Region, China, under a Fellowship Award under Grant HKU RFS2122-7S04; in part by NSFC/RGC CRS under Grant CRS_HKU702/24; in part by the Areas of Excellence Scheme Grant under Grant AoE/E-601/22-R; in part by the Collaborative Research Fund under Grant C1009-22G; in part by Grant 17212423 and Grant 17304925; and in part by the Croucher Senior Fellowship and Shenzhen-Hong Kong-Macau Technology Research Programme (Type C) under Grant SGDX20230821091559018. A preliminary version of this work [DOI: 10.1109/ICDCS60910.2024.00013] has been accepted by the 44th IEEE International Conference on Distributed Computing Systems (ICDCS 2024). *(Corresponding author: Xianhao Chen.)*

Guanqiao Qu, Zheng Lin, Qian Chen, Xianhao Chen, and Kaibin Huang are with the Department of Electrical and Computer Engineering, The University of Hong Kong, Hong Kong SAR, China (e-mail: gqqu@eee.hku.hk; linzheng@eee.hku.hk; qchen@eee.hku.hk; xchen@eee.hku.hk; huangkb@eee.hku.hk).

Jian Li is with the School of Cyber Science and Technology, University of Science and Technology of China, Hefei, Anhui 230027, China (e-mail: lijian9@ustc.edu.cn).

Fangming Liu is with the Peng Cheng Laboratory, Shenzhen 518000, China, and also with Huazhong University of Science and Technology, Wuhan 430074, China (e-mail: fangminghk@gmail.com).

This article has supplementary downloadable material available at https://doi.org/10.1109/TON.2026.3688811, provided by the authors.

Digital Object Identifier 10.1109/TON.2026.3688811

I. INTRODUCTION

IN THE era of Artificial Intelligence of Things (AIoT), there is a growing trend of pushing Artificial Intelligence (AI) services from the cloud to local devices, enabling diverse AI-empowered applications [2], [3], [4]. Aligned with this trend, on-device AI inference becomes prevalent due to increasingly powerful mobile AI chips, substantial privacy benefits, and low response time. For example, large language model (LLM)-enabled mobile health requires access to personal health information to generate customized advice [5], [6]; home robots must perceive private indoor environments to make informed decisions [7]. In such scenarios, on-device inference is often preferred, as uploading sensitive personal data to cloud or edge servers may be prohibited due to privacy concerns and data protection regulations [8], [9].

To effectively support on-device inference, AI model downloading will become a key component of next-generation mobile networks, as recognized by 3GPP [10]. On-device inference demands *real-time*, *frequent* 6G model downloading services for several reasons. First, due to the sheer number of AI applications and the ever-growing size of AI models (e.g., Google's on-device LLM Gemini Nano-2 with 3.25 billion parameters), it is impractical for users to prestore all needed models on mobile devices. Instead, they can delete infrequently-used models from local storage and only fetch them on demand from 6G mobile networks. Second, real-time model downloading also enables users to access context-aware AI services when entering a new region [10], such as downloading region-specific autonomous driving models or augmented reality models optimized for local conditions [10], [11], [12], [13]. At last, users should download the latest model versions periodically from cloud/edge learning (e.g., federated learning [14], [15]), as models continuously adapt to new data. Consequently, supporting real-time model downloading is essential to deliver storage-efficient, customized, and evolving AI for mobile users anywhere and anytime.

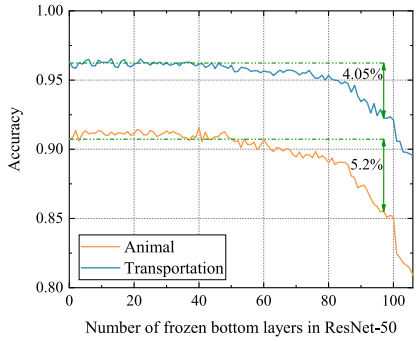


Fig. 1. Inference accuracy vs. the number of frozen bottom layers in fine-tuned ResNet-50. We fine-tune a ResNet-50 [25], pre-trained on CIFAR100 [26], for two downstream tasks: "transportation" classification and "animal" classification. Specifically, the classes "airplane", "automobile", "ship", and "truck" in CIFAR10 [26] are grouped into the superclass "transportation", while the classes "bird", "cat", "deer", "dog", "frog", and "horse" are grouped into the superclass "animal".

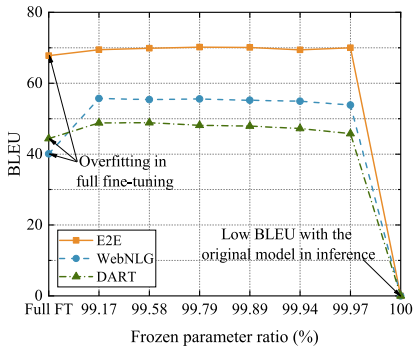


Fig. 2. BLEU vs. the frozen parameter ratio of fine-tuned GPT-2 medium models. We employ LoRA to fine-tune a pre-trained GPT-2 medium model on various datasets, including E2E [27], WebNLG [28], and DART [29]. BLEU, a widely used metric in evaluating the quality of the text generated by machine translation, is adopted here, with higher scores indicating that machine translation results are closer to those of professional human translations.

Conventional model downloading features the delivery of AI models from the remote cloud to local devices. The excessive downloading latency not only results in poor user experience but also renders it inapplicable for latency-sensitive services [16], [17]. For example, according to 3GPP, autonomous vehicles and robotic applications require AI model downloading to be completed within one second [10]. Given the large sizes of models and the limited bandwidth of remote cloud centers, such fast model downloading may only be achieved by placing AI models in edge networks closer to users, giving rise to the paradigm of *edge model caching*. However, unlike cloud centers, edge servers have limited storage capacity and can cache only a subset of popular models. To enhance model downloading performance, one fundamental research question, therefore, is model placement, e.g., how can we optimally place AI models on edge servers to enhance model downloading performance?

In this paper, we address a novel AI model placement problem by exploiting parameter sharing to improve model storage efficiency at the network edge. A broad range of AI models, such as convolutional neural networks (CNNs) or LLMs, can share a significant proportion of parameters, as they can be derived from the same pre-trained models [18], [19]. For example, bottom-layer freezing is a classic technique in transfer learning and multi-task learning, since the bottom layers in deep neural networks, such as convolution layers, usually share common knowledge reusable for different downstream tasks [20], [21], [22], [23]. In the era of LLMs, parameter-efficient fine-tuning (PEFT), such as LoRA [24], has emerged as an effective approach to adapt foundation models for downstream applications by freezing a large proportion of parameters, thereby saving computing and memory resources. For example, LoRA freezes all parameters in the pre-trained LLMs and introduces only a small number of trainable parameters for fine-tuning, typically less than 1% of the total, implying over 99% of the parameters remain unchanged during fine-tuning. As shown in Fig. 1, the inference accuracy of a fine-tuned ResNet-50 degrades only slightly as the number of bottom layers frozen from the pre-trained model increases. Even when the first 90% of trainable layers, up to layer 97, are frozen, the average accuracy degradation is only about 4.7%, compared with the full-layer fine-tuning. As for LLMs, Fig. 2 illustrates that the BLEU scores of various fine-tuned GPT-2 medium models exhibit slight variation across different frozen parameter ratios. The fine-tuned models maintain satisfactory performance, even when up to 99.97% of the parameters are frozen from the pre-trained model. In a nutshell, downstream models can share a significant proportion of parameters from the same pre-trained model, inherently making storage more efficient.

By taking advantage of the aforementioned salient property, we will design a parameter-sharing model caching (TrimCaching) framework for edge model caching. Specifically, given a set of wireless edge servers, we address the problem of placing models with shared parameters on edge servers to maximize the cache hit ratio for AI model downloading [30], [31], [32]. While this problem clearly falls into classical content placement problems, the shared parameters distinguish our scheme from traditional placement schemes for general content. In particular, when placing models with more shared parameters on an edge server, the storage becomes more efficient. However, this notable advantage also introduces a *fundamental challenge*: the resulting submodular constraints make existing content placement schemes inapplicable, which lack theoretical guarantees in our scenario. The main contributions of this paper are summarized as follows.

1) We define the parameter-sharing model caching problem. In multi-edge scenarios, we formulate the model placement problem to maximize the cache hit ratio (e.g., the ratio of downloading requests that can be served within delay requirements) under the storage capacity constraints of edge servers.
2) We show that the resulting problem is a submodular maximization problem with submodular constraints. After problem mapping, we conclude that there is no polynomial-time algorithm to solve it with a constant approximation guarantee due to the submodular constraints resulting from shared parameter blocks.
3) We investigate a special case of the proposed problem, where the number of shared parameter blocks is

independent of the problem scale (e.g., the scale of the AI model library). We argue this special case often holds in reality. We develop a successive greedy algorithm and a rounding dynamic programming (DP)-based algorithm to obtain a $(1-\epsilon)/2$-approximate solution with polynomial time complexity.

4) We design a greedy-based algorithm for the original problem for the general case. Although a constant approximation guarantee cannot be achieved as alluded to earlier, we show that the algorithm is efficient and effective through performance evaluation.

The rest of this paper is organized as follows. Section II reviews the related work. Section III elaborates on the system model and the TrimCaching framework. The problem formulation is presented in Section IV. The model placement algorithm is developed for the special case in Section V and for the general case in Section VI. Section VII presents the simulation results, and Section VIII concludes the paper.

II. Related Work

Content caching in wireless edge networks has attracted significant attention since it brings content closer to end users, leading to the field of edge caching [33], [34]. Specifically, popular files can be pre-cached on wireless edge servers, such as small base stations [35], thereby allowing users to download files with reduced latency. To this end, one fundamental research problem in edge caching is the content placement problem: how to place content on edge servers to enhance the quality of experience (QoE) of end users [36], [37]. Due to the overlapping coverage of wireless edge servers, users can download content from any of the nearby edge servers that cache the requested content. Moreover, since content items are typically assumed to be independent in conventional edge caching, the storage constraints are naturally knapsack constraints. These characteristics result in a class of QoE maximization problems that are typically identified as submodular maximization problems with knapsack constraints [30], [31], [32]. Simple greedy methods are usually effective in solving such problems, providing approximation guarantees in polynomial time [30]. However, in our work, due to the parameter sharing across AI models, the shared parameters need to be cached only once on a server, introducing submodularity into the storage constraints. For this reason, when adapting the aforementioned content placement strategies, e.g., greedy algorithms [30], to our case, no theoretical constant approximation guarantee can be achieved.

Caching AI models in distributed wireless edge networks can facilitate model delivery for inference and training [38], [39], [40], [41], [42]. By enabling end users to download required models from edge servers, this approach significantly reduces service latency [43], [44], [45]. Analogous to conventional edge content caching, existing studies on edge model caching primarily focus on optimizing model placement to enhance user QoE under the storage constraints of edge servers. However, these works do not consider parameter sharing among AI models when making model placement decisions. Although Wu et al. exploit shared parameters in AI models to develop a multi-user model downloading method [46], this work focuses on designing a multicasting scheme rather than model placement. To the best of our knowledge, this work is the first to define the parameter-sharing model placement problem, identify its mathematical properties, and develop corresponding solution approaches.

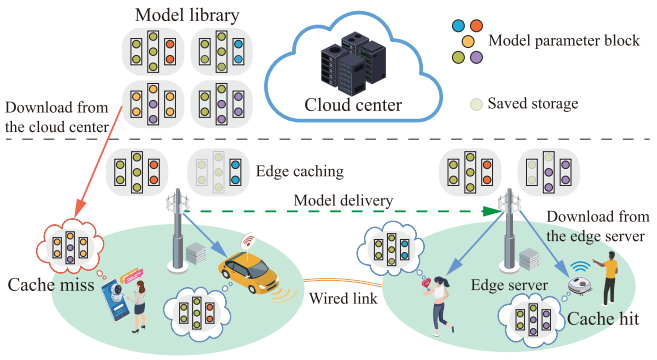


Fig. 3. The TrimCaching framework in a multi-edge scenario.

III. Parameter-Sharing Model Caching Framework

To enhance model caching efficiency, this section proposes the TrimCaching framework, including the edge network scenario, parameter-sharing model library, storage constraints, and the formulation of end-to-end (E2E) latency.

A. Network Scenario

We consider a multi-edge multi-user scenario in wireless edge networks, as illustrated in Fig. 3. Let $\mathcal{M} = \{1,\ldots,m,\ldots,M\}$ denote the set of interconnected wireless edge servers (e.g., base stations), with a set of users $\mathcal{K} = \{1,\ldots,k,\ldots,K\}$ distributed across their coverage areas. To support fast AI model downloading services for inference, these edge servers collectively cache a set of AI models $\mathcal{I} = \{1,\ldots,i,\ldots,I\}$. User k requests to download model i for performing local inference with probability $p_{k,i}$.[1] We use $\bar{T}_{k,i}$ to denote the E2E latency requirement of user k for consuming model i, which consists of model downloading and local inference latency. The frequently used notations are summarized in Table I.

B. Parameter-Sharing Model Library

We consider parameter sharing among models in $\mathcal{I}$. Given that AI models can contain billions of parameters, a *parameter block* refers to a set of parameters, which reduces problem complexity. A parameter block can refer to a layer in a CNN, a block in a transformer, a set of low-dimensional trainable parameters in PEFT (e.g., the LoRA technique), and even an entire backbone network, and so on, depending on how parameters are grouped. Let $\mathcal{J}$ denote the set of parameter blocks of models in $\mathcal{I}$. Furthermore, let $\mathcal{J}_i$ denote the set of parameter blocks of model i, and let $\mathcal{I}_j$ denote the set

[1]In practice, $p_{k,i}$ can be obtained by monitoring and analyzing user historical traffic [47].

