Calibrated Bandit Learning for Decentralized Task Offloading in Ultra-Dense Networks

Rui Zhang, Peng Cheng, *Member, IEEE*, Zhuo Chen, *Senior Member, IEEE*, Sige Liu, Branka Vucetic, *Life Fellow, IEEE*, and Yonghui Li, *Fellow, IEEE*

***Abstract*— The integration of mobile edge computing (MEC) into an ultra-dense network (UDN) can provide ubiquitous task offloading services to computation-demanding users leveraging densely deployed micro base stations. The conventional multi-user task offloading strategies are performed centrally, where a central node makes global task offloading decisions on server selection and resource allocation. In practice, the deployment becomes prohibitively complex with the increasing number of users as it involves high communication overhead and complex global optimization operations. In this paper, we develop a novel decentralized task offloading strategy in UDN, enabling users to independently make local task offloading decisions. We formulate the associated optimization problem to minimize the long-term average task delay among all users. On this basis, we develop a novel calibrated contextual bandit learning (CCBL) algorithm, where users can learn the computational delay functions of micro base stations and predict the task offloading decisions of other users in a decentralized manner. The convergence of the proposed CCBL algorithm is verified via the approachability theory. Moreover, we transfer the target of calibrated learning from all micro base stations to a single user and propose a user-oriented CCBL algorithm to further decrease the computational complexity and increase the convergence rate. Simulation results illustrate that our proposed algorithm outperforms the existing decentralized algorithms and approaches the centralized one.**

***Index Terms*— Decentralized task offloading, ultra-dense networks, contextual bandit learning, calibrated learning.**

I. Introduction

THE rapid development of 5G and beyond has motivated a great number of new mobile computing applications, such as wearable virtual reality [1], health monitoring [2], and autonomous driving [3], which generate a large volume of computationally intensive delay-sensitive tasks at mobile users. Due to the inherent limitation of computational capacity and battery life, mobile users alone are unlikely to complete these tasks in a timely manner. As a concept of paradigm-shift, mobile edge computing (MEC) [4]–[6] was proposed to push mobile computing and data analysis to the network edge close to the mobile users. In this case, mobile users are allowed to offload their computation tasks to the MEC servers, enjoying the benefits of fast task execution with low computation and transmission latencies. Furthermore, it has been widely accepted that offloading the time-sensitive tasks generated by applications such as AR/VR [7], [8] to MEC servers for lower computational delay is a key technology of 5G and beyond [9].

Many multi-user centralized task offloading schemes have been proposed in [10]–[12], where the task features (e.g., the data and computation amounts for the task execution) are uploaded to a central node, such as a macro base station (MaBS), for global decision making on the MEC server selection and computational resource allocation at each MEC server. In such global decision making, the task offloading decision was usually framed as a convex/non-convex optimization problem. In [10], the optimal computational resource allocation was formulated as a convex optimization problem, aiming to minimize the weighted sum of transmission energy consumption of mobile users under the constraint on computation latency. The authors in [11] proposed a deep-Q network-based task offloading and a resource allocation algorithm to minimize the overall offloading cost in terms of energy consumption, computation requirements, and delay. To maximize a weighted sum of reductions in the task delay and energy consumption, the authors in [12] formulated the problem as a mixed integer nonlinear program. This program jointly optimizes the task offloading decision, uplink transmission power of mobile users, and computing resource allocation at the MEC server. With the network evolution, the MEC services will be offered in an ultra-densely network (UDN) [13]–[16] with ultra-dense deployed microcell base stations (MiBSs), enabling ubiquitous computation offloading capabilities for mobile users. However, due to the interaction between multiple users and MEC servers, the centralized offloading strategy is highly inefficient, which is further exacerbated by the high communication overhead due to the task features of a large number of mobile users. Finally, the global optimization operation at the central node is inevitably complex, resulting in a delayed decision delivery.

Manuscript received September 5, 2021; revised December 21, 2021; accepted February 7, 2022. Date of publication February 16, 2022; date of current version April 18, 2022. The work of Peng Cheng was supported by ARC under Grant DE190100162 and DP210103410. The work of Yonghui Li was supported by ARC under Grant DP190101988 and DP210103410. The associate editor coordinating the review of this article and approving it for publication was C. R. Murthy. *(Corresponding authors: Peng Cheng; Yonghui Li.)*

Rui Zhang is with the Department of Information Engineering, The Chinese University of Hong Kong, Hong Kong, China (e-mail: ruizhang@cuhk.edu.hk).

Peng Cheng is with the Department of Computer Science and Information Technology, La Trobe University, Melbourne, VIC 3086, Australia, and also with The University of Sydney, Sydney, NSW 2006, Australia (e-mail: p.cheng@latrobe.edu.au; peng.cheng@sydney.edu.au).

Zhuo Chen was with CSIRO DATA61, Sydney, NSW 1710, Australia (e-mail: zhuo.chen@ieee.org).

Sige Liu, Branka Vucetic, and Yonghui Li are with the School of Electrical and Information Engineering, The University of Sydney, Sydney, NSW 2006, Australia (e-mail: sige.liu@sydney.edu.au; branka.vucetic@sydney.edu.au; yonghui.li@sydney.edu.au).

Color versions of one or more figures in this article are available at https://doi.org/10.1109/TCOMM.2022.3152262.

Digital Object Identifier 10.1109/TCOMM.2022.3152262

As a stark contrast with the centralized policy, in a distributed policy, no user can exchange observations but make decisions independently based on their individual local observations [17], [18]. In other words, there is no information sharing among users, thus no communication overhead is introduced. However, the lack of information sharing is very likely to result in conflict among users. To enhance the collaboration among users with minimal communication overhead, decentralized policies are introduced in [19]–[21]. A decentralized policy relies on users to make scheduling decisions locally without information sharing among users. However, different from a distributed policy, historic information sharing exists after decision makings. This information does not affect the task offloading decisions in the current round but enables users to learn peer behaviors more effectively. Therefore, users can predict the scheduling decisions of others in the future and avoid conflicts.

In this paper, we develop a novel decentralized task offloading strategy in a UDN, where local task offloading decisions are made by users independently without information sharing among them. Different from the existing decentralized strategies [19]–[21], we first consider the competition of computational resources among users. This requires an advanced strategy to reduce the conflict. We further consider various task features, such as the size of temporary variables generated during the processing, which are highly related to the computational delay of tasks. Mathematically, we formulate the task offloading as an optimization problem, aiming to minimize the long-term average task delay of all users with a restricted number of MiBSs. To achieve the decentralized offloading, we decouple the optimization problem into multiple identically independent bandit learning problems, where each user minimizes its own long-term average task delay and learn the computational delay function related to task features. Meanwhile, we develop a calibrated forecaster for each user to predict the task offloading outcomes of others to reduce conflict.

Specifically, we develop a novel contextual calibrated learning (CCBL) algorithm by fusing the bandit learning and calibrated learning. For the first few tasks (initialization), users will offload at least one to the MiBS, whose computational delay function is totally unknown *a priori*. In the following, we will introduce three core steps and an auxiliary one in each task offloading round after the initialization. When a task is generated at a user, in the first step, the user selects an MiBS locally and offloads the task to this MiBS. This task offloading decision strikes a balance between exploration (offloading the task to a MiBS whose computational delay function is not well-learned) and exploitation (leveraging the prediction of other users' task offloading decisions to minimize the delay of this task). In the second step, the MaBS monitors the task offloading decisions of all users in this round and broadcasts such information. In this way, all users within the coverage area can receive the task offloading decisions of the others. Based on this knowledge, in the third step, the calibrated forecaster at each user estimates other users' decisions in the next round. The prediction will assist each user in offloading its task for the next round. The auxiliary step follows the return of the task result, and in this step, each user updates its estimate of the computational delay function involving the selected MiBS and the task feature in this round. Following the four steps, CCBL reduces conflict with others and enables a user to minimize the long-term average task delay.

Under the CCBL algorithm, we also discuss the convergence of the proposed calibrated forecasters and prove that the convergence rate decreases with the increasing number of users and MiBSs. To further decrease the computational complexity and convergence rate of CCBL, we transfer the learning target of a calibrated forecaster from all MiBSs to only a single user and propose a user-oriented CCBL (UOCCBL) algorithm. Simulation results show that the proposed CCBL algorithm outperforms the existing decentralized task offloading strategies and that it is only slightly inferior to the centralized one.

A. Contribution

The fusion of communication and computing is a key enabler in 5G and beyond networks [22], and how to reduce the communication/computing latency for computational-intensive applications is the major bottleneck and a contemporarily significant topic [23]. This paper incorporates the mobile edge computing into 5G, and the main contributions of this paper can be summarized as follows.

- We develop a novel decentralized task offloading scheme in UDN, enabling users to minimize the long-term average task delay without the need for computational capabilities of MiBSs or the offloading decisions of other users.
- We propose a novel CCBL algorithm by fusing bandit learning and calibrated learning, enabling users to simultaneously learn the computational delay functions of MiBSs and predict the task offloading decisions of others. We also analyze the convergence rate of the proposed CCBL algorithm.
- We propose a UOCCBL algorithm to decrease the computational complexity by transferring the learning target of the calibrated forecaster from all MiBSs to a single user.

The rest of the paper is organized as follows. In Section II, we present the UDN task offloading model and formulate the decentralized contextual task offloading problem. In Section III, we decouple the problem to independent contextual MAB problems and elaborate on the proposed CCBL algorithm for decentralized task offloading. An extended task offloading algorithm UOCCBL is proposed in Section IV to decrease the computational complexity. Simulation results are presented in Section V followed by conclusions in Section VI. For convenience, we list the most important symbols in Table I.

II. System Model and Problem Formulation

A. System Model

We consider a UDN composed of one MaBS, N_B MiBSs, and N_U randomly located users shown in Fig. 1.

TABLE I
LIST OF SYMBOLS

Symbol	Definition
b, N_B, $\mathcal{B}$	The index, total number, and the set of MiBSs.
u, N_U, $\mathcal{U}$	The index, total number, and the set of users.
r, N_R, $\mathcal{R}$	The index, total number, and the set of task rounds.
x, N_X, $\mathcal{X}$	The index, total number, and the set of task features.
h, N_H, $\mathcal{H}$	The index, total number, and the set of task offloading outcomes from the view of a user.
s, N_ϵ, $\mathcal{S}=\Delta(\mathcal{H})$	The index, total number, and the set of probability distributions over task offloading outcomes from the view of a user.
$\mathcal{B}_u$, $\mathcal{U}_b$	The set of MiBSs that cover user u, the set of users that are covered by MiBS b.
$t^{\text{tra}}_{u,b}(r)$, $t^{\text{com}}_{u,b}(r)$, and $t_{u,b}(r)$	The transmission, computational, and total delay of u offloading its task to b in r.
$t^{(h)}_{u,b}(r)$, $\tilde{t}^{(h)}_{u,b}(r)$	The value and the estimation of $t_{u,b}(r)$ when the h-th outcome happens.
$z_{u,b}(r)\in\{0,1\}$	The indicator that indicates whether user u offloads its task to MiBS b in round r.
$\boldsymbol{z}_u(r)=[z_{u,0}(r),z_{u,1}(r),\ldots,z_{u,N_B}(r)]$	The task offloading decision of user u in round r.
$\boldsymbol{Z}(r)=\{\boldsymbol{z}_u(r)\}_{u\in\mathcal{U}}$	The task offloading decisions of all users in round r.
$\boldsymbol{h}_u(r)\in\mathcal{H}$	The task offloading outcome in round r from the view of user u.
$s^{(h)}_u(r)$	The probability of task offloading outcome h in round r from the view of user u.
$\boldsymbol{s}_u(r)=[s^{(1)}_u(r),...,s^{(N_H)}_u(r)]^T$	The probabilities of task offloading outcomes in round r from the view of user u.
$\boldsymbol{\zeta}_h$, $\boldsymbol{p}_s$	The h-th vector in $\mathcal{H}$, and the s-th vector in $\mathcal{S}$.
$H(r)$, $S(r)$	The value of index h and s in round r.

