Latency Optimization for Blockchain-Empowered Federated Learning in Multi-Server Edge Computing

Dinh C. Nguyen, *Member, IEEE*, Seyyedali Hosseinalipour, *Member, IEEE*, David J. Love, *Fellow, IEEE*, Pubudu N. Pathirana, *Senior Member, IEEE*, and Christopher G. Brinton, *Senior Member, IEEE*

***Abstract*— In this paper, we study a new latency optimization problem for blockchain-based federated learning (BFL) in multi-server edge computing. In this system model, distributed mobile devices (MDs) communicate with a set of edge servers (ESs) to handle both machine learning (ML) model training and block mining simultaneously. To assist the ML model training for resource-constrained MDs, we develop an offloading strategy that enables MDs to transmit their data to one of the associated ESs. We then propose a new decentralized ML model aggregation solution at the edge layer based on a consensus mechanism to build a global ML model via peer-to-peer (P2P)-based blockchain communications. Blockchain builds trust among MDs and ESs to facilitate reliable ML model sharing and cooperative consensus formation, and enables rapid elimination of manipulated models caused by poisoning attacks. We formulate latency-aware BFL as an optimization aiming to minimize the system latency via joint consideration of the data offloading decisions, MDs' transmit power, channel bandwidth allocation for MDs' data offloading, MDs' computational allocation, and hash power allocation. Given the mixed action space of discrete offloading and continuous allocation variables, we propose a novel deep reinforcement learning scheme with a parameterized advantage actor critic algorithm. We theoretically characterize the convergence properties of BFL in terms of the aggregation delay, mini-batch size, and number of P2P communication rounds. Our numerical evaluation demonstrates the superiority of our proposed scheme over baselines in terms of model training efficiency, convergence rate, system latency, and robustness against model poisoning attacks.**

***Index Terms*— Federated learning, blockchain, edge computing, actor-critic learning, network optimization.**

I. Introduction

In recent years, the demand for deploying machine learning (ML) in wireless networks and Internet of Things (IoT) applications has increased dramatically. However, due to growing concerns associated with data privacy, it is infeasible to transmit all collected data from IoT edge devices to a central location (e.g., a datacenter) for model training. Federated learning (FL) has emerged as a popular approach for distributed ML which allows for model training without requiring data sharing [1], [2]. Under FL, devices operate as workers to train local ML models using their own datasets, and exchange their model updates with an aggregator, e.g., an edge server (ES), over multiple communication rounds to converge on a global model. While FL distributes the data processing step across devices, the model aggregation step is still often carried out at a single location, which imposes security issues including single-point-of-failure and server malfunction. Additionally, this poses scalability restrictions for ML training processes, especially as the number of IoT devices involved and their geographic reach continue to expand [3]. Therefore, it is desirable to develop a more decentralized FL architecture for realizing scalable model training while preserving security in next-generation intelligent networks.

In this context, blockchain is a promising technology for enabling reliable decentralized FL via its peer-to-peer (P2P) networking topology empowered by an immutable, transparent and tamper-proof data ledger [4]. The use of blockchain in FL can mitigate the single-point-of-failure issue, and builds trust between devices and multiple servers for secure ML model training [5]. Given these benefits, blockchain-based federated learning (BFL) has been recently investigated in different domains, such as vehicular communications [6] and mobile crowdsensing [7]. To implement BFL systems, devices need to interface with decentralized servers in settings where communication and computation resources are limited, which will impact the ML model training quality. Moreover, each device exhibits interest in participating in block mining to further gain blockchain rewards, e.g., cryptocurrency tokens, which in turn enhances the reliability and security for FL. This leads to the concept of *mobile mining* which has been adopted in practical BFL environments [8], [9]. The concurrence of both ML model training and block mining introduces new challenges to network service latency and resource management, motivating a holistic optimization architecture for efficient BFL in wireless networks.

A. Related Works

We summarize related works in latency optimization and resource allocation for standard FL and decentralized BFL.

1) Standard FL: Latency optimization has recently received significant attention in FL research. The work in [10] proposed a joint device scheduling and bandwidth allocation framework for wireless FL to improve the convergence rate of ML model

Manuscript received 16 March 2022; revised 17 June 2022; accepted 30 June 2022. Date of publication 12 October 2022; date of current version 22 November 2022. This work was supported under Grant ONR N00014-22-1-2305, Grant ONR N00014-21-1-2472, and Grant NSF CNS-2146171. *(Corresponding author: Dinh C. Nguyen.)*

Dinh C. Nguyen, David J. Love, and Christopher G. Brinton are with the Elmore Family School of Electrical and Computer Engineering, Purdue University, West Lafayette, IN 47907 USA (e-mail: nguye772@purdue.edu; hosseina@purdue.edu; djlove@purdue.edu; cgb@purdue.edu).

Seyyedali Hosseinalipour is with the Department of Electrical Engineering, University at Buffalo (SUNY), Buffalo, NY 14228 USA (e-mail: alipour@buffalo.edu).

Pubudu N. Pathirana is with the School of Engineering, Deakin University, Waurn Ponds, VIC 3216, Australia (e-mail: pubudu.pathirana@deakin.edu.au).

Color versions of one or more figures in this article are available at https://doi.org/10.1109/JSAC.2022.3213344.