TABLE I
FREQUENTLY USED NOTATIONS

Symbol	Description
$u(m,i)$, $\dot{u}(m,i)$	Number of cache hits and rounded number of cache hits of placing model i on edge server m.
$U(\cdot)$	Cache hit ratio of all edge servers.
$\hat{U}_m(\cdot)$	Cache hit ratio of edge server m in the special case.
$\mathbf{X}$, $\mathbf{X}_m$	Model placement decision for all edge servers and for edge server m in the general case.
$\hat{\mathbf{X}}_m$, $\hat{\mathbf{X}} = \bigcup_{m\in\mathcal{M}} \hat{\mathbf{X}}_m$	Model placement decision for edge server m and for all edge servers in the special case.
$\mathbf{X}^*$, $\hat{\mathbf{X}}_m^*$	The optimal solution to $\mathcal{P}1.1$ and $\mathcal{P}2.1_m$, respectively.
$\mathcal{M}$, $\mathcal{M}_k$	Set of all edge servers and set of edge servers covering user k.
$\mathcal{K}$	Set of users.
$\mathcal{I}$, $\mathcal{I}_j$	Set of AI models and set of models containing parameter block j.
$\hat{\mathcal{I}}_m$	Set of AI models cached on edge server m in the special case.
$\mathcal{J}$, $\mathcal{J}_i$, $\mathcal{J}^{\text{sh}}$	Set of parameter blocks, set of parameter blocks of model i, and set of shared parameter blocks.
$p_{k,i}$, $\bar{T}_{k,i}$	Request probability and E2E latency requirement of user k for model i.
$T_{m,k,i}$	Latency for user k to download model i from edge server m and perform on-device inference.
Q_m, $g_m(\cdot)$	Storage capacity and storage usage of edge server m.
D_i, D'_j	Size of model i and size of parameter block j.
$\bar{C}_{m,k}$, $C_{m,m'}$	Expected downloading data rate between edge server m and user k, and transmission data rate between edge server m and m'.

of models in $\mathcal{I}$ that contain parameter block j. It is worth noting that a parameter block can be exclusive to one model or shared by multiple models. We formally define the concept of a shared parameter block.

Definition 1 (Shared Parameter Block): A parameter block j is a shared parameter block if it is contained in more than one AI model in model set $\mathcal{I}$, i.e., $\left|\mathcal{I}_j\right| \geq 2$.

Furthermore, we denote $\mathcal{J}^{\text{sh}} = \left\{j \mid \left|\mathcal{I}_j\right| \geq 2\right\}$ by the set of the shared parameter blocks across the models in $\mathcal{I}$. If a parameter block corresponds to a layer, a shared parameter block is also referred to as a *shared layer*. Conversely, a parameter block is called a *specific parameter block* if $\left|\mathcal{I}_j\right| = 1$.

C. Storage Constraints

Let a binary variable $x_{m,i}$ indicate the placement status of model i on edge server m, where $x_{m,i} = 1$ indicates that model i is cached on edge server m. Considering the storage capacity of edge servers, $x_{m,i}$ satisfies that

$$g_m(\mathbf{X}_m) = \sum_{j\in\mathcal{J}} D'_j \left[1 - \prod_{i\in\mathcal{I}_j}\left(1 - x_{m,i}\right)\right] \leq Q_m, \ \forall m \in \mathcal{M}, \quad (1)$$

where $\mathbf{X}_m = \left\{x_{m,i} \mid i \in \mathcal{I}\right\}$ denotes the model placement decision for edge server m, D'_j is the size of parameter block j, and Q_m is the storage capacity of edge server m. Here, $1 - \prod_{i\in\mathcal{I}_j}\left(1 - x_{m,i}\right) = 1$ implies that parameter block j is stored only once on edge server m if it is contained by more than one model cached on the server, thereby enhancing storage efficiency.

D. E2E Latency

Considering user k requests model i, two cases of model downloading exist.

- **Downloading from associated edge servers**: User k first sends a model request to its associated edge servers in $\mathcal{M}_k$, where $\mathcal{M}_k$ represents the set of edge servers covering user k. If there exists an edge server $m \in \mathcal{M}_k$ that caches model i, and the latency of downloading the model from server m and performing on-device inference does not exceed the latency requirement $\bar{T}_{k,i}$, then a cache hit occurs.
- **Downloading from non-associated edge servers**: If none of the edge servers in $\mathcal{M}_k$ caches model i, the model will be fetched from another edge server m in $\mathcal{M} \setminus \mathcal{M}_k$ where the model is cached. Specifically, the model is delivered from edge server m to an edge server m' in $\mathcal{M}_k$ and then to user k. A cache hit occurs if the E2E latency (including edge-to-edge, edge-to-user, and on-device inference latency) meets the latency requirement $\bar{T}_{k,i}$.

The expected downloading data rate between edge server m and its associated user k is given by

$$\bar{C}_{m,k} = \bar{B}_{m,k}\log_2\left(1 + \frac{\bar{P}_{m,k}\gamma_0 d_{m,k}^{-\alpha_0}}{n_0 \bar{B}_{m,k}}\right), \quad (2)$$

where γ_0 is the antenna-related factor, α_0 is the path loss factor, $d_{m,k}$ is the distance between user k and edge server m, $\bar{P}_{m,k}$ and $\bar{B}_{m,k}$ are the expected transmit power and spectrum bandwidth of edge server m allocated to user k, respectively, and n_0 is the power spectral density of the additive white Gaussian noise. Besides, we assume the transmission data rate between edge server m and m' is a constant and is denoted as $C_{m,m'}$.

We denote $T_{m,k,i}$ as the E2E latency of user k, including model downloading latency and on-device inference latency, when downloading model i from edge server m, which is given by

$$T_{m,k,i} = \begin{cases} \dfrac{D_i}{\bar{C}_{m,k}} + t_{k,i}, & \text{if } m \in \mathcal{M}_k, \\ \min\limits_{m'\in\mathcal{M}_k}\left(\dfrac{D_i}{C_{m,m'}} + \dfrac{D_i}{\bar{C}_{m',k}}\right) + t_{k,i}, & \text{if } m \notin \mathcal{M}_k, \end{cases} \quad (3)$$

where D_i is the size of model i and $t_{k,i}$ is the inference latency of user k using model i.

E. Design Objective

The TrimCaching framework aims to judiciously place AI models on edge servers to serve as many user requests as possible. For cache misses on edge servers, user requests can be forwarded to the cloud center for fetching the model. However, since downloading models from the cloud can be much slower, our policy design aims to maximize the cache hit ratio, which is a common design objective in the literature [48], [49]. Similar to prior works [50], [51], the cache hit ratio of model caches across edge servers in $\mathcal{M}$ is given by

$$U(\mathbf{X}) = \frac{\sum\limits_{k\in\mathcal{K}}\sum\limits_{i\in\mathcal{I}} p_{k,i}\left[1 - \prod\limits_{m\in\mathcal{M}}\left(1 - x_{m,i}\mathbb{I}_1(m,k,i)\right)\right]}{\sum\limits_{k\in\mathcal{K}}\sum\limits_{i\in\mathcal{I}} p_{k,i}}, \quad (4)$$

which captures the expected fraction of user model requests that can be served with the cached models on edge servers within the users' latency requirements. Here, $\mathbf{X} = \{x_{m,i} \mid m \in \mathcal{M}, i \in \mathcal{I}\}$ represents the model caching decisions. Besides, $\mathbb{I}_1(m,k,i)$ is an indicator function, which is given by

$$\mathbb{I}_1(m,k,i) = \mathbb{I}_{\{T_{m,k,i} \le \bar{T}_{k,i}\}}, \tag{5}$$

where $\mathbb{I}_1(m,k,i) = 1$ if and only if $T_{m,k,i} \le \bar{T}_{k,i}$. Consequently, $1 - \prod_{m \in \mathcal{M}} \left(1 - x_{m,i}\mathbb{I}_1(m,k,i)\right) = 1$ indicates that at least one edge server with model i in $\mathcal{M}$ can serve user k within the latency constraint.

IV. Cache Hit Ratio Maximization Problem

This section first formulates the cache hit ratio maximization problem under the TrimCaching framework. Then, we map the formulated problem to a known NP-hard problem and show that no polynomial-time algorithm can solve this problem with a constant approximation guarantee.

A. Problem Formulation

The TrimCaching framework aims to maximize the cache hit ratio for users' model requests by addressing the model placement problem under the storage capacity of edge servers and user latency requirements. The problem formulation is given as follows.

$$\mathcal{P}1.1: \max_{\mathbf{X}} \; U(\mathbf{X}) \tag{6a}$$

$$\text{s.t.} \quad (1), \tag{6b}$$

$$x_{m,i} \in \{0,1\}, \; \forall m \in \mathcal{M}, \forall i \in \mathcal{I}, \tag{6c}$$

where (1) ensures that the total storage used at each edge server does not exceed its capacity.

Note that our formulation ignores user mobility because the problem is solved based on a "snapshot" of user locations. This is commonly adopted in model placement schemes [30]. In practice, our algorithm can make model placement decisions by solving the above problem, then re-initiate model placement when the performance degrades to a certain threshold. Our simulation results will show that our algorithm is resilient to user mobility over time, thus eliminating the need for frequent model replacement that would consume backbone bandwidth.

B. Problem Mapping

Solving $\mathcal{P}1.1$ is extremely challenging due to the product of integer decision variables arising from parameter block sharing. In this subsection, we show that the problem can be mapped to a known NP-hard problem and that no polynomial-time algorithm can achieve a constant approximation guarantee.

We begin by introducing the concept of submodularity. A set function $f: 2^{\mathbf{W}} \to \mathbb{R}$ is called submodular if $f(\mathbf{S}) + f(\mathbf{T}) \ge f(\mathbf{S} \cup \mathbf{T}) + f(\mathbf{S} \cap \mathbf{T})$ holds for all subsets $\mathbf{S}, \mathbf{T} \subseteq \mathbf{W}$, where W is a finite ground set [52]. An equivalent characterization is that f is submodular if and only if $f(\mathbf{S} \cup \{w\}) - f(\mathbf{S}) \ge f(\mathbf{T} \cup \{w\}) - f(\mathbf{T})$ for all $\mathbf{S} \subseteq \mathbf{T} \subseteq \mathbf{W}$ and $w \in \mathbf{W} \setminus \mathbf{T}$. Conversely, a function f is called supermodular if the reversed inequalities hold for every pair of subsets [53].

The problem mapping for $\mathcal{P}1.1$ is presented in the following proposition.

Proposition 1: $\mathcal{P}1.1$ can be mapped to a submodular maximization problem with M submodular constraints.

Proof: The proof is provided in Appendix A in our supplementary material. □

Next, we characterize the computational hardness of $\mathcal{P}1.1$.

Proposition 2: $\mathcal{P}1.1$ is an NP-hard problem. Moreover, no polynomial-time algorithm that solves $\mathcal{P}1.1$ with a constant approximation guarantee exists.

Proof: The proof is provided in Appendix B in our supplementary material. □

Although $\mathcal{P}1.1$ cannot be solved approximately in general, in the following sections, we will first introduce a special case of $\mathcal{P}1.1$, which captures a typical parameter sharing scenario in practice, for which a polynomial-time algorithm can be developed to obtain a solution with a constant approximation guarantee. Subsequently, for the general case, we will propose a greedy algorithm to solve $\mathcal{P}1.1$. Although this algorithm does not offer a constant approximation guarantee, the solution approach is still highly effective.

V. Special Case: A Small Fixed Number of Shared Parameter Blocks

This section first presents the assumption for the special case of $\mathcal{P}1.1$, supported by concrete real-world examples. Under this special case, we design an algorithm with polynomial-time complexity and a constant approximation guarantee for maximizing the cache hit ratio in the TrimCaching framework.

A. Assumption

Recall that $\mathcal{J}^{\text{sh}}$ denotes the set of shared parameter blocks. The following assumption formally describes the special case considered in this paper.

Assumption 1: There is a small fixed number of shared parameter blocks among models in $\mathcal{I}$; that is, there exists a constant C, satisfying $C \ll |\mathcal{I}|$ and $\left|\mathcal{J}^{\text{sh}}\right| \le C$.

Such special cases are often observed in practice, as numerous downstream AI models can be derived from a small number of pre-trained models. For example, transfer learning enables fine-tuning models pre-trained on large-scale datasets for a wide range of specific tasks [21], [54]. In practice, this is supported by frameworks such as TensorFlow and PyTorch, which provide a few pre-trained CNNs (e.g., ResNet models pre-trained on ImageNet) that can be adapted to various downstream tasks by freezing some layers and updating only task-specific layers [20], [55]. Similarly, in the LLM domain, PEFT methods adapt foundation models by updating only a tiny fraction of parameters, producing a large number of downstream models that share the same backbone models. For instance, Apple Intelligence freezes a base pre-trained model and fine-tunes many lightweight adapter layers for

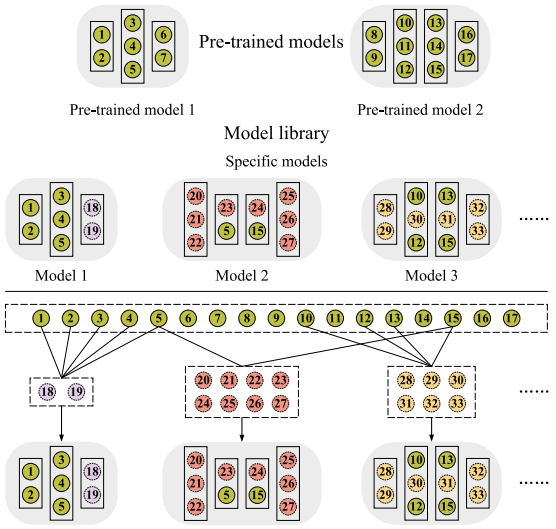


Fig. 4. An example of the special case with a small fixed number of shared parameter blocks. In the figure, regardless of the scale of the model library, the shared parameter blocks (green) come from two pre-trained models. Nodes in other colors represent specific parameter blocks.