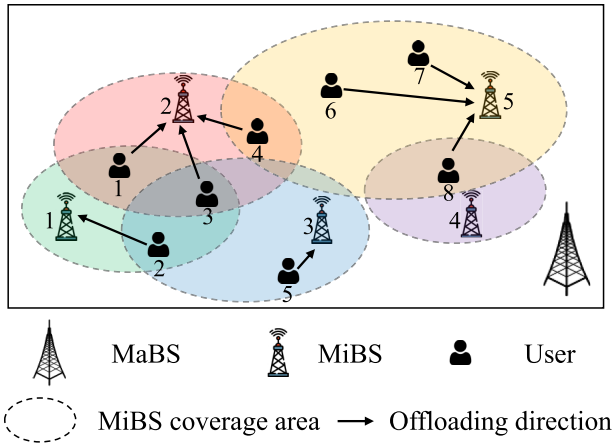


Fig. 1. An example of UDN, where $N_B=5$ and $N_U=8$.

Let $b\in\mathcal{B}=\{1,\ldots,N_B\}$ denote the MiBS index, $u\in\mathcal{U}=\{1,\ldots,N_U\}$ the user index, and $r\in\mathcal{R}=\{1,\ldots,N_R\}$ the task round index. Tasks are generated by all users at the beginning of each task round r, and the time duration of each round is t_0. In this paper, we adopt a binary task computation offloading mode, where each task is executed as a whole either locally or remotely at a nearby MiBS.

As shown in Fig. 1, the coverage areas of the MiBSs are limited by their transmission powers, and may overlap with each other due to the ultra-dense deployment of MiBSs. Thus, a user may be covered by multiple MiBSs simultaneously and may offload its tasks to any of them for fast execution. In addition, all the users are covered by the MaBS with wide coverage. Note that the MaBS does not provide computational resources, but monitors network activities and broadcasts the task offloading decisions of all users. The data size of task offloading decisions is only several bits, thus the introduced network overhead can be ignored. Specifically, they can be transmitted by the Physical Broadcast Channel (PBCH) [24] defined in 4G-LTE/5G protocols.

1) Communication Model: We first introduce the communication model. We define a vector $\boldsymbol{z}_u(r)=[z_{u,0}(r),z_{u,1}(r),\ldots,z_{u,N_B}(r)]$ to indicate the task offloading decisions. When $b>0$, $z_{u,b}(r)=1$ refers to user u offloading its task to MiBS b in round r, and $z_{u,b}(r)=0$ otherwise. When $b=0$, $z_{u,0}(r)=1$ refers to local computing. For simplicity, $b=0$ denotes local computing hereafter. We assume that each task can be offloaded to at most one MiBS, and we have

$$\sum_{b\in\{\mathcal{B}_u,0\}} z_{u,b}(r)=1,\quad \forall u\in\mathcal{U},\ r\in\mathcal{R}, \tag{1}$$

where $\mathcal{B}_u$ denotes the set of MiBSs that cover user u. Similarly, we denote by $\mathcal{U}_b$ the set of users that are covered by MiBS b. For example, as shown in Fig. 1, we have $\mathcal{B}_3=\{1,2,3\}$ and $\mathcal{U}_2=\{1,3,4\}$.

The tasks are transmitted from users to MiBSs through wireless uplink channels. The single-carrier frequency division multiple-access (SC-FDMA) technique is adopted, which is the multiple access scheme for uplink in the Long Term Evolution [25].[1] The uplink subcarriers are orthogonal, and each is assumed to have the bandwidth ω_0. The uplink data rate of user u offloading its task in round r to MiBS b is given by [27]

$$\upsilon_{u,b}(r)=\omega_0\log_2\left(1+\frac{p_u g_{u,b}(r)}{\sigma_0^2}\right), \tag{2}$$

where p_u is the fixed transmission power of user u, $g_{u,b}(r)$ is the power gain between user u and MiBS b, and σ_0^2 is the variance of additive white Gaussian noise (AWGN). Specifically, we have $g_{u,b}(r)=\alpha_{u,b}|h_{u,b}(r)|^2$, where $\alpha_{u,b}=128.1+37.6\log_{10}d_{u,b}$ (dB) is the large-scale fading gain following the

[1]To obtain channel state information, we can use the channel estimation methods provided in [26], which include least square (LS) estimator, the finite impulse response (FIR) algorithm, the least minimum mean square error (LMMSE) estimator, and the Gauss-Markov estimator. These methods estimate the channel states data with the help of reference symbols in SC-FDMA.

3GPP path loss model [28], $d_{u,b}$ is the transmission distance between user u and MiBS b, and $h_{u,b}(r) \sim \mathcal{CN}(0,1)$ is the small-scale fading coefficient. The transmission delay of user u offloading its task in round r to MiBS b can be given by

$$t^{\text{tra}}_{u,b}(r) = \begin{cases} c_u(r)/\upsilon_{u,b}(r), & \text{if } b > 0, \\ 0, & \text{if } b = 0, \end{cases} \tag{3}$$

where $c_u(r)$ is the data size of task at user u in round r. Without loss of generality, we assume that the data size of tasks at each user is fixed, i.e., $c_u(r) = c_u$, $\forall r \in \mathcal{R}$. Similar to many studies [29], we also assume that the data size of the task results after processing is usually much smaller than that of the task so that the downlink transmission delay for returning the results can be omitted.

2) Computing Model: In the following, we elaborate on our computing model. Instead of adopting the conventional two-variable computing model [4], [27], which only considers the task size and the CPU frequency of the base stations, in this paper, we consider a more general multi-variable one. In fact, the computing time of a task is determined by the features of the task and the associated MiBS. The former may include the data size of tasks, the tasks' demands on computing cycles, the size of temporary variables generated during the processing, the task type (e.g., data compression, photo rendering, and audio decoding), etc. The latter may include the frequency of the processing units, the number of processing units, the memory size, the availability of parallel computing, etc [30]. Among the above features, only the data size of tasks is linearly related to the computing time, while the relationships between other features and the computing time is not linear. On this basis, we define the task computational delay as $c_u(r)f(b,\psi_u(r))$, where $\psi_u(r) \in \mathcal{X} = \{1,\ldots,N_X\}$ is the feature of task in round r at user u. The computational delay density function $f(b,x)$ refers to the computing time when MiBS b processes an one-bit task with feature $x \in \mathcal{X}$ using full computing resources. Similarly, we define $f_u(x)$ as the computing time when user u processes an one-bit task with feature $x \in \mathcal{X}$ locally.

As an MiBS may process several tasks simultaneously from multiple users, the computational delay of user u offloading its task to MiBS b in round r can be calculated as

$$t^{\text{com}}_{u,b}(r) = \begin{cases} c_u(r)f(b,\psi_u(r))/\kappa_{u,b}(r), & \text{if } b > 0, \\ c_u(r)f_u(\psi_u(r)), & \text{if } b = 0, \end{cases} \tag{4}$$

where $\kappa_{u,b}(r)$ is the fraction that MiBS b allocates its computing resources to user u in round r and satisfies $\sum_{u\in\mathcal{U}} \kappa_{u,b}(r) \leqslant 1$. The specific allocation policy is determined by each MiBS and may not be the same at all MiBSs. Clearly, if too many users choose to offload their tasks to the same MiBS, a significantly increased computational delay will be introduced. A well-designed computational resource allocation strategy at the MiBSs can effectively reduce the task failure occurrence.

B. Problem Formulation

In this section, we formulate the UDN task offloading problem, which aims to minimize the long-term average task delay among all users with a restricted number of MiBSs. We adopt the average delay as the performance metric as it is one of the most significant performance indicators in 5G-MEC [11], [12]. The task delay of user u offloading its task to MiBS b in round r can be given by

$$t_{u,b}(r) = \begin{cases} t^{\text{tra}}_{u,b}(r) + t^{\text{com}}_{u,b}(r), & \text{if } t^{\text{tra}}_{u,b}(r) + t^{\text{com}}_{u,b}(r) \leqslant t_0, \\ \eta t_0, & \text{otherwise.} \end{cases} \tag{5}$$

In the first case, the task result is returned to user u or the task is completed locally before the end of round r. Thus the task delay is the summation of transmission and computing delay (the transmission delay of local computing equals to 0 according to (3)). In the second case, the task can not be completed before a pre-defined deadline [31], e.g., end of round r, then MiBS b will drop the unfinished task and send a message containing a "failure" indicator back to user u. Specially, as user u can accurately predict local computing time $t_{u,b}(r)$ with the knowledge of $f_u(x)$, it will always offload task to MiBSs if the local computing can not be completed on time. The user will set $t_{u,b}(r) = \eta t_0$ as penalty, where η is the penalty coefficient. Similar to the return of task results, the transmission of failure indicators is assumed to be error-free, and the transmission time is omitted.

Mathematically, the task offloading problem can be formulated as $\mathcal{P}$:

$$\min_{z_{u,b}(r)\in\{0,1\}} \Big\{ \lim_{N_R\to\infty} \frac{1}{N_R} \sum_{r\in\mathcal{R}} \sum_{u\in\mathcal{U}} \sum_{b\in\{\mathcal{B}_u,0\}} z_{u,b}(r) t_{u,b}(r) \Big\}$$

$$s.t. \quad \sum_{b\in\{\mathcal{B}_u,0\}} z_{u,b}(r) = 1, \quad \forall u \in \mathcal{U},\ r \in \mathcal{R},$$

$$z_{u,b}(r) \in \{0,1\}, \quad \forall u \in \mathcal{U},\ b \in \{\mathcal{B}_u, 0\},\ r \in \mathcal{R},$$

where the two constraints ensure that each task of user u can be offloaded to at most one MiBS that covers u. Here, we would like to highlight that the problem $\mathcal{P}$, different from the conventional static optimization, is quite challenging. In $\mathcal{P}$, the computational delay function $f(b,x)$ involved in $t_{u,b}(r)$ is unknown. Therefore, it is impossible to design a task offloading policy by solving a static nonconvex nonlinear integer programming problem like in [32]. A possible solution is to learn $f(b,x)$ through task offloading and observing the resultant delay $t^{\text{com}}_{u,b}(r)$. In this context, we can develop a centralized task offloading policy by utilizing a collaborative multi-agent multi-armed bandit (MAB) learning [27]. However, centralized decision making is required, collecting all users' task features $\psi_u(r)$ and then returning the task offloading decisions $\boldsymbol{z}_u(r)$. Clearly, a significant transmission delay may be introduced, and this is highly undesirable in practice.

III. Calibrated Contextual Bandit Learning

A. Motivation

The above limitations motivate us to develop a decentralized task offloading policy without peer-to-peer information exchange, where all users can make decisions locally. The key is to decouple $\mathcal{P}$ into N_U independent contextual MAB problems; each user minimizes its own long-term average task delay $\lim_{N_R\to\infty} \frac{1}{N_R} \sum_{r\in\mathcal{R}} \sum_{b\in\{\mathcal{B}_u,0\}} z_{u,b}(r) t_{u,b}(r)$, where

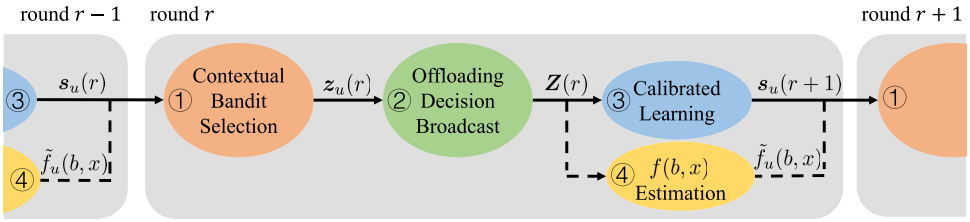


Fig. 2. Flowchart diagram of the CCBL scheme implemented at user u.