Algorithm 1 TrimCaching Spec Algorithm

Input: $\mathcal{I}$, $\mathcal{K}$, and $\mathcal{M}$.
Output: $\hat{\mathbf{X}}$.
1 **Initialize:** $\hat{\mathbf{X}}_m = \mathbf{0}$ and $\mathbb{I}_2(m,k,i) = 1$.
2 **for** $m \in \mathcal{M}$ **do**
3 Solve $\mathcal{P}2.1_m$ with Algorithm 3 to obtain $\hat{\mathbf{X}}_m$ for edge server m.
4 **end**
5 $\hat{\mathbf{X}} = \bigcup_{m\in\mathcal{M}} \hat{\mathbf{X}}_m$.

diverse tasks, such as summarization, proofreading, and mail replies [56]. Moreover, Hugging Face hosts tens of thousands of LoRA modules, which are fine-tuned from a few pre-trained LLMs [24], [57]. Notably, even GPT models alone have hundreds of LoRA modules available. These examples demonstrate that, despite the vast number of models in the considered model library, there might only be a very small number of shared parameter blocks, where a block can be a layer or an entire pre-trained backbone (e.g., a pre-trained GPT model). An illustrative example of the considered special case is shown in Fig. 4, where all shared parameter blocks originate from 2 pre-trained models.

B. TrimCaching Spec Algorithm

Under Assumption 1 in the special case, the number of shared parameter blocks is independent of the model library size, and thus of the problem scale, making it feasible to traverse all the shared parameter blocks. Building on this, our key idea is to develop an approximation algorithm by traversing the combinations of all shared parameter blocks while judiciously selecting specific parameter blocks, resulting in a polynomial-time algorithm and $\frac{1-\epsilon}{2}$ approximation ratio.

We propose a successive greedy-based algorithm to solve $\mathcal{P}1.1$ under the special case. The proposed TrimCaching Spec algorithm is summarized in Algorithm 1. This algorithm operates by decomposing $\mathcal{P}1.1$ into M sub-problems, each corresponding to edge server m, and solving sub-problems sequentially in ascending order of server indices. Specifically, in Line 3 of Algorithm 1, the caching decision $\hat{\mathbf{X}}_m$ for edge server m is determined by solving the m-th sub-problem, which is formulated as follows.

$$\mathcal{P}2.1_m: \max_{\hat{\mathbf{X}}_m} \hat{U}_m\left(\hat{\mathbf{X}}_m\right) \tag{7a}$$

$$\text{s.t.} \quad \sum_{j\in\mathcal{J}} D'_j \left[1 - \prod_{i\in\mathcal{I}_j}\left(1-\hat{x}_{m,i}\right)\right] \le Q_m, \ \forall m \in \mathcal{M}, \tag{7b}$$

$$\hat{x}_{m,i} \in \{0,1\}, \ \forall i \in \mathcal{I}, \tag{7c}$$

where[2]

$$\hat{U}_m\left(\hat{\mathbf{X}}_m\right) = \frac{\sum_{k\in\mathcal{K}}\sum_{i\in\mathcal{I}} p_{k,i}\hat{x}_{m,i}\mathbb{I}_1(m,k,i)\,\mathbb{I}_2(m,k,i)}{\sum_{k\in\mathcal{K}}\sum_{i\in\mathcal{I}} p_{k,i}}. \tag{8}$$

Here, $\mathbb{I}_2(m,k,i)$ is given by

$$\mathbb{I}_2(m,k,i) = \prod_{m'=1}^{m-1}\left(1 - \hat{x}_{m',i}\mathbb{I}_{\{T_{m',k,i}\le \bar{T}_{k,i}\}}\right), \tag{9}$$

where $\mathbb{I}_2(m,k,i) = 1$ represents that the model request for model i of user k has not been satisfied by any of the first $m-1$ edge servers. For initialization, $\mathbb{I}_2(m,k,i)$ is set to 1 when $m = 1$.

The details of solving $\mathcal{P}2.1_m$ will be presented in the next subsection. With $\hat{\mathbf{X}}_m$, the solution to $\mathcal{P}1.1$ produced by Algorithm 1 is denoted by $\hat{\mathbf{X}} = \bigcup_{m\in\mathcal{M}} \hat{\mathbf{X}}_m$, satisfying the following proposition.

Proposition 3: The cache hit ratio of $\hat{\mathbf{X}}$ is equal to the sum of the cache hit ratios of $\hat{\mathbf{X}}_m$ for edge server m, i.e.,

$$U\left(\hat{\mathbf{X}}\right) = U\left(\bigcup_{m\in\mathcal{M}} \hat{\mathbf{X}}_m\right) = \sum_{m\in\mathcal{M}} \hat{U}_m\left(\hat{\mathbf{X}}_m\right). \tag{10}$$

Proof: The proof is shown in Appendix C in our supplementary material. □

Furthermore, based on Proposition 3, we have the following proposition.

Proposition 4: Assume that each sub-problem $\mathcal{P}2.1_m$ can be solved optimally in Algorithm 1. Under the considered special case, the TrimCaching Spec algorithm obtains a solution $\hat{\mathbf{X}}$ to $\mathcal{P}1.1$ which is lower bounded by $U\left(\hat{\mathbf{X}}\right) \ge \frac{1}{2}U\left(\mathbf{X}^*\right)$, where $\mathbf{X}^*$ is the optimal solution to $\mathcal{P}1.1$.

Proof: The proof is presented in Appendix D in our supplementary material. □

[2]For notation simplicity, $\hat{U}_m\left(\hat{\mathbf{X}}_m\right)$ is used as shorthand for $\hat{U}_m\left(\hat{\mathbf{X}}_m \middle| \bigcup_{m'=1}^{m-1} \hat{\mathbf{X}}_{m'}\right)$.

C. Rounding DP Approach and ϵ-Optimal Solutions for Sub-Problems

In this subsection, we propose a rounding DP-based algorithm to obtain an ϵ-optimal solution to $\mathcal{P}2.1_m$. To begin with, we address the submodularity of the constraint in $\mathcal{P}2.1_m$ by decoupling the caching decisions for shared and specific parameter blocks. Specifically, under Assumption 1, we determine the placement of shared parameter blocks via the exhaustive search. Once the placement decisions for shared parameter blocks are determined, we develop a rounding DP-based method to make the placement decisions for specific parameter blocks. For ease of presentation, let $\mathcal{A}$ denote the set of all possible combinations of shared parameter blocks in $\mathcal{J}^{\text{sh}}$, where each element $\mathcal{N} \in \mathcal{A}$ represents a specific combination of shared parameter blocks. For any $\mathcal{N}$, let $\mathcal{I}_{\mathcal{N}} = \{i \mid \mathcal{J}_i \subseteq \mathcal{N}\}$ be the set of models whose shared parameter blocks are fully contained in $\mathcal{N}$. Additionally, let $d_{\mathcal{N}}$ denote the total size of the parameter blocks in $\mathcal{N}$, and let $D(i)$ denote the total size of the specific parameter blocks of model i.[3]

1) Number of Cache Hits of Models in $\mathcal{I}_{\mathcal{N}}$ on an Edge Server: Given $\mathcal{N}$, the number of cache hits of model $i \in \mathcal{I}_{\mathcal{N}}$ on edge server m is given by

$$u(m,i) = \sum_{k \in \mathcal{K}} p_{k,i} \mathbb{I}_1(m,k,i)\, \mathbb{I}_2(m,k,i), \tag{11}$$

which is a fixed-point number due to the existence of $p_{k,i}$. We denote the granularity of $u(m,i)$ of models in $\mathcal{I}_{\mathcal{N}}$ on edge server m by $\delta_{m,\mathcal{N}}$, which reflects the precision of $u(m,i)$.[4] Therefore, the total number of cache hits of models in $\mathcal{I}_{\mathcal{N}}$ on edge server m has at most $W_{m,\mathcal{N}}+1$ possible values, where $W_{m,\mathcal{N}} = \frac{\sum_{i \in \mathcal{I}_{\mathcal{N}}} u(m,i)}{\delta_{m,\mathcal{N}}}$, and the $w_{m,\mathcal{N}}$-th value is $w_{m,\mathcal{N}}\delta_{m,\mathcal{N}}$, for $w_{m,\mathcal{N}} \in \{0,1,\ldots,W_{m,\mathcal{N}}\}$. Since the DP method traverses all possible values of the total number of cache hits, a smaller $\delta_{m,\mathcal{N}}$ leads to a larger $W_{m,\mathcal{N}}$, thereby increasing the DP complexity.

To facilitate the DP execution, we introduce a constant factor $\epsilon \in [0,1]$, and $u(m,i)$ is rounded to

$$\dot{u}(m,i) = \begin{cases} \left\lfloor \dfrac{u(m,i)}{\epsilon u_{m,\min}} \right\rfloor, & \epsilon > 0, \\ u(m,i), & \epsilon = 0, \end{cases} \tag{12}$$

where $u_{m,\min} = \min_{i \in \mathcal{I}} u(m,i)$. After rounding, the number of total cache hits of models in $\mathcal{I}_{\mathcal{N}}$ on edge server m has at most $\dot{W}_{m,\mathcal{N}}+1$ possible values, where $\dot{W}_{m,\mathcal{N}} = \frac{\sum_{i \in \mathcal{I}_{\mathcal{N}}} \dot{u}(m,i)}{\dot{\delta}_{m,\mathcal{N}}}$, $\dot{\delta}_{m,\mathcal{N}}$ is the granularity of $\dot{u}(m,i)$ for models in $\mathcal{I}_{\mathcal{N}}$ on edge server m, and the $\dot{w}_{m,\mathcal{N}}$-th value is $\dot{w}_{m,\mathcal{N}}\dot{\delta}_{m,\mathcal{N}}$, for $\dot{w}_{m,\mathcal{N}} \in \{0,1,\ldots,\dot{W}_{m,\mathcal{N}}\}$.

2) Maximum Cache Hit Ratio of Models in $\mathcal{I}_{\mathcal{N}}$ on an Edge Server: Let $\mathcal{T}(e_{\mathcal{N}}, \dot{w}_{\mathcal{N}})$ represent the minimum data size of specific parameter blocks that must be cached on edge server m, which achieves $\dot{w}_{\mathcal{N}}\dot{\delta}_{m,\mathcal{N}}$ cache hits (i.e., the $\dot{w}_{\mathcal{N}}$-th value of the number of cache hits) with the first $e_{\mathcal{N}}$ models in $\mathcal{I}_{\mathcal{N}}$. The state-transition equation for updating $\mathcal{T}(e_{\mathcal{N}}, \dot{w}_{\mathcal{N}})$ is given by

$$\mathcal{T}(e_{\mathcal{N}}, \dot{w}_{\mathcal{N}}) = \begin{cases} \min \begin{cases} \mathcal{T}(e_{\mathcal{N}}-1, \dot{w}_{\mathcal{N}}), \\ \mathcal{T}\left(e_{\mathcal{N}}-1, \dot{w}_{\mathcal{N}} - \dfrac{\dot{u}(m,e_{\mathcal{N}})}{\dot{\delta}_{m,\mathcal{N}}}\right) + D(e_{\mathcal{N}}), \end{cases} & \dot{w}_{\mathcal{N}} \geq \dfrac{\dot{u}(m,e_{\mathcal{N}})}{\dot{\delta}_{m,\mathcal{N}}}, \\ \mathcal{T}(e_{\mathcal{N}}-1, \dot{w}_{\mathcal{N}}), & \dot{w}_{\mathcal{N}} < \dfrac{\dot{u}(m,e_{\mathcal{N}})}{\dot{\delta}_{m,\mathcal{N}}}, \end{cases} \tag{13}$$

where $e_{\mathcal{N}} \in \{0,1,\ldots,|\mathcal{I}_{\mathcal{N}}|\}$, and the initial value of $\mathcal{T}(e_{\mathcal{N}}, \dot{w}_{\mathcal{N}})$ is

$$\mathcal{T}(e_{\mathcal{N}}, \dot{w}_{\mathcal{N}}) = \begin{cases} \infty, & \text{if } \dot{w}_{\mathcal{N}} \neq 0, \\ 0, & \text{if } \dot{w}_{\mathcal{N}} = 0 \text{ or } e_{\mathcal{N}} = 0. \end{cases} \tag{14}$$

The detailed process of updating $\mathcal{T}(e_{\mathcal{N}}, \dot{w}_{\mathcal{N}})$ is illustrated in Fig. 5 and summarized below. When calculating $\mathcal{T}(e_{\mathcal{N}}, \dot{w}_{\mathcal{N}})$, we consider the following two cases. (1) If $\dot{w}_{\mathcal{N}} \geq \frac{\dot{u}(m,e_{\mathcal{N}})}{\dot{\delta}_{m,\mathcal{N}}}$, i.e., the number of cache hits of model $e_{\mathcal{N}}$ is less than or equal to the $\dot{w}_{\mathcal{N}}$-th value of the number of cache hits, then $\mathcal{T}(e_{\mathcal{N}}, \dot{w}_{\mathcal{N}})$ is the minimum of $\mathcal{T}(e_{\mathcal{N}}-1, \dot{w}_{\mathcal{N}})$ and $\mathcal{T}\left(e_{\mathcal{N}}-1, \dot{w}_{\mathcal{N}} - \frac{\dot{u}(m,e_{\mathcal{N}})}{\dot{\delta}_{m,\mathcal{N}}}\right) + D(e_{\mathcal{N}})$.

- If $\mathcal{T}(e_{\mathcal{N}}-1, \dot{w}_{\mathcal{N}})$ is smaller, it indicates that a subset of the first $e_{\mathcal{N}}-1$ models can achieve the $\dot{w}_{\mathcal{N}}$-th value of the number of cache hits with a smaller data size. Therefore, the same data size is sufficient to achieve the $\dot{w}_{\mathcal{N}}$-th value of the number of cache hits using the first $e_{\mathcal{N}}$ models.
- If $\mathcal{T}\left(e_{\mathcal{N}}-1, \dot{w}_{\mathcal{N}} - \frac{\dot{u}(m,e_{\mathcal{N}})}{\dot{\delta}_{m,\mathcal{N}}}\right) + D(e_{\mathcal{N}})$ is smaller, it represents $\dot{w}_{\mathcal{N}}\dot{\delta}_{m,\mathcal{N}}$ is achieved by including model $e_{\mathcal{N}}$, which contributes $\dot{u}(m,e_{\mathcal{N}})$, and using a subset of the first $e_{\mathcal{N}}-1$ models to provide the remaining number of cache hits. Therefore, $D(e_{\mathcal{N}})$ is added to $\mathcal{T}\left(e_{\mathcal{N}}-1, \dot{w}_{\mathcal{N}} - \frac{\dot{u}(m,e_{\mathcal{N}})}{\dot{\delta}_{m,\mathcal{N}}}\right)$.

(2) If $\dot{w}_{\mathcal{N}} < \frac{\dot{u}(m,e_{\mathcal{N}})}{\dot{\delta}_{m,\mathcal{N}}}$, i.e., the number of cache hits of model $e_{\mathcal{N}}$ exceeds the $\dot{w}_{\mathcal{N}}$-th value of the number of cache hits, then the $\dot{w}_{\mathcal{N}}$-th value of the number of cache hits must be achieved only by a subset of the first $e_{\mathcal{N}}-1$ models without including model $e_{\mathcal{N}}$.

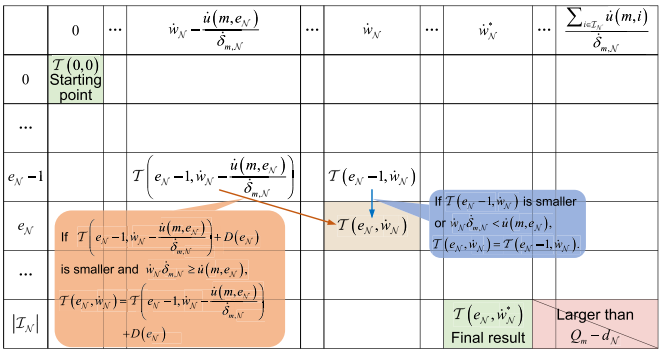


Fig. 5. The process of updating $\mathcal{T}(e_{\mathcal{N}}, \dot{w}_{\mathcal{N}})$ for edge server m, where the first row is the index of the number of cache hits and the first column is the model index in $\mathcal{I}_{\mathcal{N}}$.

[3]For example, in Fig. 4, $\{1,2,3,4,5,15\}$ and $\{10,12,13,15\}$ are two valid instances of $\mathcal{N}$. For $\mathcal{N} = \{1,2,3,4,5,15\}$, $\mathcal{I}_{\mathcal{N}}$ includes models 1 and 2. Moreover, $d_{\mathcal{N}}$ is the total size of parameter blocks {1, 2, 3, 4, 5, 15}, and $D(1)$ is the total size of parameter blocks {18, 19}.

[4]For example, given $\mathcal{I}_{\mathcal{N}} = \{1,2\}$, if $u(m,1) = 0.12$ and $u(m,2) = 0.14$, the precision of both values is two decimal places, and $\delta_{m,\mathcal{N}} = 0.01$.

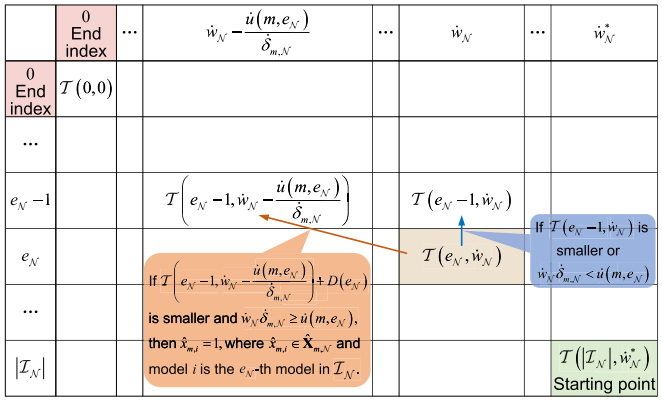


Fig. 6. The process of determining $\hat{\mathbf{X}}_{m,\mathcal{N}}$, where the first row and the first column are the same as those in Fig. 5.

After obtaining $\mathcal{T}(e_{\mathcal{N}},\dot{w}_{\mathcal{N}})$ for all $e_{\mathcal{N}}$ and $\dot{w}_{\mathcal{N}}$, the index corresponding to the maximum number of cache hits of models in $\mathcal{I}_{\mathcal{N}}$ on edge server m is given by

$$\dot{w}^*_{\mathcal{N}}=\arg\max_{\dot{w}_{\mathcal{N}}}\{\dot{w}_{\mathcal{N}}\mid\mathcal{T}(|\mathcal{I}_{\mathcal{N}}|,\dot{w}_{\mathcal{N}})\leq Q_m-d_{\mathcal{N}}\}. \tag{15}$$

With $\dot{w}^*_{\mathcal{N}}$, the maximum cache hit ratio of models in $\mathcal{I}_{\mathcal{N}}$ on edge server m is expressed as

$$\hat{U}_m\left(\hat{\mathbf{X}}_{m,\mathcal{N}}\right)=\frac{\sum\limits_{i\in\hat{\mathcal{I}}_{m,\mathcal{N}}}u(m,i)}{\sum\limits_{k\in\mathcal{K}}\sum\limits_{i\in\mathcal{I}}p_{k,i}}, \tag{16}$$

where $\hat{\mathbf{X}}_{m,\mathcal{N}}$ is the model caching decision corresponding to $\dot{w}^*_{\mathcal{N}}$, and $\hat{\mathcal{I}}_{m,\mathcal{N}}=\left\{i\mid\hat{x}_{m,i}=1,\hat{x}_{m,i}\in\hat{\mathbf{X}}_{m,\mathcal{N}}\right\}$. The details for determining $\hat{\mathbf{X}}_{m,\mathcal{N}}$ will be presented in the following paragraphs. Moreover, note that the rounding DP-based algorithm described above operates based on $\dot{u}(m,i)$, whereas we use $u(m,i)$ to calculate $\hat{U}_m\left(\hat{\mathbf{X}}_{m,\mathcal{N}}\right)$ in (16) at last.

3) Optimal Model Caching of Models in $\mathcal{I}_{\mathcal{N}}$ on an Edge Server: The determination of $\hat{\mathbf{X}}_{m,\mathcal{N}}$ is summarized in Algorithm 2 and illustrated in Fig. 6. The algorithm determines $\hat{\mathbf{X}}_{m,\mathcal{N}}$ recursively by comparing $\dot{u}(m,e_{\mathcal{N}})$ with the remaining number of cache hits $\dot{w}_{\mathcal{N}}\dot{\delta}_{m,\mathcal{N}}$, starting from model $|\mathcal{I}_{\mathcal{N}}|$ and number of cache hits $\dot{w}^*_{\mathcal{N}}\dot{\delta}_{m,\mathcal{N}}$. In Line 5 of Algorithm 2, the $e_{\mathcal{N}}$-th model in $\mathcal{I}_{\mathcal{N}}$ is placed on edge server m if the following two conditions are satisfied.

- $\dot{w}_{\mathcal{N}}\dot{\delta}_{m,\mathcal{N}}\geq\dot{u}(m,e_{\mathcal{N}})$, implying that the number of cache hits of the $e_{\mathcal{N}}$-th model is no greater than the remaining number of cache hits to be satisfied.
- $\mathcal{T}\left(e_{\mathcal{N}}-1,\dot{w}_{\mathcal{N}}-\frac{\dot{u}(m,e_{\mathcal{N}})}{\dot{\delta}_{m,\mathcal{N}}}\right)+D(e_{\mathcal{N}})<\mathcal{T}(e_{\mathcal{N}}-1,\dot{w}_{\mathcal{N}})$, representing that the number of cache hits $\dot{w}_{\mathcal{N}}\dot{\delta}_{m,\mathcal{N}}$ is achieved by model $e_{\mathcal{N}}$, contributing $\dot{u}(m,e_{\mathcal{N}})$, and a subset of the first $e_{\mathcal{N}}-1$ models, contributing $\dot{w}_{\mathcal{N}}\dot{\delta}_{m,\mathcal{N}}-\dot{u}(m,e_{\mathcal{N}})$. Besides, the corresponding total data size, $\mathcal{T}\left(e_{\mathcal{N}}-1,\dot{w}_{\mathcal{N}}-\frac{\dot{u}(m,e_{\mathcal{N}})}{\dot{\delta}_{m,\mathcal{N}}}\right)+D(e_{\mathcal{N}})$, is less than $\mathcal{T}(e_{\mathcal{N}}-1,\dot{w}_{\mathcal{N}})$.

After processing model $e_{\mathcal{N}}$, the algorithm repeats the same steps above to process the $e_{\mathcal{N}}-1$-th model until all models in $\mathcal{I}_{\mathcal{N}}$ have been checked. Note that there may be more than one feasible model caching decision for $\mathcal{I}_{\mathcal{N}}$ that can achieve the same number of cache hits $\dot{w}^*_{\mathcal{N}}\dot{\delta}_{m,\mathcal{N}}$ with the data size $\mathcal{T}\left(|\mathcal{I}_{\mathcal{N}}|,w^*_{\mathcal{N}}\right)$. Although Algorithm 2 produces one feasible $\hat{\mathbf{X}}_{m,\mathcal{N}}$, this does not affect the optimality.

Algorithm 2 Model Caching Decision Algorithm

Input: m, $\mathcal{I}_{\mathcal{N}}$, $\dot{\delta}_{m,\mathcal{N}}$, $\dot{u}(m,e_{\mathcal{N}})$, and $\mathcal{T}(e_{\mathcal{N}},\dot{w}_{\mathcal{N}})$.
Output: $\hat{\mathbf{X}}_{m,\mathcal{N}}$.
1 **Initialize:** $e_{\mathcal{N}}=|\mathcal{I}_{\mathcal{N}}|$, $\dot{w}_{\mathcal{N}}=\dot{w}^*_{\mathcal{N}}$, and $\hat{\mathbf{X}}_{m,\mathcal{N}}=\mathbf{0}$
2 **while** $e_{\mathcal{N}}\neq 0$ *and* $\dot{w}_{\mathcal{N}}\neq 0$ **do**
3 **if** $\dot{w}_{\mathcal{N}}\dot{\delta}_{m,\mathcal{N}}\geq\dot{u}(m,e_{\mathcal{N}})$ **then**
4 **if** $\mathcal{T}\left(e_{\mathcal{N}}-1,\dot{w}_{\mathcal{N}}-\frac{\dot{u}(m,e_{\mathcal{N}})}{\dot{\delta}_{m,\mathcal{N}}}\right)+D(e_{\mathcal{N}})<\mathcal{T}(e_{\mathcal{N}}-1,\dot{w}_{\mathcal{N}})$ **then**
5 $\hat{x}_{m,i}=1$, where $\hat{x}_{m,i}\in\hat{\mathbf{X}}_{m,\mathcal{N}}$ and model i is the $e_{\mathcal{N}}$-th model in $\mathcal{I}_{\mathcal{N}}$.
6 $\dot{w}_{\mathcal{N}}=\dot{w}_{\mathcal{N}}-\frac{\dot{u}(m,e_{\mathcal{N}})}{\dot{\delta}_{m,\mathcal{N}}}$.
7 **end**
8 **end**
9 $e_{\mathcal{N}}=e_{\mathcal{N}}-1$.
10 **end**

4) ϵ-Optimal Solution to $\mathcal{P}2.1_m$ and Rounding DP-Based Algorithm Outline: After traversing all feasible $\mathcal{N}$ in $\mathcal{A}$, the maximum number of cache hits of models on edge server m corresponds to the index $\dot{w}^*_{\mathcal{N}^*}$, where $\mathcal{N}^*=\arg\max_{\mathcal{N}}\{\dot{w}^*_{\mathcal{N}}\}$. Letting $\hat{\mathbf{X}}_{m,\mathcal{N}^*}$ be the solution $\hat{\mathbf{X}}_m$ to $\mathcal{P}2.1_m$, the maximum cache hit ratio of edge server m is given by

$$\hat{U}_m\left(\hat{\mathbf{X}}_m\right)=\hat{U}_m\left(\hat{\mathbf{X}}_{m,\mathcal{N}^*}\right)=\frac{\sum\limits_{i\in\hat{\mathcal{I}}_m}u(m,i)}{\sum\limits_{k\in\mathcal{K}}\sum\limits_{i\in\mathcal{I}}p_{k,i}}, \tag{17}$$

where $\hat{\mathcal{I}}_m=\hat{\mathcal{I}}_{m,\mathcal{N}^*}$.

The rounding DP-based algorithm is outlined in Algorithm 3.

Furthermore, we establish the following proposition for Algorithm 3.

Proposition 5: The cache hit ratio produced by Algorithm 3 satisfies $\hat{U}_m\left(\hat{\mathbf{X}}_m\right)\geq(1-\epsilon)\,\hat{U}_m\left(\hat{\mathbf{X}}^*_m\right)$, where $\hat{\mathbf{X}}^*_m$ is the optimal solution to $\mathcal{P}2.1_m$.

Proof: The proof is shown in Appendix E in our supplementary material. □

D. Analysis of the TrimCaching Spec Algorithm

1) Time Complexity Analysis: With Proposition 5, we first establish the following theorem on the time complexity of the proposed TrimCaching Spec Algorithm under the special case.