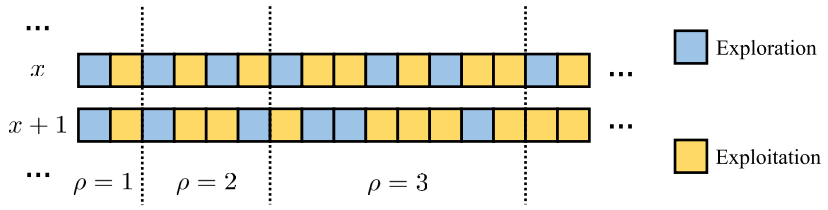


Fig. 3. Exemplary exploration-exploitation trade-off trail at bandit selection. If a user adopts the trail as shown in Fig. 4, and we pick up the rounds that the task feature is x. Then we can find that among the selected rounds, the indexes of rounds used for exploration are $1, 3, 5, 7, 10, 12, 15, \ldots$ and the indexes of rounds used for exploitation are $2, 4, 6, 8, 9, 11, 13, 14, 16, \ldots$

$f(b,x)$ can also be learned during this process. However, the challenge is that here user u has no knowledge of $\kappa_{u,b}(r)$ in $t_{u,b}(r)$ (c.f. (4)) due to the lack of the number of tasks offloaded to MiBS b in round r. This motivates us to develop a calibrated forecaster for user u to predict the task offloading outcomes, which reveal the number of tasks offloaded to each MiBS.

We denote by $\boldsymbol{h}_u(r) \in \mathcal{H}_u$ the task offloading outcome in round r from the view of user u, i.e., the combination of all other users' task offloading decisions in round r except user u. Let $N_H \leqslant (N_B)^{N_U-1}$ denote the size of $\mathcal{H}_u$, determined by the topology of UDN. In the proposed calibrated forecaster, the task offloading outcomes are learned via analyzing the outcomes in the previous rounds $\{\boldsymbol{h}_u(r')\}_{1\leqslant r'<r}$, and updating the prediction results continuously.

To achieve this decentralized task offloading policy, we propose a novel CCBL algorithm shown in Fig. 2, which includes three core steps (Steps ①②③) and an auxiliary one (Step ④). On a high level, in Step ① when tasks are generated at users in round r, user u selects an MiBS $b \in \mathcal{B}_u$ locally to perform task offloading, leveraging the information obtained from the last round. In Step ②, the MaBS monitors the task offloading decisions $\boldsymbol{Z}(r) = \{\boldsymbol{z}_u(r)\}_{u\in\mathcal{U}}$ and broadcasts this information. In Step ③, each user u transfers $\boldsymbol{Z}(r)$ to $\boldsymbol{h}_u(r)$, which denotes the task offloading outcome in round r (the task offloaded by user u is excluded). Then user u analyzes $\{\boldsymbol{h}_u(r')\}_{1\leqslant r'\leqslant r}$ and generates $\boldsymbol{s}_u(r+1) = [s_u^{(1)}(r+1), \ldots, s_u^{(N_H)}(r+1)]^T$, which represent the estimated probabilities of the task offloading outcomes in round $r+1$, where $s_u^{(h)}(r+1)$ denotes the probability that task offloading outcome h happens in round $r+1$. The prediction will assist user u to offload its task for round $r+1$ starting from Step ①. In Step ④, with the knowledge of $\boldsymbol{Z}(r)$, each user u updates $\tilde{f}_u(b,x)$, which is its estimate of the computational delay function $f(b,x)$. In the following, we will elaborate on these steps, and conduct the theoretical analysis of the convergence rate.

B. The CCBL Algorithm

1) Contextual Bandit Selection (Step ①): When a task is generated at user u in round r, an MiBS is selected by the user locally. Ideally, if the computational delay function $f(b,x)$ of all MiBSs is known *a priori*, user u should always choose the MiBS that is expected to result in the minimal delay $t_{u,b}(r)$. However, in practice, users may not have the knowledge of $f(b,x)$ and therefore need to learn it through task offloading. Clearly, it is necessary to strike a trade-off in the MiBS selection, targeting an MiBS whose $f(b,x)$ is not well learned (exploration), or a MiBS which is believed to result in the minimal $t_{u,b}(r)$ (exploitation).

To address this trade-off, we resort to the contextual bandit learning theory [33]. We first design multiple decision trails, one for each task feature, which can ensure that with the increase of N_R, the proportion of the task offloading chances reserved for exploration becomes smaller. An example of the decision trails is shown in Fig. 3, where for each task feature x, the offloading chances are divided into group $\rho = 1, 2, \ldots,$ and the number of rounds in group ρ is $J_\rho = 2^\rho$. In each group, ρ task offloading chances are used for exploration, while the others are for exploitation. Note that the first task offloading chance of each task feature x is used for exploration due to the lack of the knowledge of $f(b,x)$. When a task is generated at user u in round r, the user will allocate the decision trail for $x = \psi_u(r)$ and select between exploration and exploitation according to the trail. In the following, we will elaborate on how to select MiBSs in exploration and exploitation.

For exploration, let $C(b,x)$ denote the number of times that a user has selected MiBS b for tasks with feature x before round r. Each user aims to learn $f(b,x)$ via offloading tasks to MiBS b with the minimal $C(b,x)$. Therefore, user u should select the following MiBS to perform task offloading in round r

$$b_u(r) = \arg\min_{b\in\mathcal{B}_u} C(b, \psi_u(r)). \tag{6}$$

For exploitation, each user u computes the expected task delay of all MiBSs and offloads its task to MiBS $b_u(r)$ with the minimal expected delay in round r. As shown in Fig. 2, $\boldsymbol{s}_u(r)$ is generated by the calibrated forecasters at Step ③ in the last round ($r-1, r \geqslant 2$). As a special case, the task in the first round ($r=1$) is offloaded to a random MiBS for exploration. Based on the estimation of $\boldsymbol{s}_u(r)$, user u can estimate the delay of offloading its task to MiBS b in round r as

$$\tilde{t}_{u,b}(r) = \begin{cases} \sum_{h\in\mathcal{H}} s_u^{(h)}(r)\tilde{t}_{u,b}^{(h)}(r), & \text{if } b>0, \\ c_u(r)f_u(\psi_u(r)), & \text{if } b=0, \end{cases} \tag{7}$$

where $\tilde{t}_{u,b}^{(h)}(r)$ is the estimate of $t_{u,b}^{(h)}(r)$, the delay of user u offloading task to MiBS b in round r and the h-th outcome happens (if $b>0$). When $b=0$, $\tilde{t}_{u,b}^{(h)}(r)$ is the accurate total time delay of local computing. To this end, user u should select the following MiBS to perform task offloading in round r

$$b_u(r) = \arg\min_{b\in\{\mathcal{B}_u,0\}} \tilde{t}_{u,b}(r). \tag{8}$$

Then, we have $\boldsymbol{z}_u(r) = [z_{u,1}(r),\ldots,z_{u,N_B}(r)] = \left[\mathbb{1}_{b_u(r)=1},\ldots,\mathbb{1}_{b_u(r)=N_B}\right]$ which is the result of Step ①, where the indicator $\mathbb{1}_\xi = 1$ when event ξ happens and $\mathbb{1}_\xi = 0$ otherwise. To clearly explain the notations here, we give an example in the following. Assume that in a UDN shown in Fig. 1, user $u=1$ decides to offload its task to MiBS $b=2$ in round $r=3$ according to (8). Then we have $b_1(3)=2$, $z_{1,2}(3)=1$, and $\boldsymbol{z}_1(3)=[0,0,1,0,0,0]$.

2) Offloading Decision Broadcast (Step ②): After Step ①, each user u uploads its task according to their decision $\boldsymbol{z}_u(r)$. The MaBS collects $\boldsymbol{Z}(r) = \{\boldsymbol{z}_u(r)\}_{u\in\mathcal{U}}$ via monitoring the network and broadcasts $\boldsymbol{Z}(r)$. This information will be received and analyzed by users for calibrated learning and $f(b,x)$ estimation in Steps ③ and ④, respectively. Note that in contrast to a centralized decision making scheme, the offloading decisions sharing happens after bandit selection in the proposed CCBL algorithm. Therefore, the task offloading decision broadcast has no impact on the task delay $t_{u,b}(r)$.

3) Calibrated Learning (Step ③): In this step, we develop a calibrated forecaster based on the calibrated learning, enabling each user u to learn $\boldsymbol{s}_u(r+1)$ based on $\boldsymbol{Z}(r)$, which will assist the user in offloading its task in round $r+1$. Calibrated learning was initially designed for an agent playing game with Nature [34]. If we consider a finite set of possible outcomes of Nature, the agent uses a forecaster to predict the outcomes through continuous observing. The sequence of forecasts is called calibrated if the prediction converges to the observed long-term probabilities of the outcomes [35]–[37].

In this UDN task offloading problem, each user u (agent) attempts to predict $\boldsymbol{h}_u(r+1)$ which reveals the number of tasks offloaded to all MiBSs $b\in\mathcal{B}$ (Nature) in round $r+1$. Recall that the finite set of task offloading outcomes is $\mathcal{H}$. We denote by $\mathcal{S}=\Delta(\mathcal{H})\subset\mathbb{R}^{N_H}$ the set of probability distributions over N_H outcomes from the view of users. The forecaster at user u aims to learn $\mathcal{S}$ and draws $\boldsymbol{s}_u(r+1)$ from $\mathcal{S}$ in Step ③. However, learning $\mathcal{S}$ will be very challenging if the number of its elements is not known *a priori*. Therefore, in this paper, we consider a relaxed problem: learning a finite set $\mathcal{S}=\{\boldsymbol{p}_s\}_{1\leqslant s\leqslant N_\epsilon}$ with N_ϵ elements. In the following, we first give the mathematical definition of an ϵ-calibrated forecaster, which attempts to learn $\mathcal{S}$ with a favorable performance.

Definition 1 ([38]): A forecaster at user u is referred to as ϵ-calibrated if for any $\epsilon>0$ and all strategies of other users, almost surely,

$$\limsup_{N_R\to\infty} \frac{1}{N_R}\left\|\sum_{r=1}^{N_R}[\boldsymbol{s}_u(r)-\boldsymbol{h}_u(r)]\right\| \leqslant \epsilon. \tag{9}$$

The definition means that the prediction result of an ϵ-calibrated forecaster $\boldsymbol{s}_u(r)$ converges to the observed long-term probabilities of the outcomes $\boldsymbol{h}_u(r)$ with high probability [39]. For simplicity, in this subsection, we replace $\boldsymbol{s}_u(r)$ with $\boldsymbol{s}(r)$ and $\boldsymbol{h}_u(r)$ with $\boldsymbol{h}(r)$ hereafter.

Theorem 1: In the UDN task offloading problem, there exists an ϵ–calibrated forecaster which generates $\boldsymbol{s}_u(r)$ from $\mathcal{S}$ for task in each round r.

Proof: We prove the theorem based on the approachability theory [40]. Consider a game between two players with finite action sets $\mathcal{I}$ and $\mathcal{J}$. We denote by $I(r)\in\mathcal{I}$ and $J(r)\in\mathcal{J}$ the actions taken by the players in round r, and $\boldsymbol{m}(I(r), J(r))\in\mathbb{R}^N$ the corresponding payoff. We assume that each player takes actions locally and has no knowledge of the other's strategy. Let $\mathcal{A}\subset\mathbb{R}^N$ be a set, then by definition, $\mathcal{A}$ is approachable if there exists a strategy for the first player such that for all strategies of the second one, almost surely,

$$\lim_{N_R\to\infty}\inf_{\boldsymbol{A}\in\mathcal{A}}\left\|\boldsymbol{A}-\frac{1}{N_R}\sum_{t=1}^{N_R}\boldsymbol{m}(I(r),J(r))\right\|=0. \tag{10}$$

This definition reveals that when a set is approachable, the first player can follow certain strategy and obtain a stable payoff in the long term, regardless of the second player's strategy.