Theorem 1: Under Assumption 1 for the special case of $\mathcal{P}1.1$, where the number of shared parameter blocks is small and fixed, the TrimCaching Spec algorithm has a polynomial-time computational complexity $O(MI)$.

Proof: The proof is presented in Appendix F in our supplementary material. □

In the special case, the complexity of the TrimCaching Spec algorithm can be further reduced if models are fine-tuned with the bottom-layer freezing technique, which involves freezing

Algorithm 3 Rounding DP-Based Algorithm

Input: m, ϵ, $\mathcal{K}$, $\mathcal{I}$.
Output: $\hat{U}_m\left(\hat{\mathbf{X}}_m\right)$, $\hat{\mathbf{X}}_m$.
Initialize: $\hat{U}_m\left(\hat{\mathbf{X}}_m\right)=0$, $\hat{\mathbf{X}}_m=\mathbf{0}$, $\mathcal{N}^*=\emptyset$, and $\hat{U}_m\left(\hat{\mathbf{X}}_{m,\mathcal{N}^*}\right)=0$.
Calculate $u(m,i)$, and $\dot{u}(m,i)$ with (11) and (12), respectively.
for $\mathcal{N}\in\mathcal{A}$ **do**
  Calculate $d_{\mathcal{N}}$.
  if $d_{\mathcal{N}}>Q_m$ **then**
    Continue.
  end
  Calculate $\dot{\delta}_{m,\mathcal{N}}$ of $\dot{u}(m,i)$ for models in $\mathcal{I}_{\mathcal{N}}$.
  Calculate $D(e_{\mathcal{N}})$ for all $e_{\mathcal{N}}$ in $\mathcal{I}_{\mathcal{N}}$.
  Initialize $\mathcal{T}(e_{\mathcal{N}},\dot{w}_{\mathcal{N}})$ using (14).
  for $e_{\mathcal{N}}\in\{1,\ldots,|\mathcal{I}_{\mathcal{N}}|\}$ **do**
    for $\dot{w}_{\mathcal{N}}\in\left\{1,\ldots,\dot{W}_{m,\mathcal{N}}\right\}$ **do**
      Calculate $\mathcal{T}(e_{\mathcal{N}},\dot{w}_{\mathcal{N}})$ with (13).
    end
  end
  Calculate $\dot{w}^*_{\mathcal{N}}$ with (15).
  Calculate $\hat{\mathbf{X}}_{m,\mathcal{N}}$ with Algorithm 2.
  Calculate $\hat{U}_m\left(\hat{\mathbf{X}}_{m,\mathcal{N}}\right)$ with (16).
  if $\hat{U}_m\left(\hat{\mathbf{X}}_{m,\mathcal{N}}\right)>\hat{U}_m\left(\hat{\mathbf{X}}_{m,\mathcal{N}^*}\right)$ **then**
    $\mathcal{N}^*=\mathcal{N}$.
  end
end
$\hat{\mathbf{X}}_m=\hat{\mathbf{X}}_{m,\mathcal{N}^*}$.

a number of the bottom layers of a pre-trained model and only updating the top layers. This approach is commonly used in transfer learning. In this case, we have the following corollary.

Corollary 1: When models in the model library are fine-tuned from a single pre-trained model using the bottom-layer freezing technique, where each model shares a sequence of consecutive bottom layers from the pre-trained model, the time complexity of the TrimCaching Spec algorithm can be significantly reduced from $O\left(2^{|\mathcal{J}^{\text{sh}}|}MI\right)$ to $O((\kappa+1)MI)$, where κ denotes the maximum number of bottom layers shared between any model in the library and the pre-trained model.

Proof: The proof is provided in Appendix G in our supplementary material. □

2) Approximation Guarantee Analysis: Next, we analyze the approximation guarantee of the proposed TrimCaching Spec Algorithm. The details are summarized in the following theorem.

Theorem 2: The TrimCaching Spec algorithm obtains a solution $\hat{\mathbf{X}}$ to $\mathcal{P}1.1$ satisfying $U\left(\hat{\mathbf{X}}\right)\geq\frac{1-\epsilon}{2}U\left(\mathbf{X}^*\right)$.

Proof: The proof is provided in Appendix H in our supplementary material. □

It is noted that, when $M=1$, $\mathcal{P}1.1$ is equivalent to $\mathcal{P}2.1_m$. In this case, the TrimCaching Spec algorithm can directly use Algorithm 3 to solve the problem, leading to the following corollary.

Algorithm 4 TrimCaching Gen Algorithm

Input: $\mathcal{K}$, $\mathcal{I}$, and $\mathcal{M}$.
Output: $\mathbf{X}$ and $U(\mathbf{X})$.
1 **Initialize:** $l=0$, $\mathbf{X}^l=\mathbf{0}$, and $U\left(\mathbf{X}^l\right)=0$.
2 **while** *There exists* $\{m,i\}$ *such that* $g_m\left(\rho\left(\mathbf{X}^l_m,m,i\right)\right)\leq Q_m$ *and* $\Delta U\left(\mathbf{X}^l,m,i\right)\neq 0$. **do**
3 $l=l+1$.
4 Calculate $\Delta U\left(\mathbf{X}^{l-1},m,i\right)$ with (18) for $m\in\mathcal{M}$ and $i\in\mathcal{I}$.
5 $\{m^*,i^*\}=\arg\max_{m,i}\{\Delta U\left(\mathbf{X}^{l-1},m,i\right)\mid g_m\left(\rho\left(\mathbf{X}^{l-1}_m,m,i\right)\right)\leq Q_m\}$.
6 $\mathbf{X}^l=\rho\left(\mathbf{X}^{l-1},m^*,i^*\right)$.
7 **end**
8 $\mathbf{X}=\mathbf{X}^l$.

Corollary 2: In the single-edge scenario (M=1), the solution to $\mathcal{P}1.1$ produced by the TrimCaching Spec algorithm satisfies $U\left(\hat{\mathbf{X}}\right)\geq(1-\epsilon)U\left(\mathbf{X}^*\right)$.

Proof: The proof directly follows from Proposition 5. □

VI. The General Case: Arbitrary Parameter Sharing

This section examines the general case of $\mathcal{P}1.1$, where models can arbitrarily share parameter blocks, in contrast to the special case with a small fixed number of shared parameter blocks. Formally speaking, the number of shared parameter blocks among models may increase with the scale of the model library, such that searching for all combinations of shared parameter blocks, as required by the TrimCaching Spec algorithm, results in exponential time complexity in terms of the problem scale, which should be avoided in the general case.

To solve $\mathcal{P}1.1$ in the general case, we propose a greedy-based algorithm, TrimCaching Gen, outlined in Algorithm 4. The detailed procedures of Algorithm 4 are as follows. First, in the l-th step, based on the placement decision $\mathbf{X}^{l-1}$ determined in the $l-1$-th step, Algorithm 4 computes the marginal increase in the cache hit ratio of placing model i on edge server m in Line 4, which is expressed as

$$\Delta U\left(\mathbf{X}^{l-1},m,i\right)=U\left(\rho\left(\mathbf{X}^{l-1},m,i\right)\right)-U\left(\mathbf{X}^{l-1}\right), \tag{18}$$

where

$$\rho\left(\mathbf{X}^{l-1},m,i\right)=\begin{cases}\mathbf{X}^{l-1}\setminus\{x_{m,i}=0\}\cup\{x_{m,i}=1\}, & \text{if } x_{m,i}=0,\\ \mathbf{X}^{l-1}, & \text{if } x_{m,i}=1,\end{cases} \tag{19}$$

represents $x_{m,i}$ in $\mathbf{X}^{l-1}$ is updated to 1 if it was 0, and remains unchanged otherwise. Second, in Line 5, the algorithm identifies $\{m^*,i^*\}$ which yields the maximum marginal increase in the cache hit ratio, while ensuring that $g_{m^*}\left(\rho\left(\mathbf{X}^{l-1}_{m^*},m^*,i^*\right)\right)$ does not exceed the edge server capacity, where $\mathbf{X}^{l-1}_m$ is the model placement decision of edge server m in the $l-1$-th step based on $\mathbf{X}^{l-1}$, with $\mathbf{X}^{l-1}=\bigcup_{m\in\mathcal{M}}\mathbf{X}^{l-1}_m$. Third, in Line 6, $\mathbf{X}^{l-1}$ is updated to $\mathbf{X}^l=\rho\left(\mathbf{X}^{l-1},m^*,i^*\right)$. Finally, the algorithm repeats the above steps until no feasible pair $\{m,i\}$ can be found that

improves the cache hit ratio without violating any capacity constraint.

Next, we establish the following theorems for the TrimCaching Gen algorithm.

Theorem 3: The proposed algorithm solves $\mathcal{P}1.1$ with the time complexity of $O\left(M^2I^2\right)$.

Proof: The proof is similar to that of Theorem 1, which is omitted here. □

Theorem 4: The TrimCaching Gen algorithm obtains a solution X to $\mathcal{P}1.1$ satisfying $U(\mathbf{X}) \geq \frac{1}{\Gamma}U(\mathbf{X}^*)$, where $\Gamma = \max\{|\mathbf{X}| \mid g_m(\mathbf{X}_m) \leq Q_m, \forall m \in \mathcal{M}\}$, and $\mathbf{X}_m$ is the model placement decision of edge server m in X with $\mathbf{X} = \bigcup_{m\in\mathcal{M}} \mathbf{X}_m$.

Proof: The proof is omitted here. A similar proof can be found in [58] and [59]. □

Remark 1: Since the lower bound decreases with the increasing number of AI models and edge servers, there is no constant approximation guarantee for Algorithm 4. This coincides with Proposition 2, which states that no polynomial-time approximation algorithm exists for the general case.

VII. Numerical Results

In this section, we present simulation results for the proposed TrimCaching framework.

A. Simulation Setup

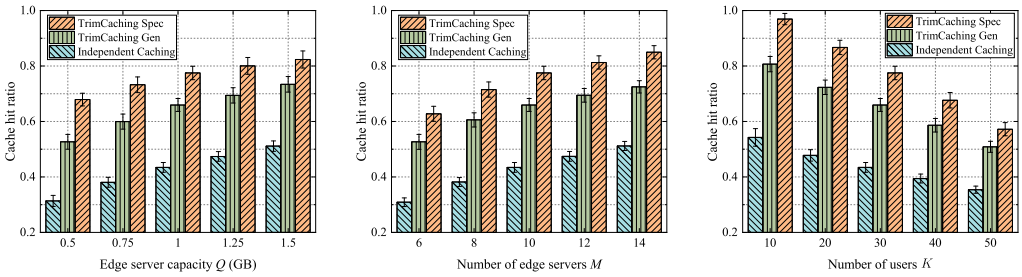
In the simulation, K users and M edge servers are uniformly distributed in a square area of 1 km^2. The latency requirements of users for downloading models and performing on-device inference are uniformly distributed over the interval $[0.5, 1]$ s [10]. The number of users is set to $K = \{10, 20, 30, 40, 50\}$. Each edge server has a coverage radius of 275 m, and the associated user set of edge server m is denoted as $\mathcal{K}_m$. The expected bandwidth and transmit power allocated by edge server m to user k in $\mathcal{K}_m$ are $\bar{B}_{m,k} = \frac{B}{p_A|\mathcal{K}_m|}$ and $\bar{P}_{m,k} = \frac{P}{p_A|\mathcal{K}_m|}$, respectively. Here, $B = 400$ MHz and $P = 43$ dBm [60] represent the total bandwidth and transmit power of an edge server, respectively. The user active probability is set to $p_A = 0.5$. The communication data rate between edge servers is set to $C_{m,m'} = 10$ Gbps [61], [62]. Besides, we set γ_0 and α_0 in (2) to 1 and 4 [46], [63], respectively. The number of edge servers is set to $M = \{6, 8, 10, 12, 14\}$. Besides, the storage capacity of each edge server is denoted by $Q_m = Q$, which is uniformly set across all servers and varies from 0.5 GB to 5 GB. It is noted that the storage capacity of edge servers can be much larger than 5 GB in reality. However, the model library can also be significantly larger than our constructed parameter-sharing model library, which will be introduced later. Due to our limited computing resources for model fine-tuning, we proportionally reduce the storage capacity of edge servers and the size of the model library, which will not impact the phenomenon observed in the experiments.

We construct two parameter-sharing model libraries, each derived from a different model family. The first model library is based on the ResNet family, including ResNet-18, ResNet-34, and ResNet-50 [25]. The second model library is based on the GPT family, including GPT-2 small, medium, and large. The parameter sharing among AI models in the special and general cases is constructed as follows.

TABLE II
Superclass Selection in Stage 1 and Stage 2

Stage 1	Stage 2
fruit and vegetables	flowers, trees
medium-sized mammals	large carnivores, large omnivores and herbivores, people, reptiles, small mammals
vehicles 2	large man-made outdoor things, vehicles 1

- **Special case**: In this case, we design parameter sharing in both the ResNet- and the GPT-2-based model libraries. In the ResNet-based model library, we fine-tune the three pre-trained ResNet models on CIFAR100 [26]. For each model structure, we obtain 100 downstream models, each corresponding to one class in CIFAR100. We adopt the bottom-layer freezing techniques to fine-tune these models. The number of frozen bottom layers (each corresponding to a shared parameter block in our system model) for ResNet-18, ResNet-34, and ResNet-50 falls within the ranges of $[29, 40]$, $[49, 72]$, and $[87, 106]$, respectively.
  For the GPT-2-based model library, we fine-tune the three pre-trained GPT-2 models on datasets, including E2E [27], WebNLG [28], and DART [29], to create 100 downstream models for each model structure. Moreover, we adopt LoRA to fine-tune these models, where downstream models fine-tuned from the same pre-trained model share its parameters. The sizes of newly introduced trainable parameters by LoRA for the GPT-2 small, medium, and large are: {0, 188,348, 335,804, 630,716, 1,220,540, 2,400,188, 4,759,484} Bytes, {0, 470,487, 863,703, 1,650,135, 3,222,999, 6,368,727, 12,660,247} Bytes, and {0, 852,175, 1,589,455, 3,064,015, 6,013,135, 11,911,375, 23,708,047} Bytes, respectively. Additionally, the GPT-2-based model library also includes models fine-tuned via the full-parameter fine-tuning approach.
- **General case**: In this case, we design parameter sharing in the ResNet-based model library. To support arbitrary parameter sharing among models in the library (shared parameter blocks do not simply come from a small set of pre-trained models, as in the special case, and the number of shared blocks increases as the library grows), we adopt a two-stage fine-tuning strategy. In the first stage, we select multiple superclasses from CIFAR100. For each selected superclass, all parameters of the pre-trained ResNet-18, ResNet-34, and ResNet-50 are fine-tuned using data from all its classes, resulting in one distinct model per structure for each superclass. In the second stage, each first-stage fine-tuned model is further fine-tuned on datasets from classes within a new superclass that is semantically similar to the one used in the first stage. The pairing of superclasses between stage 1 and stage 2 is provided in Table II. Moreover, the second-stage fine-tuning is conducted using bottom-layer freezing, without restricting the number of frozen layers,



(a) Cache hit ratio vs. Q, where $M = 10$ and $K = 30$.

(b) Cache hit ratio vs. M, where $Q = 1$ GB and $K = 30$.

(c) Cache hit ratio vs. K, where $Q = 1$ GB and $M = 10$.

Fig. 7. Cache hit ratio in the special case, where a small fixed number of shared parameter blocks is considered, using the ResNet-based model library. The error bar denotes the standard deviation, which is the same for the subsequent figures.

where each first-stage model is fine-tuned separately on the data of each class, resulting in a total of 45 models.

Moreover, the on-device inference latency for the ResNet-based and the GPT-2-based model libraries ranges from 1 ms to 5 ms and from 40 ms to 200 ms, respectively [10]. The request probabilities of each user for the models in each model library follow a Zipf distribution [64].

For comparison, three algorithms are considered:

- **TrimCaching Spec**: Algorithm 1, developed for the special case.
- **TrimCaching Gen**: Algorithm 4, developed for the general case but also applicable to the special case.
- **Independent Caching:** AI models are cached independently without considering parameter sharing, referring to traditional content placement schemes [48].

Moreover, ϵ is set to 0.1 in Algorithm 3 by default.

All simulation results are averaged from 100 network topologies. For each topology, we further evaluate the cache hit ratio using 10^3 Rayleigh fading channel realizations. It is noted that the caching decisions are made based on average channel gains, while the performance is examined under Rayleigh fading.

B. Performance in the Special Case

We first compare the algorithms in the special case. Since the TrimCaching Gen algorithm can also be applied to the special case, it is used as a benchmark. Fig. 7 presents the cache hit ratio of these algorithms using the ResNet-based model library. As shown in Fig. 7(a), increasing the storage capacity Q improves the cache hit ratio of all algorithms, as more models can be cached on edge servers. Unsurprisingly, both the TrimCaching Spec and TrimCaching Gen algorithms outperform the Independent Caching framework due to the storage efficiency of parameter-sharing caching. Moreover, the TrimCaching Spec algorithm consistently outperforms the TrimCaching Gen algorithm, as it offers a constant approximation guarantee, whereas the TrimCaching Gen algorithm does not. On average, the TrimCaching Spec algorithm achieves 11.93% and 33.93% higher cache hit ratios than the TrimCaching Gen algorithm and the Independent Caching framework, respectively. Moreover, a similar trend can be observed in Fig. 7(b), which varies the number of edge servers.

Fig. 7(c) evaluates the performance of the algorithms as the number of users increases. As expected, the cache hit ratio decreases as the number of users grows due to reduced transmission data rates. However, the proposed algorithms maintain substantial performance advantages. When $K = 50$, the TrimCaching Spec and TrimCaching Gen algorithms improve the cache hit ratio by about 21.81% and 15.47%, respectively, over the Independent Caching framework. Besides, the TrimCaching Spec algorithm achieves an average performance gain of 11.50% compared with the TrimCaching Gen algorithm, demonstrating its superior performance in the special case, while the TrimCaching Gen algorithm remains competitive.

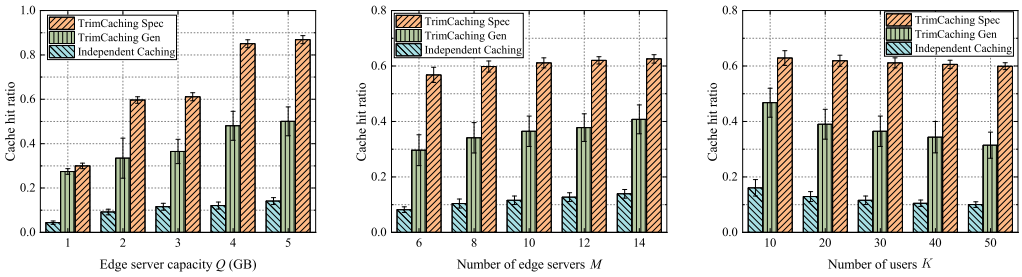
Fig. 8 demonstrates the cache hit ratio of the algorithms using the GPT-2-based model library, exhibiting trends similar to those shown in Fig. 7. The TrimCaching Spec algorithm demonstrates a notable improvement in average cache hit ratios compared with the TrimCaching Gen algorithm and the Independent Caching framework. As illustrated in Fig. 8, the TrimCaching Spec algorithm achieves higher cache hit ratios than the TrimCaching Gen algorithm and the Independent Caching framework by 25.44% and 54.26%, 24.71% and 49.11%, 23.68% and 49.06%, respectively, under varying storage capacities, numbers of edge servers, and numbers of users.

C. Performance in the General Case

As shown in Fig. 9, this subsection compares the cache hit ratio of the TrimCaching Gen algorithm and the Independent Caching framework using the ResNet-based model library in the general case. A trend similar to the special case can be observed. As the storage capacity or the number of edge servers increases, the cache hit ratio improves since more models can be cached. Moreover, increasing the number of users leads to a decline in the cache hit ratio due to the limited spectrum bandwidth. More importantly, the TrimCaching Gen algorithm significantly outperforms the Independent Caching framework in different scenarios.

D. Running Time Comparison

In this subsection, we compare the running time of the TrimCaching Spec algorithm, the TrimCaching Gen algorithm, and the exhaustive search using the ResNet-based model

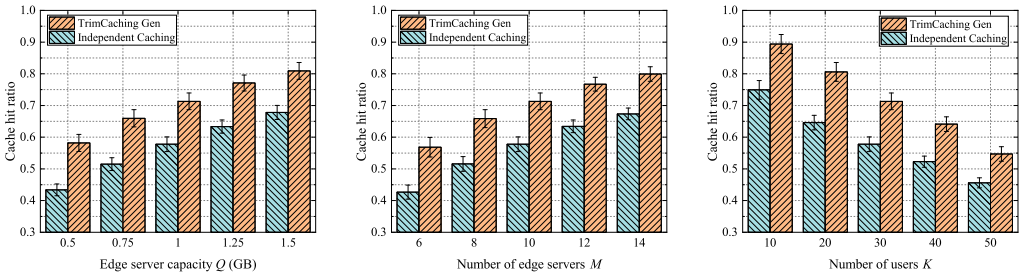


(a) Cache hit ratio vs. Q, where $M = 10$ and $K = 30$.

(b) Cache hit ratio vs. M, where $Q = 3$ GB and $K = 30$.

(c) Cache hit ratio vs. K, where $Q = 3$ GB and $M = 10$.

Fig. 8. Cache hit ratio in the special case using the GPT-2-based model library.

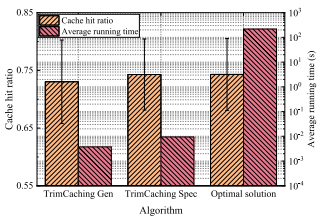


(a) Cache hit ratio vs. Q, where $M = 10$ and $K = 30$.

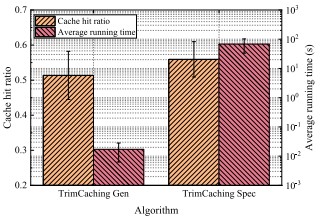
(b) Cache hit ratio vs. M, where $Q = 1$ GB and $K = 30$.

(c) Cache hit ratio vs. K, where $Q = 1$ GB and $M = 10$.

Fig. 9. Cache hit ratio in the general case using the ResNet-based model library.



(a) Results in the special case, where $Q = 0.1$ GB and each user requests 9 models.



(b) Results in the general case, where $Q = 0.2$ GB and each user requests 27 models.

Fig. 10. Cache hit ratio and average running time of different algorithms.

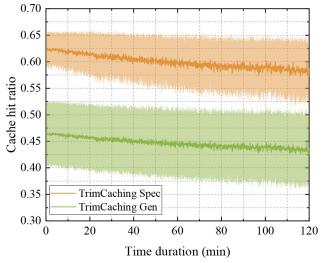


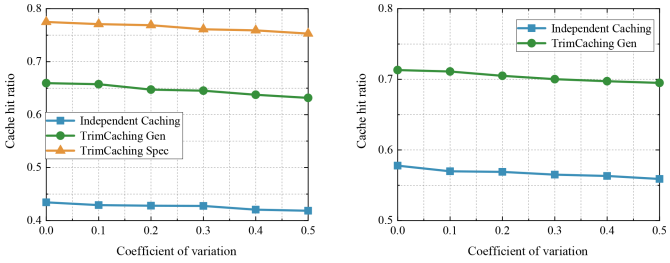
Fig. 11. Impact of user mobility on the cache hit ratio over time.

library. Here, the exhaustive search is employed to obtain the optimal solution, and incurs exponential complexity, i.e., 2^{MKI}. To run the exhaustive search efficiently, the length and width of the area are reduced to 400 m, and M and K are set to 2 and 6, respectively. Moreover, ϵ of Algorithm 3 is set to 0 in this subsection.

Fig. 10(a) compares the performance of our algorithms with the exhaustive search in the special case. On the one hand, the TrimCaching Spec algorithm achieves the same cache hit ratio as the optimal solution, while the cache hit ratio of the TrimCaching Gen algorithm is only 1.3% lower than the optimal solution. On the other hand, the TrimCaching Spec algorithm and the TrimCaching Gen algorithm are, on average, about 22,900 and 58,000 times faster than the exhaustive search, respectively, demonstrating the effectiveness and efficiency of our algorithms in the special case. Moreover, Fig. 10(b) compares the performance of the TrimCaching Gen algorithm with the TrimCaching Spec algorithm in the general case. The average running time of the TrimCaching Gen algorithm is about 3,900 times faster than the TrimCaching Spec algorithm. This underscores the necessity of designing the TrimCaching Gen algorithm for the general case, since the TrimCaching Spec algorithm exhibits exponential complexity in the general case.

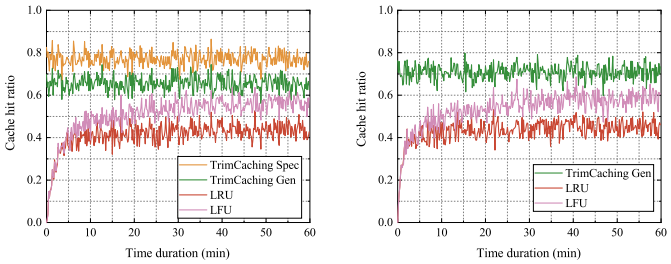
E. Robustness to User Mobility

In Fig. 11, we demonstrate the robustness of our algorithms over time by considering user mobility. In this subsection, we set M, K, and Q to 10, 10, and 3 GB, respectively, using the GPT-2-based model library. Moreover, three mobility patterns are considered, representing pedestrians, bikes, and vehicles. The initial speeds of users are randomly drawn from $[0.5, 1.8]$ m/s, $[2, 8]$ m/s, and $[5.5, 20]$ m/s, respectively. In each time slot, the acceleration is selected from $[-0.3, 0.3]$ m/s^2, $[-1, 1]$ m/s^2, and $[-3, 3]$ m/s^2, respectively. The initial movement orientations are uniformly distributed in $[0, \pi]$ rad. The angular velocities



(a) Results in the special case. (b) Results in the general case.

Fig. 12. Cache hit ratio versus the coefficient of variation of estimated request probabilities using the ResNet-based model library, where $Q = 1$ GB, $K = 30$, and $M = 10$.



(a) Comparisons in the special case. (b) Comparisons in the general case.

Fig. 13. Cache hit ratio comparisons with online placement strategies using the ResNet-based model library, where Q, K, and M follow those in Fig. 12.

range in $\left[-\frac{\pi}{4}, \frac{\pi}{4}\right]$ rad/s, $\left[-\frac{\pi}{3}, \frac{\pi}{3}\right]$ rad/s, and $\left[-\frac{\pi}{2}, \frac{\pi}{2}\right]$ rad/s. Users are assumed to change their speeds/orientations at the beginning of each time slot, and the time slot duration is set to 5 s.

Fig. 11 illustrates the robustness of the TrimCaching Spec and TrimCaching Gen algorithms in the special case using the GPT-2-based model library. While the cache hit ratio degrades over time due to user mobility, the performance degradation of the TrimCaching Spec and TrimCaching Gen algorithms is only about 3.91% and 2.83%, respectively, over 2 hours. This demonstrates the effectiveness of the TrimCaching framework in the long run, even though user mobility has not been explicitly considered in our problem formulation. This also implies that model replacement does not need to be re-conducted frequently, thereby saving backbone bandwidth resources.