In the UDN task offloading problem, we have $\mathcal{I}=\mathcal{S}$ and $\mathcal{J}=\mathcal{H}$. We define the payoff

$$\boldsymbol{m}(s,h) = [\boldsymbol{0}^{(N_H)},\ldots,\boldsymbol{0}^{(N_H)},\boldsymbol{p}_s-\boldsymbol{\zeta}_h,\boldsymbol{0}^{(N_H)},\ldots,\boldsymbol{0}^{(N_H)}] \in\mathbb{R}^{N_\epsilon\times N_H}, \tag{11}$$

where $\boldsymbol{0}^{(N_H)}=[0,\ldots,0]\in\mathbb{R}^{1\times N_H}$ and $\boldsymbol{\zeta}_h$ is the h-th vector in $\mathcal{H}$ ($1\leqslant h\leqslant N_H$). Thus, $N=N_\epsilon\times N_H$. We rewrite (10) as

$$\lim_{N_R\to\infty}\inf_{\boldsymbol{A}\in\mathcal{A}}\left\|\boldsymbol{A}-\frac{1}{N_R}\sum_{r=1}^{N_R}\boldsymbol{m}(S(r),H(r))\right\|=0, \tag{12}$$

where $S(r)$ indicates that $\boldsymbol{s}(r)=\boldsymbol{p}_{S(r)}$, and $H(r)$ indicates that $\boldsymbol{h}(r)=\boldsymbol{\zeta}_{H(r)}$. Note that $\boldsymbol{s}(r)$ is not used in the UDN task offloading problem as shown in Fig. 2, but its value does not affect the approachability theory when $N_R\to\infty$. Thus, we set $H(r)=1$. Furthermore, we define $\overline{\boldsymbol{m}}_{N_R}$ in (13), as shown at the bottom of the next page. We also define a closed convex set $\mathcal{A}$ as

$$\mathcal{A}\triangleq\left\{[a_1^1,\ldots,a_1^{N_H},\ldots,a_{N_\epsilon}^1,\ldots,a_{N_\epsilon}^{N_H}]:\sum_{s=1}^{N_\epsilon}\sum_{h=1}^{N_H}a_s^h\leqslant\epsilon\right\}. \tag{14}$$

If we substitute (13) and (14) into (12), then we get (9). That is, the existence of an ϵ-calibrated forecaster for the UDN task offloading problem is equivalent to the approachability of $\mathcal{A}$.

According to [40], a closed convex set $\mathcal{A}$ is approachable if and only if

$$\forall \boldsymbol{\zeta}_h \in \mathcal{H}, \exists \boldsymbol{p}_s \in \mathcal{S}, \quad \boldsymbol{m}(s,h) \in \mathcal{A}. \tag{15}$$

According to (11), we can construct $\mathcal{S}$ such that there exists $\boldsymbol{p}_s$, which satisfies $\| \boldsymbol{p}_s - \boldsymbol{\zeta}_h \| \leqslant \epsilon$, thus $\boldsymbol{m}(s,h) \in \mathcal{A}$ [41]. Therefore, the approachability of $\mathcal{A}$ is proved and we can ensure the existence of the ϵ-calibrated forecaster. □

After proving the existence of the ϵ-calibrated forecaster, now we introduce a method to calculate $\boldsymbol{s}(r+1)$, the prediction result of the ϵ-calibrated forecaster. As indicated in [40], user u can generate $\boldsymbol{s}(r+1)$ according to an optimal prediction distribution $\gamma^*(r) = [\gamma_1(r), \ldots, \gamma_{N_\epsilon}(r)]$ over $\mathcal{S}$, such that for any strategy of others users, there is

$$[\overline{\boldsymbol{m}}_{r-1} - \boldsymbol{\Pi}_{\mathcal{A}}(\overline{\boldsymbol{m}}_{r-1})] \cdot [\boldsymbol{m}(\gamma^*(r), h) - \boldsymbol{\Pi}_{\mathcal{A}}(\overline{\boldsymbol{m}}_{r-1})] \leqslant 0, \tag{16}$$

where $\boldsymbol{\Pi}_{\mathcal{A}}(\overline{\boldsymbol{m}}_{r-1})$ denotes the projection of $\overline{\boldsymbol{m}}_{r-1}$ onto $\mathcal{A}$. The existence of $\gamma^*(r)$ is proved in [40]. In the following, we first compute the projection $\boldsymbol{\Pi}_{\mathcal{A}}(\overline{\boldsymbol{m}}_{r-1})$, then we introduce how to obtain $\gamma^*(r)$.

The projection $\boldsymbol{\Pi}_{\mathcal{A}}(\overline{\boldsymbol{m}}_{r-1})$ refers to the closest point in $\mathcal{A}$ to $\overline{\boldsymbol{m}}_{r-1}$. Note that $\mathcal{A}$ is a convex set, the minimization problem can be solved as follows

$$\begin{aligned} \boldsymbol{\Pi}_{\mathcal{A}}(\overline{\boldsymbol{m}}_{r-1}) = \ & \arg\min_{\boldsymbol{A} \in \mathcal{A}} \|\boldsymbol{A} - \overline{\boldsymbol{m}}_{r-1}\|_2^2 \\ & \text{s.t.} \ \ a_s^h \geqslant 0, \ \ \sum_{s=1}^{N_\epsilon} \sum_{h=1}^{N_H} a_s^h \leqslant \epsilon, \end{aligned} \tag{17}$$

The projection onto $\mathcal{A}$ can be done to a desired precision δ_1 with complexity $\mathcal{O}(\log(1/\delta_1))$ [38]. Then, the optimal prediction distribution $\gamma^*(r)$ can be obtained through solving a mini-max problem via linear programming, which is given in (18), as shown at the bottom of the page. The computation leads to a polynomial complexity in N_H and N_ϵ. Alternatively, the problem can be solved approximately by utilizing the multiplicative weights algorithm [42]. Then, with a predefined small violation $\delta_2 > 0$ in each of N_H constraints given by (16), the complexity of such a solution reduces to $\mathcal{O}(N_H N_\epsilon \ln(N_\epsilon)/\delta_2^2)$ [38].

In **Algorithm 1**, we show how the ϵ-calibrated forecaster at user u generates $\boldsymbol{s}(t+1)$ from $\mathcal{S}$.

4) Computational Delay Function $f(b,x)$ Estimation (Step ④): With $\boldsymbol{Z}(r)$, each user u can update its learning result of $f(b,x)$. Specifically, when user u receives the returned task result, it can measure the computational delay $t_{u,b}^{com}(r)$. With the assumption that MiBSs allocate computing resources to all received tasks equally, user u can update its estimate of $f(b, \psi_u(r))$ as

$$\tilde{f}(b,\psi_u(r)) \leftarrow \frac{t_{u,b}^{com}(r)\kappa_{u,b}(r)/c_u + C(b,\psi_u(r))\tilde{f}(b,\psi_u(r))}{C(b,\psi_u(r)) + 1}. \tag{19}$$

Based on the aforementioned four steps in the CCBL algorithm, we summarize the algorithm in its entirety in **Algorithm 2**.

Algorithm 1 The Calibrated Learning Algorithm

1: **if** $t = 1$ **then**
2: Set $\gamma^*(r) = (1/N_\epsilon, \ldots, 1/N_\epsilon)$.
3: **end if**
4: Update $\overline{\boldsymbol{m}}_r$ following (13).
5: Calculate $\boldsymbol{\Pi}_{\mathcal{A}}(\overline{\boldsymbol{m}}_r)$ following (17).
6: Calculate $\gamma^*(r+1)$ following (18).
7: Select $\boldsymbol{s}(r+1)$ from $\mathcal{S}$ according to $\gamma^*(r+1)$.

Algorithm 2 The CCBL Algorithm for User u

1: Set $C(b,x) = 0$ for all task types. ▷ Initialization
2: **for** $r = 1, \ldots, N_R$ **do**
3: Make a choice between **Exploration** and **Exploitation**. ▷ Step ①
4: **if Exploration** is chosen **then**
5: Offload task to the MiBS $b_u(r)$ given in (6).
6: **else**
7: Calculate the estimated delay of all MiBSs following (7).
8: Offload task to the MiBS $b_u(r)$ given in (8).
9: **end if**
10: Upload $\boldsymbol{z}_u(r)$ to the MaBS, and then receive $\boldsymbol{Z}(r)$ from it. ▷ Step ②
11: Receive task result from MiBS $b_u(r)$ and measure the delay $t_{u,b}^{tra}(r)$ and $t_{u,b}^{com}(r)$.
12: Generate $\boldsymbol{s}_u(r+1)$ via **Algorithm 1**. ▷ Step ③
13: Update $\tilde{f}(b,x)$ following (19). ▷ Step ④
14: Update $r \leftarrow r+1$ and $C(b,\psi_u(r)) \leftarrow C(b,\psi_u(r)) + 1$.
15: **end for**

C. Computational Complexity and Convergence Rate

Based on the complexity of (17) and (18), the complexity of **Algorithm 2** is $\mathcal{O}(N_H N_\epsilon \ln(N_\epsilon)/\delta_2^2 + \log(1/\delta_1) + N_\epsilon)$. Note that Step ③ happens after Step ② (task offloading), thus the

$$\overline{\boldsymbol{m}}_{N_R} \triangleq \frac{1}{N_R}\left[\sum_{r=1}^{N_R} \mathbb{1}_{S(r)=1}(\boldsymbol{p}_1 - \boldsymbol{h}(r)), \ldots, \sum_{r=1}^{N_R} \mathbb{1}_{S(r)=N_\epsilon}(\boldsymbol{p}_{N_\epsilon} - \boldsymbol{h}(r))\right] = \frac{1}{N_R}\sum_{r=1}^{N_R} \boldsymbol{m}(S(r), H(r)). \tag{13}$$

$$\gamma^*(r) = \arg\min_{\gamma(r)} \max_{1 \leqslant h \leqslant N_H} [\overline{\boldsymbol{m}}_{r-1} - \boldsymbol{\Pi}_{\mathcal{A}}(\overline{\boldsymbol{m}}_{r-1})] \cdot \boldsymbol{m}(\gamma(r), h) = \arg\min_{\gamma(r)} \max_{1 \leqslant h \leqslant N_H} \sum_{s=1}^{N_\epsilon} \{\gamma_s(r)[\overline{\boldsymbol{m}}_{r-1} - \boldsymbol{\Pi}_{\mathcal{A}}(\overline{\boldsymbol{m}}_{r-1})] \cdot \boldsymbol{m}(s,h)\}. \tag{18}$$

computing of **Algorithm 1** at users happens simultaneously with the computing of tasks at MiBSs. Therefore, the computing time of **Algorithm 1** is not included in the task delay in (5). Due to the interaction between the contextual bandit learning and calibration learning in the CCBL algorithm, it becomes very challenging to provide the analysis on the convergence rate of the whole CCBL algorithm. Instead, we turn to analyze the convergence rate of the calibrated learning algorithm (**Algorithm 1**).

*Theorem 2: For the calibrated forecaster given in **Algorithm 1**, almost surely, we have*

$$\limsup_{N_R\to\infty}\frac{N_R^{1/N_H+1}}{\sqrt{\ln(N_R)}}\sup_{\mathcal{S}\in\mathfrak{S}}\left\|\frac{1}{N_R}\sum_{r=1}^{N_R}\mathbb{1}_{\boldsymbol{p}_{S(r)}\in\mathcal{S}}(\boldsymbol{p}_{S(r)}-\boldsymbol{\zeta}_{H(r)})\right\|\leqslant\Gamma_{N_H},\quad(20)$$

where $\mathfrak{S}$ is the Borel sigma-algebra of $\mathcal{S}$ and the constant Γ_{N_H} depends only on N_H.

Proof: We first introduce a meta-forecaster that proceeds in regimes [43]. In the ρ-th group ($\rho=1,2,\dots$) of a meta-forecaster, an ϵ_ρ-calibrated forecaster can be reached within J_ρ rounds with well-designed value of ϵ_ρ and J_ρ. That is, the resulting meta-forecaster is calibrated in the sense of (9), and even uniformly calibrated in the following sense [38], i.e., almost surely,

$$\lim_{N_R\to\infty}\sup_{\mathcal{S}\in\mathfrak{S}}\left\|\frac{1}{N_R}\sum_{r=1}^{N_R}\mathbb{1}_{\boldsymbol{p}_{S(r)}\in\mathcal{S}}(\boldsymbol{p}_{S(r)}-\boldsymbol{\zeta}_{H(r)})\right\|=0.\quad(21)$$

The uniform calibration implies any ϵ-calibration via the choices for $\mathfrak{S}$. An application of the Borel-Cantelli Lemma and Cesaro's Lemma shows that, when ϵ_ρ decreases towards 0 and J_ρ increases such that $\epsilon_\rho^{N_H}J_\rho$ tends to infinity fast enough, the uniformly calibrated forecaster shown in (21) can be reached. Recall that $J_\rho=2^\rho$, thus if we set $\epsilon_\rho=2^{-\rho/(N_H-1)}$, then J_ρ and $\sqrt{1/(\epsilon_\rho^{N_H-1}J_\rho)}$ are of the same order of magnitude. Therefore, we can achieve **Theorem 2**. □

Therefore, for the calibrated learning, the convergence rate decreases for an increasing number of task offloading outcomes N_H. However, for the regression process, if we assume that some fixed number of samples are required to estimate the reward of each joint action profile with sufficient precision, increasing N_U or N_H reduces the sampling rate and thereby the convergence speed. Therefore, adopting the optimal N_H that can guarantee a balanced trade-off between the convergence speed of the calibrated learning and regression process, is a challenge of the calibrated forecaster. This is out of the scope of this paper, but alternatively, we can choose a Nature that is easier to calibrate. In the next section, we propose a new calibrated forecaster and demonstrate by simulation its superiority over the CCBL in specific scenarios.

IV. User-Oriented Calibrated Forecaster

In the previous section, we have proposed the CCBL algorithm for decentralized task offloading, where the calibrated forecaster keeps observing the task offloading outcome $\boldsymbol{h}_u(r)$ and predicts its value in the next round $r+1$. The combination of all MiBSs are regarded as Nature, and therefore N_H being large (up to $(N_B)^{N_U-1}$) results in high computational complexity in **Algorithm 1**. To vastly reduce the computational complexity, in this section, we propose a novel user-oriented calibrated forecaster. The key is to transfer the calibration target from all MiBSs to a single user. In this case, each user $u'\in\mathcal{U}_{-u}$ is regarded as new Nature. Under this framework, each user now requires up to N_U-1 calibrated forecasters and each forecaster needs to calibrate up to N_B outcomes. Therefore, the total number of outcomes that require calibration is decreased to $N_B(N_U-1)$, which is linear in N_U.

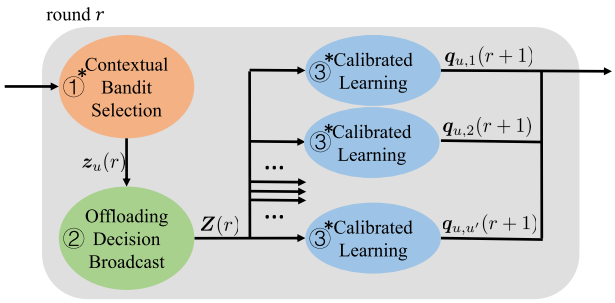


Fig. 4. Flowchart diagram of the UOCCBL scheme implemented at user u.

The flowchart of the UOCCBL algorithm is given in Fig. 4. For clear presentation, we omitted round $r-1$ and $r+1$ and step ④. Next, we elaborate on the new procedures of Step ① and Step ③, which are asterisked in the flowchart. It should be noted that compared with the CCBL algorithm, in the UOCCBL algorithm, the total number of outcomes that require calibration is largely decreased due to the transfer of the target of calibration. However, the decrease in computing complexity comes with a price. Later in the numerical results, we show that the long-term performance of the UOCCBL algorithm is inferior to that of the CCBL algorithm.

A. The UOCCBL Algorithm

1) Contextual Bandit Selection (Step ①): The calibrated learning result of user u in round $r-1$ is task offloading strategies of all other users $\boldsymbol{Q}_u(r)=\{\boldsymbol{q}_{u,u'}(r)\}_{u'\in\mathcal{U}_{-u}}$. From the perspective of user u, $\boldsymbol{q}_{u,u'}(r)=[q_{u,u'}^{(1)}(r),\dots,q_{u,u'}^{(N_B)}(r)]$ and $q_{u,u'}^{(b)}(r)$ is the estimated probability that user u' offloads its task to MiBS b in round r. Based on $\boldsymbol{Q}_u(r)$, user u can estimate the delay of offloading its task to MiBS b in round r as

$$\tilde{t}_{u,b}(r)=\sum_{\boldsymbol{w}\in\mathcal{W}}\Pr\{\boldsymbol{w}|\boldsymbol{Q}_u(r)\}\tilde{t}_{u,b}^{(\boldsymbol{w})}(r),\quad(22)$$

where logical matrix $\mathcal{W}\subset\mathbb{R}^{N_U\times N_B}$ includes all the possible task offloading decisions of N_U users, $\Pr\{\boldsymbol{w}|\boldsymbol{Q}_u(r)\}$ denotes the conditional probability of event $\boldsymbol{w}\in\mathcal{W}$ given $\boldsymbol{Q}_u(r)$. $\tilde{t}_{u,b}^{(\boldsymbol{w})}(r)$ is the estimate of $t_{u,b}(r)^{\boldsymbol{w}}$, the delay of user u offloading task to MiBS b in round r and event $\boldsymbol{w}$ happens.

2) Calibrated Learning (Step ③): In this step, $\boldsymbol{Z}(r)$ is analyzed by the calibrated forecaster at each user u to learn the task offloading strategies of all other users $\boldsymbol{Q}_u(r+1)$ for their

tasks in round $r+1$. The learning result enables user u to predict others' next decisions $\tilde{\boldsymbol{Z}}_{-u}(r+1) = \{\tilde{\boldsymbol{z}}_{u'}(r+1)\}_{u' \in \mathcal{U}_{-u}}$ (Note that this is the estimated value), and assists user u to offload its task for the next round staring from Step ①.

Recall that the finite set of offloading decisions of user $u' \in \mathcal{U}_{-u}$ is $\mathcal{B}$. We denote by $\mathcal{L} = \Delta(\mathcal{B}) \subset \mathbb{R}^{N_B}$ the set of probability distributions over N_B outcomes and $\mathcal{L} = \{\boldsymbol{y}_l\}_{1 \leqslant l \leqslant N_\epsilon}$ has N_ϵ elements. The forecaster at user u aims to learn $\mathcal{L}$ and draws $\boldsymbol{q}_{u,u'}(r+1)$ from $\mathcal{L}$ for all $u' \in \mathcal{U}_{-u}$ in Step ③. Similar to **Algorithm 1**, in each loop we first update $\overline{\boldsymbol{m}}'_r$ following (23), as shown at the bottom of the page, where $l(r')$ indicates that $\boldsymbol{y}(r') = \boldsymbol{p}_{l(r')}$, and $\boldsymbol{p}_l$ is the l-th vector in $\mathcal{L}$ ($1 \leqslant l \leqslant N_\epsilon$). Then calculate $\boldsymbol{\Pi}_{\mathcal{D}}(\overline{\boldsymbol{m}}'_r)$, the projection of $\overline{\boldsymbol{m}}'_r$ onto $\mathcal{D}$, following

$$\boldsymbol{\Pi}_{\mathcal{D}}\left(\overline{\boldsymbol{m}}'_{r-1}\right) = \arg\min_{\boldsymbol{D} \in \mathcal{D}} \left\|\boldsymbol{D} - \overline{\boldsymbol{m}}'_{r-1}\right\|_2^2. \tag{24}$$

where $\mathcal{D}$ is defined as

$$\begin{aligned}\mathcal{D} = \{&[d_1^1, \ldots, d_1^{N_B}, \ldots, d_{N_\epsilon}^1, \ldots, d_{N_\epsilon}^{N_B}]: \\ &d_l^b \geqslant 0, \quad \sum_{l=1}^{N_\epsilon} \sum_{b=1}^{N_B} d_l^b \leqslant \epsilon\}.\end{aligned} \tag{25}$$

At last, the calibration calculates $\gamma^*(r+1)$ follows,

$$\begin{aligned}&\gamma^*(r+1) \\ &= \arg\min_{\gamma(r+1)} \max_{b \in \mathcal{B}} \sum_{l=1}^{N_\epsilon} \{\gamma_s(r+1)\left[\overline{\boldsymbol{m}}'_r - \boldsymbol{\Pi}_{\mathcal{D}}\left(\overline{\boldsymbol{m}}'_r\right)\right] \\ &\quad \cdot \boldsymbol{m}'(l,b)\},\end{aligned} \tag{26}$$

where $\boldsymbol{m}'(l,b) = [\mathbf{0}^{(N_B)}, \ldots, \mathbf{0}^{(N_B)}, \boldsymbol{p}_l - \boldsymbol{\delta}_b, \mathbf{0}^{(N_B)}, \ldots, \mathbf{0}^{(N_B)}] \in \mathbb{R}^{N_\epsilon \times N_B}$ and $\mathbf{0}^{(N_B)} = [0, \ldots, 0] \in \mathbb{R}^{1 \times N_B}$. We denote $\boldsymbol{\delta}_b \in \mathbb{R}^{N_B}$ a vector whose b-th element is 1 and the others are 0. Then each forecaster selects $\boldsymbol{q}_{u,u'}(r+1)$ from $\mathcal{L}$ according to $\gamma^*(r+1)$.

The transfer of calibrated learning targets makes users no longer need to predict the task offloading decisions of all other users. Instead, each user only learns nearby peers' behavior, which significantly simplifies the algorithm implementation.

V. Simulation Results

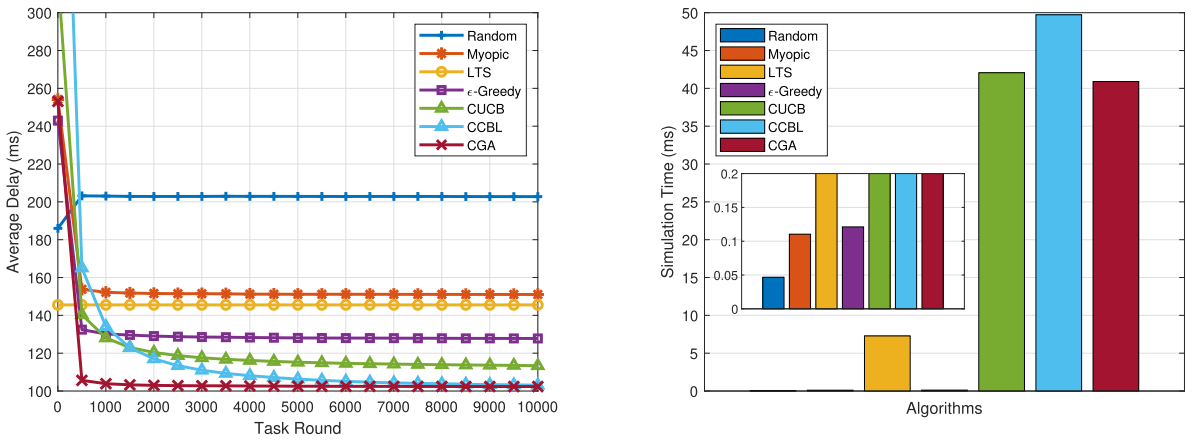
In this section, numerical results are presented to illustrate the advantages of the proposed CCBL and UOCCBL algorithms. We first illustrate that the performance of CCBL is superior to the existing decentralized task offloading strategies and only slightly inferior to the centralized one. Then, we validate the increased effectiveness of ϵ-calibration forecasters in the CCBL algorithm with increased r. Next, we validate that the proposed CCBL algorithm works robustly across a wide range of network typologies. Finally, we illustrate the relative merits of the UOCCBL and CCBL algorithms in different task rounds.