F. Impact of Request Probability Uncertainty on Cache Hit Ratio

This subsection investigates the effect of the uncertainty of model request probabilities on the cache hit ratio, as the estimated request probabilities used for model caching decision determination may deviate from the true value $p_{k,i}$. Following prior work, we model the estimated request probability of user k for model i, denoted by $p'_{k,i}$, as a Beta distribution with mean $p_{k,i}$ [65], [66]. In our experiments, the model caching decisions are first determined based on $p'_{k,i}$, while the cache hit ratio is evaluated through Monte Carlo experiments, where each user generates a specific model request according to $p_{k,i}$ in each Monte Carlo experiment. We analyze the cache hit ratio in terms of the coefficient of variation of $p'_{k,i}$.

Fig. 12 demonstrates that the cache hit ratio of the proposed algorithms decreases slightly as the coefficient of variation of $p'_{k,i}$ increases. Fig. 12(a) shows that the cache hit ratio of the TrimCaching Spec and TrimCaching Gen algorithms degrades by 2.20% and 2.78%, respectively, as the coefficient of variation of $p'_{k,i}$ increases from 0 to 0.5 in the special case. Besides, in the general case, the cache hit ratio of the TrimCaching Gen algorithm drops by 1.82% over the same range in Fig. 12(b). These results demonstrate that the proposed algorithms remain robust with only minor performance degradation even when the estimated request probabilities deviate from their true values substantially.

G. Comparison With Online Placement Strategies

In this subsection, we compare the proposed algorithms with two classical online content placement strategies, including the Least Recently Used (LRU) and the Least Frequently Used (LFU) strategies, under both the special and general cases. The experiments are conducted in a dynamic setting, where each user generates one specific model request according to $p_{k,i}$ at the beginning of each 10-second time slot. The placement decisions of the TrimCaching Spec and TrimCaching Gen algorithms are determined once at time 0 based on $p_{k,i}$ and remain unchanged throughout the experiment. In contrast, both LRU and LFU update their placements dynamically according to the user requests at the end of each time slot. Under LRU, models that are least recently requested are removed from the cache, while under LFU, those with the least request frequency are removed [67], [68]. We consider parameter sharing in both LRU and LFU implementations. The cache hit ratio is recorded at the start of each time slot to represent the performance for that slot.

As shown in Fig. 13, the proposed algorithms consistently outperform the LRU and LFU strategies. Despite using static placement decisions computed only once based on $p_{k,i}$ at the beginning, both the TrimCaching Spec and TrimCaching Gen algorithms achieve significantly higher cache hit ratios. Specifically, after the performance of the LRU and LFU strategies stabilizes (after 30 minutes), the TrimCaching Spec algorithm improves the cache hit ratio by an average of 33.71% and 21.61% over LRU and LFU, respectively, while the TrimCaching Gen algorithm achieves improvements of 22.17% and 10.07%, respectively, in the special case, as shown in Fig. 13(a). Moreover, a similar trend can be observed in Fig. 13(b), where the TrimCaching Gen algorithm outperforms LRU and LFU by 26.05% and 13.20% on average, respectively, in the general case.

VIII. Conclusion

In this paper, we proposed the TrimCaching framework for efficient edge AI model caching. By leveraging shared parameter blocks among AI models for efficient storage, we formulated the parameter-sharing AI model placement problem to maximize the cache hit ratio under user latency requirements and storage constraints in multi-edge scenarios. This paper is the first work to formulate such a problem and identify this problem as a submodular maximization problem

with submodular knapsack constraints, which is fundamentally different from conventional edge content caching. To tackle this challenge, we first investigated the special case with a small fixed number of shared parameter blocks independent of the problem scale. We developed a polynomial-time rounding DP-based algorithm with $(1-\epsilon)/2$-approximation guarantee. After that, we addressed the general case by devising a greedy algorithm that is also efficient and effective. Our findings underscore the importance of parameter-sharing model caching for enabling low-latency AI model downloading. We hope this work will inspire further research on edge AI model caching, such as model caching in heterogeneous or hierarchical networks, thereby making it a key service of 6G mobile networks.

REFERENCES

[1] G. Qu, Z. Lin, F. Liu, X. Chen, and K. Huang, "TrimCaching: Parameter-sharing AI model caching in wireless edge networks," in *Proc. IEEE 44th Int. Conf. Distrib. Comput. Syst. (ICDCS)*, Jul. 2024, pp. 36–46.

[2] J. Huang, K. Yuan, C. Huang, and K. Huang, "D^2-JSCC: Digital deep joint source-channel coding for semantic communications," *IEEE J. Sel. Areas Commun.*, vol. 43, no. 4, pp. 1246–1261, Apr. 2025.

[3] C. Wu, J. Chen, Z. Wang, R. Liang, and R. Du, "Semantic sleuth: Identifying Ponzi contracts via large language models," in *Proc. 39th IEEE/ACM Int. Conf. Automated Softw. Eng.*, Oct. 2024, pp. 582–593.

[4] S. Hu, Z. Fang, Y. Deng, X. Chen, Y. Fang, and S. Kwong, "Toward full-scene domain generalization in multi-agent collaborative bird's eye view segmentation for connected and autonomous driving," *IEEE Trans. Intell. Transp. Syst.*, vol. 26, no. 2, pp. 1783–1796, Feb. 2025.

[5] Q. Wu, X. Chen, Z. Zhou, and J. Zhang, "FedHome: Cloud-edge based personalized federated learning for in-home health monitoring," *IEEE Trans. Mobile Comput.*, vol. 21, no. 8, pp. 2818–2832, Aug. 2022.

[6] D. C. Nguyen et al., "Federated learning for smart healthcare: A survey," *ACM Comput. Surv.*, vol. 55, no. 3, pp. 1–37, Feb. 2022.

[7] M. Li, W. Ding, and D. Zhao, "Privacy risks in reinforcement learning for household robots," in *Proc. IEEE Int. Conf. Robot. Autom. (ICRA)*, May 2024, pp. 5148–5154.

[8] Q. Chen, X. Chen, and K. Huang, "FedMeld: A model-dispersal federated learning framework for space-ground integrated networks," *IEEE Trans. Mobile Comput.*, vol. 25, no. 6, pp. 8221–8234, Jun. 2026, doi: 10.1109/TMC.2025.3647858.

[9] C. Wu, J. Chen, K. He, Z. Zhao, R. Du, and C. Zhang, "EchoHand: High accuracy and presentation attack resistant hand authentication on commodity mobile devices," in *Proc. ACM SIGSAC Conf. Comput. Commun. Secur.*, Nov. 2022, pp. 2931–2945.

[10] Technical Specification Group Services and System Aspects; Study on Traffic Characteristics and Performance Requirements for AI/ML Model Transfer in 5GS; (Release 18), Standard (TS) 22.874, version 18.2.0, 3rd Generation Partnership Project (3GPP), Dec. 2021.

[11] J. Tan, A. Noulas, D. Saez, and R. Schifanella, "Notable site recognition using deep learning on mobile and crowd-sourced imagery," in *Proc. 21st IEEE Int. Conf. Mobile Data Manage. (MDM)*, Versailles, France, Jun. 2020, pp. 137–147.

[12] Y. Ma, S. Hu, Z. Fang, Y. Ji, Y. Deng, and Y. Fang, "Sense4FL: Vehicular crowdsensing enhanced federated learning for autonomous driving," *IEEE Trans. Mobile Comput.*, early access, Mar. 16, 2026. [Online]. Available: https://ieeexplore.ieee.org/document/11434993

[13] Q. Chen et al., "Space–ground fluid AI for 6G edge intelligence," *Engineering*, vol. 54, pp. 14–19, Nov. 2025.

[14] M. Chen, Z. Yang, W. Saad, C. Yin, H. V. Poor, and S. Cui, "A joint learning and communications framework for federated learning over wireless networks," *IEEE Trans. Wireless Commun.*, vol. 20, no. 1, pp. 269–283, Jan. 2021.

[15] T. Zhu et al., "Byzantine-robust federated learning via cosine similarity aggregation," *Comput. Netw.*, vol. 254, Dec. 2024, Art. no. 110730.

[16] Z. Wang, A. E. Kalør, Y. Zhou, P. Popovski, and K. Huang, "Ultra-low-latency edge inference for distributed sensing," *IEEE Trans. Wireless Commun.*, vol. 25, pp. 1908–1922, Aug. 2026.

[17] Z. Fang, S. Hu, J. Wang, Y. Deng, X. Chen, and Y. Fang, "Prioritized information bottleneck theoretic framework with distributed online learning for edge video analytics," *IEEE Trans. Netw.*, vol. 33, no. 3, pp. 1203–1219, Jun. 2025.

[18] A. Padmanabhan et al., "Gemel: Model merging for memory-efficient, real-time video analytics at the edge," in *Proc. USENIX Symp. Netw. Syst. Des. Implement. (NSDI)*, Boston, MA, USA, Apr. 2023, pp. 973–994.

[19] Z. Lin, G. Qu, Q. Chen, X. Chen, Z. Chen, and K. Huang, "Pushing large language models to the 6G edge: Vision, challenges, and opportunities," *IEEE Commun. Mag.*, vol. 63, no. 9, pp. 52–59, Sep. 2025.

[20] TensorFlow. (2023). *Transfer Learning & Fine-Tuning*. [Online]. Available: https://www.tensorflow.org/guide/keras/transfer_learning#introduction

[21] F. Zhuang et al., "A comprehensive survey on transfer learning," *Proc. IEEE*, vol. 109, no. 1, pp. 43–76, Jan. 2021.

[22] Y. Guo, H. Shi, A. Kumar, K. Grauman, T. Rosing, and R. Feris, "SpotTune: Transfer learning through adaptive fine-tuning," in *Proc. IEEE/CVF Conf. Comput. Vis. Pattern Recognit. (CVPR)*, Long Beach, CA, USA, Jun. 2019, pp. 4800–4809.

[23] Z. Fang et al., "PACP: Priority-aware collaborative perception for connected and autonomous vehicles," *IEEE Trans. Mobile Comput.*, vol. 23, no. 12, pp. 15003–15018, Dec. 2024.

[24] E. J. Hu et al., "LoRA: Low-rank adaptation of large language models," in *Proc. Int. Conf. Learn. Represent. (ICLR)*, Apr. 2022, pp. 1–13.

[25] K. He, X. Zhang, S. Ren, and J. Sun, "Deep residual learning for image recognition," in *Proc. IEEE Conf. Comput. Vis. Pattern Recognit. (CVPR)*, Jun. 2016, pp. 770–778.

[26] A. Krizhevsky, "Learning multiple layers of features from tiny images," Univ. Toronto, Toronto, ON, Canada, Apr. 2009. [Online]. Available: https://www.cs.toronto.edu/~kriz/learning-features-2009-TR.pdf

[27] J. Novikova, O. Dušek, and V. Rieser, "The E2E dataset: New challenges for end-to-end generation," in *Proc. 18th Annu. SIGdial Meeting Discourse Dialogue*, Aug. 2017, pp. 201–206.

[28] C. Gardent, A. Shimorina, S. Narayan, and L. Perez-Beltrachini, "Creating training corpora for NLG micro-planners," in *Proc. 55th Annu. Meeting Assoc. Comput. Linguistics (Long Papers)*, vol. 1, Vancouver, BC, Canada, Jul. 2017, pp. 179–188.

[29] L. Nan et al., "DART: Open-domain structured data record to text generation," in *Proc. Conf. North Amer. Chapter Assoc. Comput. Linguistics, Hum. Lang. Technol.*, Jun. 2021, pp. 432–447.

[30] K. Shanmugam, N. Golrezaei, A. G. Dimakis, A. F. Molisch, and G. Caire, "FemtoCaching: Wireless content delivery through distributed caching helpers," *IEEE Trans. Inf. Theory*, vol. 59, no. 12, pp. 8402–8413, Dec. 2013.

[31] K. Poularakis and L. Tassiulas, "On the complexity of optimal content placement in hierarchical caching networks," *IEEE Trans. Commun.*, vol. 64, no. 5, pp. 2092–2103, May 2016.

[32] M. Dehghan et al., "On the complexity of optimal request routing and content caching in heterogeneous cache networks," *IEEE/ACM Trans. Netw.*, vol. 25, no. 3, pp. 1635–1648, Jun. 2017.

[33] T. X. Vu, S. Chatzinotas, and B. Ottersten, "Edge-caching wireless networks: Performance analysis and optimization," *IEEE Trans. Wireless Commun.*, vol. 17, no. 4, pp. 2827–2839, Apr. 2018.

[34] D. Liu, B. Chen, C. Yang, and A. F. Molisch, "Caching at the wireless edge: Design aspects, challenges, and future directions," *IEEE Commun. Mag.*, vol. 54, no. 9, pp. 22–28, Sep. 2016.

[35] F. Gabry, V. Bioglio, and I. Land, "On energy-efficient edge caching in heterogeneous networks," *IEEE J. Sel. Areas Commun.*, vol. 34, no. 12, pp. 3288–3298, Dec. 2016.

[36] H. Wu et al., "Delay-minimized edge caching in heterogeneous vehicular networks: A matching-based approach," *IEEE Trans. Wireless Commun.*, vol. 19, no. 10, pp. 6409–6424, Oct. 2020.

[37] T. X. Vu, L. Lei, S. Vuppala, A. Kalantari, S. Chatzinotas, and B. Ottersten, "Latency minimization for content delivery networks with wireless edge caching," in *Proc. IEEE Int. Conf. Commun. (ICC)*, Kansas City, MO, USA, May 2018, pp. 1–6.

[38] G. Qu, Q. Chen, X. Chen, K. Huang, and Y. Fang, "PartialLoading: User scheduling and bandwidth allocation for parameter-sharing edge inference," 2025, *arXiv:2503.22982*.