A. Conventional Algorithms for Comparison

To evaluate the performance of the proposed CCBL algorithm, we introduce several conventional algorithms for comparison.

- *Random*: In the random algorithm, all users randomly offload tasks to MiBSs. Clearly, this simplest method shall have the worst performance with low computational complexity $\mathcal{O}(1)$.
- *LTS*: We develop the baseline, the long-term stable (LTS) algorithm, where each user offloads tasks to a fixed MiBS and the task offloading strategy is optimized via an one-time exhaustive search with computational complexity $\mathcal{O}([N_X(N_B+1)]^{N_U})$. Note that the exhaustive search may take a long time as the task features are randomly generated.
- *Myopic [44]*: In the myopic algorithm, users explore all MiBS several times (we set 1 in our simulation) and stick to the one with the minimal expected delay after that. It does not utilize the knowledge of task features, thus its learning result can not accurately reveal the performance of MiBSs when task features vary. The computational complexity of each round is $\mathcal{O}(N_B)$.
- *ϵ-Greedy [45]*: In the ϵ-Greedy algorithm, users offload tasks to the MiBS with the minimal observed value of delay based on current knowledge but attempt to select other MiBS with a probability smaller than 0.7. Otherwise, it randomly offloads tasks. Unlike the myopic algorithm, the delay of MiBS processing tasks with different features is recorded separately. Then such knowledge can be utilized in improving the task processing delay prediction. The computational complexity of each round is also $\mathcal{O}(N_B)$ but its space complexity is N_X times larger than the myopic algorithm.
- *CUCB [43]*: In the Calibrated Upper Confidence Bound (CUCB) algorithm, all users obverse, learn, and predict peer task offloading decisions via calibrated forecasters. When they make scheduling decisions, they follow a UCB-based algorithm rather than the contextual bandit selection introduced in Section III-B-1). The computational complexity of each round is $\mathcal{O}(N_H N_\epsilon \ln(N_\epsilon)/\delta_2^2 + \log(1/\delta_1) + N_\epsilon)$.
- *CGA [46]*: In the centralized greedy algorithm (CGA), a central decision-maker collects task features of all users and makes task offloading decisions to minimize the total delay of all tasks. When tasks are completed, the value of delays is uploaded to the decision-maker which then learns the performance of all MiBSs in a centralized way. This algorithm achieves the minimum average delay with the cost of a central decision-maker deployment and information transmission overhead. The computational complexity of each round is $\mathcal{O}((N_B+1)^{N_U})$.

$$\overline{\boldsymbol{m}}'_r = \frac{1}{r}\left[\sum_{r'=1}^{r} \mathbb{1}_{l(r')=1}(\boldsymbol{p}_1 - \boldsymbol{z}(r')), \ldots, \sum_{r'=1}^{r} \mathbb{1}_{l(r')=N_\epsilon}(\boldsymbol{p}_{N_\epsilon} - \boldsymbol{z}(r'))\right]. \tag{23}$$



(a) (b)

Fig. 5. The (a) average delay of all tasks and (b) average simulation time of each task of all algorithms.

B. Simulation Results

The mobile edge computing network enabled by ultra-densely deployed computing services has been investigated in many publications, such as [14], [16], [47], [48]. In this paper, we mainly follow the widely adopted settings and configurations in these representative works. In specific, in the simulation, we set $\omega_0 = 15$ kHz for all $b \in \mathcal{B}$, $p_u = 0.5$ w, $c_u = 10^4$ bits for all $u \in \mathcal{U}$, and $\sigma_0^2 = 2 \times 10^{-13}$ w. We also set $N_X = 12$, $\eta = 2$, and $t_0 = 0.02$ s. The task feature $\psi_u(r)$ in round r at user u is picked from $\mathcal{X}$ randomly, the value of $f(b, x)$ for each b and x is picked from $[1, 12] \times 10^{-6}$ s/bit randomly, and $f_u(x)$ is picked from $[4, 48] \times 10^{-6}$ s/bit randomly. We set an equal computational resource allocation policy for all MiBSs to calculate $\kappa_{u,b}(r)$. That is $\kappa_{u,b}(r) = 1/\sum_{u' \in \mathcal{U}} z_{u',b}(r)$.

We compare the average delay of the proposed CCBL algorithm with the CGA and existing decentralized ones in Fig. 5(a). Here we consider a heterogeneous ultra-dense network with a typology the same as Fig. 1. As shown in Fig. 5(a), the LTS algorithm, an optimized one with fixed task offloading decisions, outperforms the random one. The average delay is also decreased compared with the random algorithm when the knowledge of MiBS computational delay function and the task features considered in the Myopic algorithm and the ϵ-Greedy algorithm, respectively. These two algorithms attempt to learn the computing performance of MiBSs but can not guide users to avoid collisions coming from peers. Therefore, when the actions of other users are predicted in the CUCB and CCBL algorithms, it is clearly shown that the average delay is further decreased as these two methods utilize all the available information in a decentralized manner. It is also shown that the proposed CCBL algorithm outperforms the UCB algorithm. This is because the contextual bandit selection (introduced in Section III-B-1) is a necessary condition for the convergence of calibrated learning [43]. If it is replaced with the CUCB algorithm, the convergence of the calibrated learning is not promised anymore. This will lead to an increase in the average delay of all tasks. We also note that the proposed CCBL algorithm is only slightly inferior to the centralized one (the performance benchmark). This significantly bridges the performance gap between the centralized and decentralized algorithms. We also record the average simulation time of each task on an AMD Ryzen Threadripper 3970X 32-Core CPU running at 3.69 GHz, Windows 10 (64 bit) with MATLAB. It shows that the simulation time of CUCB, CCBL, and CGA is much larger than that of the other algorithms. In return, the average delay of these three algorithms is the lowest. Note that the computation of the proposed CCBL algorithm mainly happens in Step ③ after Step ② (task offloading), thus the simulation time is not included in average task delay.

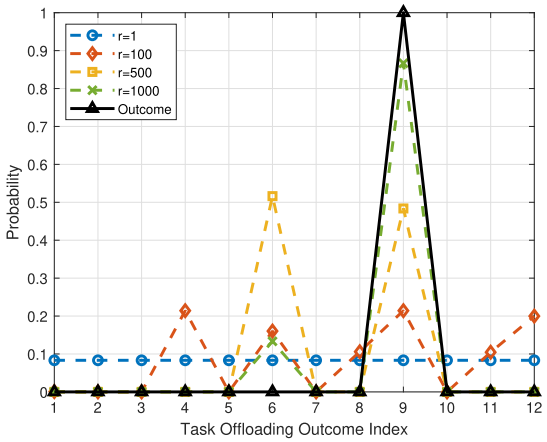


Fig. 6. The task offloading outcome $\boldsymbol{h}_u(r)$ and the estimated probabilities $\mathbf{s}_u(r)$ with increased task round r.

Fig. 6 depicts the task offloading outcome $\boldsymbol{h}_u(r)$ and its probability $\mathbf{s}_u(r)$ estimated by user $u = 1$ for different task round r. As shown in the figure, when $r = 1$, the probability of all $N_H = 12$ outcomes is $1/12$. Thus, user 1 randomly picks an outcome and takes it as the prediction result. With the increase of r, user 1 observes more task offloading decisions of others and utilizes them for the calibrated learning. As a result, the prediction results keep approaching the true outcome and almost converge to it when $r = 1000$. This validates the increased accuracy of prediction of N_ϵ calibration learning with increased r (Step ③ in Fig. 2).

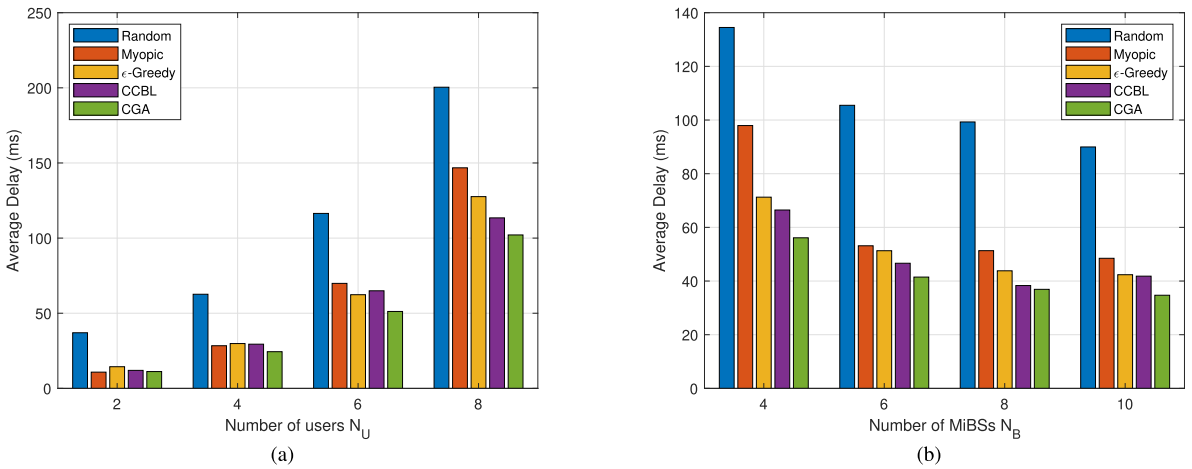


Fig. 7. The average delay of all users versus the number of (a) users and (b) MiBSs with $N_R = 10^4$.

Fig. 7 illustrates how the proposed CCBL algorithm works robustly when the number of users N_U and MiBSs N_B vary. For comparison, we also depict the performance of two benchmarks (Random and CGA) and two typical decentralized algorithms (Myopic and ϵ-Greedy). In Fig. 7(a), it is observed that the average delay of the decentralized algorithms are very close to the CGA when N_U is small ($N_U = 2$). This is because users' tasks are rarely offloaded to a certain MiBS when N_U is small, so the advantage of centralized scheduling is not apparent in these cases. With the increase of N_U, we can see that the average delay of all algorithms increases, as more tasks are required to be processed with the same number of MiBSs. Meanwhile, it is shown in the figure that the gap between the average delay of decentralized algorithms and the centralized one becomes more considerable. This is predictable as task collision becomes more common in decentralized algorithms, where some MiBSs are required to process tasks offloaded by multiple users. The superiority of deploying a central decision-maker becomes obvious when task collision happens. However, we notice in the figure that the average delay of the proposed CCBL algorithm is almost the same as the CGA with small N_U and close to CGA with large N_U. This reveals the robustness of the CCBL algorithm. In Fig. 7(b), with the increase of N_B, the average delay of all algorithms decreases. Meanwhile, the gap between the centralized and decentralized algorithms becomes smaller. This shows that adequate computational resources (MiBSs) lead to small task delays and weaken the advantages of well-designed algorithms.

Fig. 8 illustrates the impact of N_ϵ on the average delay and computing complexity (simulation time). With the increased N_ϵ, the calibrated forecaster can take more possible situations into consideration with the cost of higher computing complexity. We adopt the same setting as in Fig. 5(a) and record the simulation time. It is obvious that a larger N_U leads to higher computing complexity and larger average delay. As shown in the figure, the simulation time increases almost linearly with N_ϵ for both $N_U = 7$ and $N_U = 8$. For $N_U = 7$, when N_ϵ increases, the average delay decreases significantly for $N_\epsilon \leqslant 10$. This is because $\mathcal{S}$ can hardly cover the probability of

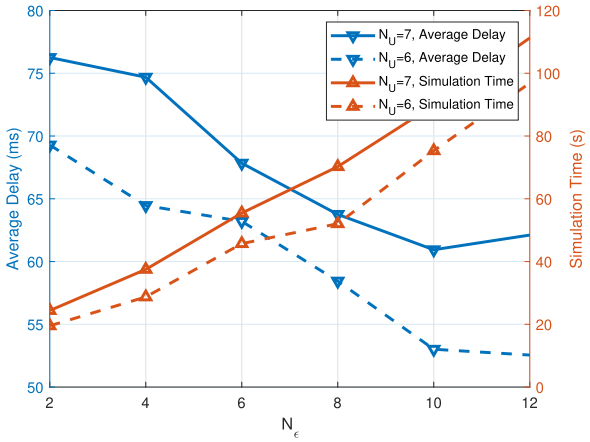


Fig. 8. The average delay of all users and the simulation time versus N_ϵ with different N_U.

N_H outcomes when N_ϵ is small, thus the increase of N_ϵ can obviously improve the prediction accuracy. However, when $N_\epsilon \geqslant 10$, the prediction accuracy will no longer benefit from the increase of N_ϵ. Thus the average delay of tasks fluctuates with the increase of N_ϵ. For $N_U = 8$, a similar trend could be observed. The simulation result reveals the necessity of choosing suitable N_ϵ for task offloading with different network typologies.

Fig. 9 illustrates the relative merits of the UOCCBL and CCBL algorithms for different N_U. As shown in the figure, the average delay of UOCCBL decreases faster than that of CCBL when $r \leqslant 2 \times 10^3$ for all N_U. This is because N_H is smaller in UOCCBL than CCBL, as mentioned in Section IV, and this will accelerate the convergence rate. It is also shown that when $N_U = 4$, the average delay of these two algorithms is almost the same when r is large. This is because when N_U is small the conflicts among users happen less frequently, thus the task offloading decisions of users can be regarded as independent. Therefore, the performance of the UOCCBL algorithm will approach that of the CCBL when r increases.

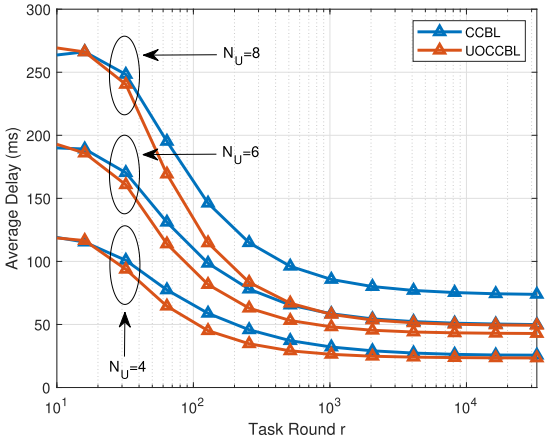


Fig. 9. The average delay of all users versus task round r using CCBL and UOCCBL algorithms with different N_U.

VI. Discussion and Extensions

A. Extensions of the Proposed Algorithm in Dynamic Networks

The proposed method only works in static scenarios, and its effectiveness might not be guaranteed whenever the network topology changes. This is because the learning results of the proposed algorithm depend on three aspects: the task features, the unknown BS computing functions, and the topology of the network. If a mobile user enters a new network, all three aspects are different. Therefore, the historic learning results are not effective anymore and a new learning is necessary. However, if mobile users move within a specific network (the coverage area of a specific MaBS), a new learning is not necessary. In the following, we will discuss two possible methods to handle the mobility by enhancing the proposed CCBL algorithm.

- We can incorporate the passively adaptive methods (e.g., SW-UCB algorithm [49] and Rexp3 algorithm [50]) into our algorithm. Thereby we can keep track of other users' behaviors by updating users' learning results based on the most recent observations.
- Besides, we can also leverage actively adaptive methods (e.g., Adapt-EvE algorithm [51] and CD-UCB algorithm [52]). They adopt extra changing detection algorithms to monitor the varying topology caused by the mobility of users. The CCBL algorithm will restart when the variation of the topology exceeds a given threshold.

B. Possibility That the MaBS Provides Computing Capability

In this paper, the function of the MaBS is not to provide strong computational capabilities but to only broadcast the task offloading decisions of all users. The MiBSs are ultra-densely deployed, thus they are normally much closer to users than the MaBS with a large coverage. Offloading tasks to MiBSs can significantly reduce the transmission latency and energy. Note that the broadcasting happens in Step ② of our algorithm. The MaBS provides users with the task offloading decisions of others and enables users to learn the behavior of others.

We consider a UDN in this paper, where the MiBSs are densely deployed. Therefore, it is rare that a user is not covered by any MiBS. However, if such a case happens, it is possible to extend our proposed policy. In specific, the MaBS can be taken as a MiBS that covers all users. It can receive tasks from users and monitor the system simultaneously, and broadcast the task offloading decisions of all users when doing task processing. At last, it returns the task results. However, as mentioned above, offloading tasks to the MaBS will result in large transmission delay, thus it will not be a preferred choice for users.

C. Experimental Results

Some actual experimental results and comparisons with them can further verify the merits of the proposed method. However, these require abundant hardware resources such as a wide-band access point with full-duplex communication capabilities, multiple access points with computational capabilities, and lots of end devices capable of communication in multiple wireless channels. The verification and comparison of a testbed construction and associated experimental could serve as a future topic.

D. Malicious Users and Imperfect Information Broadcast

In the proposed system, if malicious users exist and upload tasks to several MiBSs at the same time, they will occupy lots of computational resources and increase the average delay of all users. In this case, the MaBS can observe the unusual task offloading pattern and identify the related users as malicious ones. Then the MaBS can instruct MiBSs to stop processing the tasks from the malicious users. In this paper, we assume perfect and accurate task offloading decisions broadcast by the MaBS. In the case that the users far away from the MaBS fail to receive the broadcast information, it will make offloading decisions based on the previous learning results, and this will serve as our future research topic.

VII. Conclusion

In this paper, we developed a novel decentralized task offloading strategy in UDN, where users make task offloading decisions locally and independently. We formulated the associated optimization problem by minimizing the long-term average task delay among all users and proposed a CCBL algorithm. This algorithm enables users to learn the computational delay functions of MiBSs and predict the task offloading decisions of other users in a decentralized manner. Based on the approachability theory, we verified the convergence of the CCBL algorithm. Then we propose a UOCCBL algorithm to further decrease the computational complexity and increase convergence rate, where the target of calibrated learning is transferred from all MEC servers to a single user. Simulation results validated that the proposed CCBL algorithm significantly outperforms the existing algorithms in terms of the long-term average delay and that it is only slightly inferior to the centralized one.

References

[1] I. Bortone *et al.*, "Wearable haptics and immersive virtual reality rehabilitation training in children with neuromotor impairments," *IEEE Trans. Neural Syst. Rehabil. Eng.*, vol. 26, no. 7, pp. 1469–1478, Jul. 2018.

[2] F. Wu, T. Wu, and M. Yuce, "An Internet-of-Things (IoT) network system for connected safety and health monitoring applications," *Sensors*, vol. 19, no. 1, p. 21, Dec. 2018.

[3] S. Liu, L. Liu, J. Tang, B. Yu, and W. Shi, "Edge computing for autonomous driving: Opportunities and challenges," *Proc. IEEE*, vol. 107, no. 8, pp. 1697–1716, Jun. 2019.

[4] Y. Mao, C. You, J. Zhang, K. Huang, and K. B. Letaief, "A survey on mobile edge computing: The communication perspective," *IEEE Commun. Surveys Tuts.*, vol. 19, no. 4, pp. 2322–2358, 4th Quart., 2017.

[5] N. Abbas, Y. Zhang, A. Taherkordi, and T. Skeie, "Mobile edge computing: A survey," *IEEE Internet Things J.*, vol. 5, no. 1, pp. 450–465, Feb. 2018.

[6] T. X. Tran, A. Hajisami, P. Pandey, and D. Pompili, "Collaborative mobile edge computing in 5G networks: New paradigms, scenarios, and challenges," *IEEE Commun. Mag.*, vol. 55, no. 4, pp. 54–61, Apr. 2017.

[7] M. Jia and W. Liang, "Delay-sensitive multiplayer augmented reality game planning in mobile edge computing," in *Proc. 21st ACM Int. Conf. Modeling, Anal. Simulation Wireless Mobile Syst.* New York, NY, USA: ACM, 2018, pp. 147–154.

[8] Y. Siriwardhana, P. Porambage, M. Liyanage, and M. Ylianttila, "A survey on mobile augmented reality with 5G mobile edge computing: Architectures, applications, and technical aspects," *IEEE Commun. Surveys Tuts.*, vol. 23, no. 2, pp. 1160–1192, 2nd Quart., 2021.

[9] Y. Yu, "Mobile edge computing towards 5G: Vision, recent progress, and open challenges," *China Commun.*, vol. 13, no. 2, pp. 89–99, 2016.

[10] C. You, K. Huang, H. Chae, and B.-H. Kim, "Energy-efficient resource allocation for mobile-edge computation offloading," *IEEE Trans. Wireless Commun.*, vol. 16, no. 3, pp. 1397–1411, Mar. 2017.

[11] L. Huang, X. Feng, C. Zhang, L. Qian, and Y. Wu, "Deep reinforcement learning-based joint task offloading and bandwidth allocation for multi-user mobile edge computing," *Digit. Commun. Netw.*, vol. 5, no. 1, pp. 10–17, Feb. 2019.

[12] T. X. Tran and D. Pompili, "Joint task offloading and resource allocation for multi-server mobile-edge computing networks," *IEEE Trans. Veh. Technol.*, vol. 68, no. 1, pp. 856–868, Jan. 2019.

[13] C. Gao, Y. Li, Y. Zhao, and S. Chen, "A two-level game theory approach for joint relay selection and resource allocation in network coding assisted D2D communications," *IEEE Trans. Mobile Comput.*, vol. 16, no. 10, pp. 2697–2711, Oct. 2017.

[14] M. Kamel, W. Hamouda, and A. Youssef, "Ultra-dense networks: A survey," *IEEE Commun. Surveys Tuts.*, vol. 18, no. 4, pp. 2522–2545, 4th Quart., 2016.

[15] S. Chen, F. Qin, B. Hu, X. Li, and Z. Chen, "User-centric ultra-dense networks for 5G: Challenges, methodologies, and directions," *IEEE Wireless Commun.*, vol. 23, no. 2, pp. 78–85, Apr. 2016.

[16] Y. Sun, S. Zhou, and J. Xu, "EMM: Energy-aware mobility management for mobile edge computing in ultra dense networks," *IEEE J. Sel. Areas Commun.*, vol. 35, no. 11, pp. 2637–2646, Nov. 2017.

[17] K. Liu and Q. Zhao, "Distributed learning in multi-armed bandit with multiple players," *IEEE Trans. Signal Process.*, vol. 58, no. 11, pp. 5667–5681, Nov. 2010.