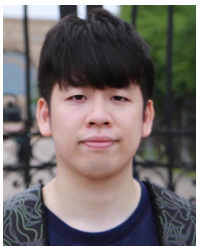
[39] D. Xu, T. Li, Y. Li, X. Su, S. Tarkoma, and P. Hui, "Edge intelligence: Architectures, challenges, and applications," 2020, *arXiv:2003.12172*.

[40] M. Xu et al., "Cached model-as-a-resource: Provisioning large language model agents for edge intelligence in space–air–ground integrated networks," *IEEE Trans. Netw.*, vol. 34, pp. 2850–2864, Jan. 2026.

[41] W. Wei, Z. Lin, T. Li, X. Li, and X. Chen, "Pipelining split learning in multi-hop edge networks," *IEEE Trans. Mobile Comput.*, early access, Apr. 17, 2026. [Online]. Available: https://ieeexplore.ieee.org/document/11482856

[42] Z. Lin, G. Qu, X. Chen, and K. Huang, "Split learning in 6G edge networks," *IEEE Wireless Commun.*, vol. 31, no. 4, pp. 170–176, Aug. 2024.
[43] N. Hudson, H. Khamfroush, and D. E. Lucani, "QoS-aware placement of deep learning services on the edge with multiple service implementations," in *Proc. Int. Conf. Comput. Commun. Netw. (ICCCN)*, Jul. 2021, pp. 1–8.
[44] K. Huang, H. Wu, Z. Liu, and X. Qi, "In-situ model downloading to realize versatile edge AI in 6G mobile networks," *IEEE Wireless Commun.*, vol. 30, no. 3, pp. 96–102, Jun. 2023.
[45] G. Qu, Q. Chen, W. Wei, Z. Lin, X. Chen, and K. Huang, "Mobile edge intelligence for large language models: A contemporary survey," *IEEE Commun. Surveys Tuts.*, vol. 27, no. 6, pp. 3820–3860, Dec. 2025.
[46] H. Wu, Q. Zeng, and K. Huang, "Efficient multiuser AI downloading via reusable knowledge broadcasting," *IEEE Trans. Wireless Commun.*, vol. 23, no. 8, pp. 10459–10472, Aug. 2024.
[47] X. Zhao, P. Yuan, H. li, and S. Tang, "Collaborative edge caching in context-aware device-to-device networks," *IEEE Trans. Veh. Technol.*, vol. 67, no. 10, pp. 9583–9596, Oct. 2018.
[48] P. Sermpezis, T. Giannakas, T. Spyropoulos, and L. Vigneri, "Soft cache hits: Improving performance through recommendation and delivery of related content," *IEEE J. Sel. Areas Commun.*, vol. 36, no. 6, pp. 1300–1313, Jun. 2018.
[49] A. Dan and D. Sitaram, "Multimedia caching strategies for heterogeneous application and server environments," *Multimedia Tools Appl.*, vol. 4, no. 3, pp. 279–312, May 1997.
[50] L. E. Chatzieleftheriou, M. Karaliopoulos, and I. Koutsopoulos, "Jointly optimizing content caching and recommendations in small cell networks," *IEEE Trans. Mobile Comput.*, vol. 18, no. 1, pp. 125–138, Jan. 2019.
[51] M. Zhang, H. Luo, and H. Zhang, "A survey of caching mechanisms in information-centric networking," *IEEE Commun. Surveys Tuts.*, vol. 17, no. 3, pp. 1473–1499, 3rd Quart., 2015.
[52] S. Fujishige, *Submodular Functions and Optimization*. New York, NY, USA: Elsevier, 2005.
[53] L. Lovász, *Mathematical Programming The State of the Art: Bonn 1982*. Berlin, Germany: Springer, 1983, ch. Submodular functions and convexity, pp. 235–257.
[54] S. H. S. Basha, S. K. Vinakota, V. Pulabaigari, S. Mukherjee, and S. R. Dubey, "AutoTune: Automatically tuning convolutional neural networks for improved transfer learning," *Neural Netw.*, vol. 133, pp. 112–122, Jan. 2021.
[55] PyTorch. *Models and Pre-Trained Weights*. [Online]. Available: https://docs.pytorch.org/vision/main/models.html
[56] Apple. (2024). *Introducing Apple's on-Device and Server Foundation Models*. Accessed: Jun. 2024. [Online]. Available: https://machinelearning.apple.com/research/introducing-apple-foundation-models
[57] IBM. (2024). *Serving Customized AI Models at Scale With LoRA*. [Online]. Available: https://research.ibm.com/blog/LoRAs-explained
[58] R. K. Iyer and J. A. Bilmes, "Submodular optimization with submodular cover and submodular knapsack constraints," in *Proc. Adv. Neural Inform. Process. Syst. (NeurIPS)*, Stateline, NV, USA, Dec. 2013, pp. 1–9.
[59] R. Iyer, S. Jegelka, and J. Bilmes, "Fast semidifferential-based submodular function optimization," in *Proc. Int. Conf. Mach. Learn. (ICML)*, Jun. 2013, pp. 855–863.
[60] Technical Specification Group Radio Access Network; NR; Base Station (BS) Radio Transmission and Reception; (Release 18), Standard (TS) 38.104, version 18.8.0, 3rd Generation Partnership Project (3GPP), Dec. 2024.
[61] GSMA. (2019). *Mobile Backhaul: An Overview*. [Online]. Available: https://www.gsma.com/futurenetworks/wiki/mobile-backhaul-an-overview/
[62] A. L. Rezaabad, H. Beyranvand, J. A. Salehi, and M. Maier, "Ultra-dense 5G small cell deployment for fiber and wireless backhaul-aware infrastructures," *IEEE Trans. Veh. Technol.*, vol. 67, no. 12, pp. 12231–12243, Dec. 2018.
[63] P. Liu, K. Luo, D. Chen, and T. Jiang, "Spectral efficiency analysis of cell-free massive MIMO systems with zero-forcing detector," *IEEE Trans. Wireless Commun.*, vol. 19, no. 2, pp. 795–807, Feb. 2020.
[64] G. K. Zipf, "Relative frequency as a determinant of phonetic change," *Harvard Stud. Classical Philol.*, vol. 40, pp. 1–95, 1929. [Online]. Available: https://www.jstor.org/stable/pdf/310585.pdf
[65] E. Bastug, M. Bennis, and M. Debbah, "Social and spatial proactive caching for mobile data offloading," in *Proc. IEEE Int. Conf. Commun. Workshops (ICC)*, Sydney, NSW, Australia, Jun. 2014, pp. 581–586.
[66] J. Zhu, X. Huang, and Z. Shao, "Learning-aided content placement in caching-enabled fog computing systems using Thompson sampling," in *Proc. IEEE Int. Conf. Acoust., Speech Signal Process. (ICASSP)*, Barcelona, Spain, May 2020, pp. 5060–5064.
[67] O. Serhane, K. Yahyaoui, B. Nour, and H. Moungla, "A survey of ICN content naming and in-network caching in 5G and beyond networks," *IEEE Internet Things J.*, vol. 8, no. 6, pp. 4081–4104, Mar. 2021.
[68] A. Seetharam, "On caching and routing in information-centric networks," *IEEE Commun. Mag.*, vol. 56, no. 3, pp. 204–209, Mar. 2018.
[69] H. Kellerer, R. Sarto Basso, and V. A. Strusevich, "Approximability issues for unconstrained and constrained maximization of half-product related functions," *Theor. Comput. Sci.*, vol. 659, pp. 64–71, Jan. 2017.
[70] M. Dawande, J. Kalagnanam, P. Keskinocak, F. S. Salman, and R. Ravi, "Approximation algorithms for the multiple knapsack problem with assignment restrictions," *J. Comb. Optim.*, vol. 4, pp. 171–186, 2000.
[71] C. Chekuri and S. Khanna, "A polynomial time approximation scheme for the multiple knapsack problem," *SIAM J. Comput.*, vol. 35, no. 3, pp. 713–728, 2005.

Guanqiao Qu (Graduate Student Member, IEEE) received the B.E. and M.E. degrees from Harbin Institute of Technology (HIT), Harbin, China, in 2020 and 2022, respectively. He is currently pursuing the Ph.D. degree with the Department of Electrical and Computer Engineering, The University of Hong Kong (HKU), Hong Kong, China. His research interests include wireless communications, networking, edge intelligence, and machine learning.



Zheng Lin (Graduate Student Member, IEEE) received the B.Eng. degree in electronic information engineering from Fuzhou University in 2020 and the M.Eng. degree in electrical and computer engineering from Fudan University in 2023. He is currently pursuing the Ph.D. degree with the Department of Electrical and Computer Engineering, The University of Hong Kong, Hong Kong, China. His research interests include wireless networking, edge intelligence, and distributed machine learning. He was awarded the WASA Best Paper Award, China Hi-Tech Fair Outstanding Science & Technology Innovation Award, China Institute of Communications Smart Tower Outstanding Innovation Practice in 2025, and Fudan Outstanding Graduate and Shanghai Outstanding Graduate Award in 2023. He serves as a Young Professional for the IEEE Vehicular Technology Society (VTS) Ad Hoc Committee and as a Technical Program Committee Member for several top-tier conferences, including ICML, ICLR, NeurIPS, ACM MM, AAAI, and IJCAI.

Qian Chen (Member, IEEE) received the B.E. and Ph.D. degrees in information and communication engineering from Harbin Institute of Technology (HIT), Harbin, China, in 2018 and 2023, respectively. She is currently a Post-Doctoral Fellow with the Department of Electrical and Computer Engineering, The University of Hong Kong (HKU), Hong Kong. From 2021 to 2022, she was a Visiting Ph.D. Student with the Information Systems Technology and Design (ISTD) Pillar, Singapore University of Technology and Design (SUTD), Singapore. Her current research interests include wireless networking, edge intelligence, and space-air-ground integrated networks. She received the Outstanding Doctoral Dissertation Nomination Award in 2024 from China Education Society of Electronics and the HKU University Research Committee (URC) Post-Doctoral Fellow Scheme in 2025.

Jian Li (Senior Member, IEEE) received the bachelor's degree from the Department of Electronics and Information Engineering, Anhui University, in 2015, and the Ph.D. degree from the Department of Electronic Engineering and Information Science, University of Science and Technology of China (USTC), in 2020. From November 2019 to November 2020, he was a Visiting Scholar with the Department of Electronic and Computer Engineering, University of Florida. From December 2020 to October 2022, he was a Post-Doctoral Researcher with the School of Cyber Science and Technology, USTC, where he was an Associate Researcher with the School of Cyber Science and Technology from November 2022 to March 2026. He is currently an Associate Professor with the School of Cyber Science and Technology, USTC. His research interests include quantum networking, network security, and wireless networks. He serves as an Editor for IEEE TRANSACTIONS ON COMMUNICATIONS and *China Communications*. He has served as the Technical Program Committee Co-Chair for IEEE IWCMC 2025 and 2026 QNC Symposium. He was a recipient of IEEE ComSoc Internet Technical Committee Early Career Award in 2025.

Fangming Liu (Senior Member, IEEE) received the B.Eng. degree from Tsinghua University, Beijing, and the Ph.D. degree from The Hong Kong University of Science and Technology, Hong Kong. He is currently a Full Professor with the Huazhong University of Science and Technology, Wuhan, China. His research interests include cloud computing and edge computing, datacenter and green computing, SDN/NFV/5G, and applied ML/AI. He received the National Natural Science Fund (NSFC) for Excellent Young Scholars and the National Program Special Support for Top-Notch Young Professionals. He was a recipient of the Best Paper Award of IEEE/ACM IWQoS 2019, ACM e-Energy 2018, and IEEE GLOBECOM 2011; the First Class Prize of Natural Science of Ministry of Education in China; and the Second Class Prize of National Natural Science Award in China.

Xianhao Chen (Member, IEEE) received the B.Eng. degree in electronic information from Southwest Jiaotong University in 2017 and the Ph.D. degree in electrical and computer engineering from the University of Florida in 2022. He is currently an Assistant Professor with the Department of Electrical and Computer Engineering, The University of Hong Kong, where he directs the Wireless Information & Intelligence (WILL) Laboratory. His research interests include wireless networking, edge intelligence, and machine learning. He serves on the technical program committees for flagship international conferences, including ACM MobiHoc, IEEE GLOBECOM, and IEEE ICC; and on the editorial boards of *ACM Computing Surveys*, IEEE TRANSACTIONS ON NETWORKING, and IEEE TRANSACTIONS ON VEHICULAR TECHNOLOGY. He received the Early Career Award from the Research Grants Council (RGC) of Hong Kong in 2024, the ECE Graduate Excellence Award for research from the University of Florida in 2022, and the ICCC Best Paper Award in 2023.

Kaibin Huang (Fellow, IEEE) received the B.Eng. and M.Eng. degrees in electrical engineering from the National University of Singapore and the Ph.D. degree from The University of Texas at Austin. He is currently the Philip K. H. Wong Wilson K. L. Wong Professor in Electrical Engineering and the Head of the Department of Electrical and Computer Engineering, The University of Hong Kong (HKU), Hong Kong. His work was recognized with seven best paper awards from the IEEE Communication Society. He has served on the editorial boards for five major journals in the area of wireless communications and co-edited twelve journal special issues. He has been named as a Highly Cited Researcher by Clarivate in the last seven years (2019–2025) and an AI 2000 Most Influential Scholar (Top 30 in Internet of Things) in 2023 and 2025. He received the 2025 IEEE Wireless Communications Technical Committee Recognition Award. He is a member of the Engineering Panel of Hong Kong Research Grants Council. He was an IEEE Distinguished Lecturer (2020–2022). He is also a Fellow of U.S. National Academy of Inventors in 2024 and a Croucher Senior Research Fellow in 2026.