[18] V. Anantharam, P. Varaiya, and J. Walrand, "Asymptotically efficient allocation rules for the multiarmed bandit problem with multiple plays—Part II: Markovian rewards," *IEEE Trans. Autom. Control*, vol. AC-32, no. 11, pp. 977–982, Nov. 1987.

[19] Z. Chen and X. Wang, "Decentralized computation offloading for multi-user mobile edge computing: A deep reinforcement learning approach," *EURASIP J. Wireless Commun. Netw.*, vol. 2020, no. 1, pp. 1–21, Dec. 2020.

[20] S. Seng, C. Luo, X. Li, H. Zhang, and H. Ji, "User matching on blockchain for computation offloading in ultra-dense wireless networks," *IEEE Trans. Netw. Sci. Eng.*, vol. 8, no. 2, pp. 1167–1177, Apr. 2021.

[21] J. Xu, L. Chen, and P. Zhou, "Joint service caching and task offloading for mobile edge computing in dense networks," in *Proc. IEEE Conf. Comput. Commun.*, Apr. 2018, pp. 207–215.

[22] K. Gai, K. Xu, Z. Lu, M. Qiu, and L. Zhu, "Fusion of cognitive wireless networks and edge computing," *IEEE Wireless Commun.*, vol. 26, no. 3, pp. 69–75, Jun. 2019.

[23] J. Ren, G. Yu, Y. He, and G. Y. Li, "Collaborative cloud and edge computing for latency minimization," *IEEE Trans. Veh. Technol.*, vol. 68, no. 5, pp. 5031–5044, May 2019.

[24] Z. Lin, J. Li, Y. Zheng, N. V. Irukulapati, H. Wang, and H. Sahlin, "SS/PBCH block design in 5G new radio (NR)," in *Proc. IEEE Globecom Workshops (GC Wkshps)*, Dec. 2018, pp. 1–6.

[25] B. E. Priyanto, H. Codina, S. Rene, T. B. Sorensen, and P. Mogensen, "Initial performance evaluation of DFT-spread OFDM based SC-FDMA for UTRA LTE uplink," in *Proc. IEEE 65th Veh. Technol. Conf. (VTC-Spring)*, Apr. 2007, pp. 3175–3179.

[26] X. Jin, "Channel estimation techniques of SC-FDMA," Ph.D. dissertation, Dept. Phys. Elect. Eng., Karlstad Univ., Karlstad, Sweden, 2007.

[27] X. Chen, L. Jiao, W. Li, and X. Fu, "Efficient multi-user computation offloading for mobile-edge cloud computing," *IEEE/ACM Trans. Netw.*, vol. 24, no. 5, pp. 2795–2808, Oct. 2016.

[28] *Evolved Universal Terrestrial Radio Access*, 3GPP, document TR 36.931, May 2011.

[29] P. Hu, H. Ning, T. Qiu, Y. Zhang, and X. Lou, "Fog computing based face identification and resolution scheme in Internet of Things," *IEEE Trans. Ind. Informat.*, vol. 13, no. 4, pp. 1910–1920, Aug. 2017.

[30] Y. Wu, F. Li, L. Ma, Y. Xie, T. Li, and Y. Wang, "A context-aware multiarmed bandit incentive mechanism for mobile crowd sensing systems," *IEEE Internet Things J.*, vol. 6, no. 5, pp. 7648–7658, Oct. 2019.

[31] Y.-L. Jiang, Y.-S. Chen, S.-W. Yang, and C.-H. Wu, "Energy-efficient task offloading for time-sensitive applications in fog computing," *IEEE Syst. J.*, vol. 13, no. 3, pp. 2930–2941, Sep. 2019.

[32] Y. Wang, M. Sheng, X. Wang, L. Wang, and J. Li, "Mobile-edge computing: Partial computation offloading using dynamic voltage scaling," *IEEE Trans. Commun.*, vol. 64, no. 10, pp. 4268–4282, Oct. 2016.

[33] T. L. Lai and H. Robbins, "Asymptotically efficient adaptive allocation rules," *Adv. Appl. Math.*, vol. 6, no. 1, pp. 4–22, Mar. 1985.

[34] D. P. Foster and R. V. Vohra, "Calibrated learning and correlated equilibrium," *Games Econ. Behav.*, vol. 21, nos. 1–2, p. 40, 1997.

[35] S. Maghsudi and S. Stańczak, "Channel selection for network-assisted D2D communication via no-regret bandit learning with calibrated forecasting," *IEEE Trans. Wireless Commun.*, vol. 14, no. 3, pp. 1309–1322, Mar. 2015.

[36] X. Zhang, M. R. Nakhai, G. Zheng, S. Lambotharan, and B. Ottersten, "Calibrated learning for online distributed power allocation in small-cell networks," *IEEE Trans. Commun.*, vol. 67, no. 11, pp. 8124–8136, Nov. 2019.

[37] X. Xu and M. Tao, "Decentralized multi-agent multi-armed bandit learning with calibration for multi-cell caching," *IEEE Trans. Commun.*, vol. 69, no. 4, pp. 2457–2472, Apr. 2021.

[38] S. Mannor and G. Stoltz, "A geometric proof of calibration," *Math. Oper. Res.*, vol. 35, no. 4, pp. 721–727, Nov. 2010.

[39] P. Auer, N. Cesa-Bianchi, Y. Freund, and R. E. Schapire, "The nonstochastic multiarmed bandit problem," *SIAM J. Comput.*, vol. 32, no. 1, pp. 48–77, 2011.

[40] D. Blackwell, "An analog of the minimax theorem for vector payoffs," *Pacific J. Math.*, vol. 6, no. 1, pp. 1–8, 1956.

[41] D. P. Foster, "A proof of calibration via Blackwell's approachability theorem," *Games Econ. Behav.*, vol. 29, nos. 1–2, pp. 73–78, Oct. 1999.

[42] Y. Freund and R. E. Schapire, "Adaptive game playing using multiplicative weights," *Games Econ. Behav.*, vol. 29, nos. 1–2, pp. 79–103, 1999.

[43] N. Cesa-Bianchi and G. Lugosi, *Prediction, Learning, and Games*. Cambridge, U.K.: Cambridge Univ. Press, 2006.

[44] F. Wang, J. Xu, and S. Cui, "Optimal energy allocation and task offloading policy for wireless powered mobile edge computing systems," *IEEE Trans. Wireless Commun.*, vol. 19, no. 4, pp. 2443–2459, Apr. 2020.

[45] V. Kuleshov and D. Precup, "Algorithms for multi-armed bandit problems," 2014, *arXiv:1402.6028*.

[46] W.-J. Feng, C.-H. Yang, and X.-S. Zhou, "Multi-user and multi-task offloading decision algorithms based on imbalanced edge cloud," *IEEE Access*, vol. 7, pp. 95970–95977, 2019.

[47] M. Chen and Y. Hao, "Task offloading for mobile edge computing in software defined ultra-dense network," *IEEE J. Sel. Areas Commun.*, vol. 36, no. 3, pp. 587–597, Mar. 2018.

[48] H. Guo, J. Liu, and J. Zhang, "Computation offloading for multi-access mobile edge computing in ultra-dense networks," *IEEE Commun. Mag.*, vol. 56, no. 8, pp. 14–19, Aug. 2018.

[49] A. Garivier and E. Moulines, "On upper-confidence bound policies for switching bandit problems," in *Algorithmic Learning Theory*, J. Kivinen, C. Szepesvári, E. Ukkonen, and T. Zeugmann, Eds. Berlin, Germany: Springer, 2011, pp. 174–188.

[50] O. Besbes, Y. Gur, and A. Zeevi, "Stochastic multi-armed-bandit problem with non-stationary rewards," in *Proc. Adv. Neural Inf. Process. Syst.*, vol. 27, 2014, pp. 199–207.

[51] C. Hartland, N. Baskiotis, S. Gelly, M. Sebag, and O. Teytaud, "Change point detection and meta-bandits for online learning in dynamic environments," in *Proc. Conférence Francophone Sur L'Apprentissage Automatique*, 2007, pp. 237–250.

[52] F. Liu, J. Lee, and N. Shroff, "A change-detection based framework for piecewise-stationary multi-armed bandit problem," in *Proc. AAAI Conf. Artif. Intell.*, 2018, vol. 32, no. 1, pp. 1–8.

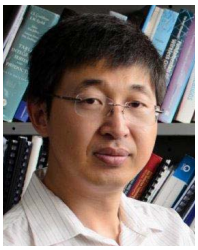
Rui Zhang received the Ph.D. degree from The University of Sydney in December 2020. He is currently with the Department of Information Engineering, The Chinese University of Hong Kong. His research interests include machine learning in wireless communications and wireless sensing and localization systems.

Peng Cheng (Member, IEEE) received the B.S. and M.S. degrees (Hons.) in communication and information systems from the University of Electronic Science and Technology of China (UESTC), Chengdu, China, in 2006 and 2009, respectively, and the Ph.D. degree from Shanghai Jiao Tong University, Shanghai, China, in 2013. From 2014 to 2017, he was a Post-Doctoral Research Scientist at CSIRO, Sydney, Australia. From 2017 to 2020, he was an ARC DECRA Fellow/Lecturer at The University of Sydney, Sydney. He is currently an ARC DECRA Fellow and a Senior Lecturer (Tenured Associate Professor in U.S. systems) with the Department of Computer Science and Information Technology, La Trobe University, Australia, and also affiliated with The University of Sydney. He has published over 70 peer-reviewed research papers in leading international journals and conferences. His current research interests include wireless AI, machine learning, the IoT, millimeter-wave communications, and compressive sensing theory.

Zhuo Chen (Senior Member, IEEE) received the B.S. degree in electrical engineering from Shanghai Jiao Tong University, Shanghai, China, in 1997, and the M.S. and Ph.D. degrees from the School of Electrical and Information Engineering, The University of Sydney, Sydney, Australia, in 2001 and 2004, respectively. He was a Senior Research Scientist with the Commonwealth Scientific and Industrial Research Organization (CSIRO), Sydney. His research interests include wireless communications, machine learning, and wireless sensor networks.

Sige Liu received the B.S. degree in communication engineering from the University of Electronic Science and Technology of China, Chengdu, China, in 2018. He is currently pursuing the Ph.D. degree with the School of Electrical and Information Engineering, The University of Sydney, Australia. His research interests include machine learning and its applications in wireless networks, the Internet of Things (IoTs), and mobile edge computing.

Branka Vucetic (Life Fellow, IEEE) is currently an ARC Laureate Fellow and the Director of the Centre of Excellence for IoT and Telecommunications, The University of Sydney. Her current research work is in wireless networks and the Internet of Things. In the area of wireless networks, she works on communication system design for millimeter wave frequency bands. In the area of the Internet of Things, she works on providing wireless connectivity for mission critical applications. She is a fellow of the Australian Academy of Technological Sciences and Engineering and the Australian Academy of Science.



Yonghui Li (Fellow, IEEE) received the Ph.D. degree in communications engineering from Beihang University, Beijing, China, in November 2002.

He is now a Professor and the Director of Wireless Engineering Laboratory, The University of Sydney, Sydney, Australia. He holds a number of patents granted and pending. His current research interests include wireless communications, with a particular focus on MIMO, millimeter wave communications, machine to machine communications, coding techniques, and cooperative communications.

Prof. Li was a recipient of the Australian Queen Elizabeth II Fellowship in 2008 and the Australian Future Fellowship in 2012. He was also a recipient of several best paper awards from IEEE conferences, such as International Conference on Communications (ICC) and International Symposium on Personal, Indoor, and Mobile Radio Communications (PIRMC). He is now an Editor of IEEE TRANSACTIONS ON COMMUNICATIONS and IEEE TRANSACTIONS ON VEHICULAR TECHNOLOGY. He also served as the Guest Editor for several IEEE journals.