Digital Object Identifier 10.1109/JSAC.2022.3213344

training. Another study in [11] investigated semi-asynchronous FL, focusing on the convergence analysis of model training under edge heterogeneity and non-independent and identically distributed (non-IID) data distributions across the edge devices. An asynchronous FL scheme was considered in [12] for unmanned aerial vehicles (UAVs)-assisted networks to minimize the model exchange latency and ML training loss via deep reinforcement learning (DRL). To mitigate straggler effects caused by resource-limited clients, the authors in [13] presented a partial offloading-assisted FL scheme using game theory. A partial offloading-based FL solution was also proposed in [14] for edge computing, focusing on the delay analysis of the data offloading and model update. Similarly, a convex optimization approach was applied in [15] to minimize the energy consumption of model updating and sharing.

2) Decentralized and Blockchain-Based FL: Several recent works have considered techniques for decentralizing FL aggregation schemes. The authors in [16] introduced a decentralized FL scheme based on model segmentation with a gossip protocol for client sampling in each model aggregation round. In [17], a decentralized FL solution was proposed using the device-to-device (D2D) concept in serverless edge networks, with a graph-coloring based scheduling policy to characterize the data communications between two devices. Our recent works in [18], [19] developed D2D-based semi-decentralized and hybrid FL frameworks, where we jointly modeled communication efficiency and statistical heterogeneity of data and obtained new convergence bounds for distributed ML.

Recently, the integration of blockchain into FL has been investigated. The study in [20] analyzed end-to-end latency for ML model training, update transmission and block mining in a BFL system. [21] analyzed the communication latency and consensus delays for BFL-based vehicular networks, deriving an optimal block arrival rate for vehicles based on system dynamics. The authors in [22] focused on developing a bandwidth allocation and device scheduling solution for digital twin-enabled BFL. Moreover, the work in [23] proposed a BFL-based privacy-preserving UAV network to optimize the energy consumption of UAVs, vehicular device service coverage and a composite service hit ratio. In recent work [24], a dynamic resource allocation framework for BFL was proposed with a focus on maximizing the training data size with respect to energy usage constraints.

B. Motivations and Key Contributions

Despite such research efforts, several limitations still exist in current BFL works, which are highlighted below:

- Most current standard FL [10], [11], [14] and decentralized FL frameworks [18] still rely on a single ES to coordinate model aggregations. In these architectures, single-point-of-failure bottlenecks may disrupt the entire FL system if the server is attacked. Only the work in [22] has considered a multi-server edge computing model for BFL, but its model aggregation still follows traditional FL.
- End-to-end latency optimization, i.e., for both model training and block mining, remains understudied in current BFL systems [20], [21], [22]. Existing works mostly aim to characterize, rather than optimize, BFL latency. Moreover, the benefits of blockchain to support robust BFL training against model attacks have not been yet investigated.
- The potential benefits of edge computing have not been well exploited in existing BFL schemes. Most works [12], [15], [24] have not considered practical resource constraints of mobile IoT devices and the potential for ESs to mitigate resulting straggler effects. Only [13], [14] have considered such a scenario, with the focus instead on partial offloading for traditional FL.
- None of the existing works have analyzed the convergence properties of BFL in a multi-server system. A comprehensive theoretical analysis will provide insights into BFL operations, leading to potential optimization techniques.

Motivated by the aforementioned limitations, we propose a novel consensus-based BFL model for efficient and robust ML model training via blockchain. Specifically, we develop a new cooperative offloading-assisted model learning and resource trading-assisted block mining framework for FL. We then propose a partial model aggregation solution for facilitating global model aggregations at the edge layer using blockchain-enabled P2P communications. Blockchain is important for our methodology in two key ways: (i) it builds trust among MDs and ESs to facilitate reliable ML model sharing and cooperative consensus formation for our federated learning approach; and (ii) it allows for rapid elimination of manipulated models from compromised ESs caused by poisoning attacks, thereby enhancing the robustness of global model training. The system latency is subsequently formulated by considering both model learning latency and block mining latency, which is then optimized by a parameterized actor-critic algorithm. The comparison of our paper with related works in terms of several key design features is summarized in Table I. In summary, the unique contributions of this paper are:

1) We propose a multi-server-assisted BFL architecture, where geo-distributed mobile devices (MDs) communicate with a set of ESs for ML model training and block mining simultaneously. To mitigate straggler effects caused by resource-constrained MDs, an offloading strategy is proposed that enables MD data transmission to an ES for ML model training. Moreover, we develop a resource trading strategy to alleviate block mining latency from resource-limited MDs.
2) We provide a holistic convergence analysis of BFL. In doing so, we consider a new partial model aggregation solution for facilitating global model aggregations at the edge layer via P2P-based blockchain communications. Our resulting bound reveals the impact of the aggregation delay, mini-batch size, and number of P2P communication rounds on the convergence rate.
3) We formulate a new system latency minimization problem, taking into account both offloading-assisted ML model training latency, model consensus latency and block mining latency. This optimization couples data offloading decisions, MD transmit powers, and the allo-

TABLE I

COMPARISON OF METHODOLOGY DESIGN FEATURES BETWEEN OUR PAPER AND RELATED WORKS IN LATENCY OPTIMIZATION AND RESOURCE ALLOCATION FOR FL

Features	[10]	[13]	[18], [19]	[22]	[25]	[26]	[27], [28]	Our work
FL design in multi-edge servers		✓		✓	✓			✓
Data offloading-assisted FL		✓			✓	✓		✓
P2P consensus-based model aggregation			✓					✓
Blockchain-based FL design				✓				✓
DRL-based resource allocation				✓			✓	✓
Parameterized A2C design								✓
FL latency optimization	✓					✓		✓

cation of channel bandwidth, MD computation, and hash power resources to minimize latency with a model quality constraint.

4) To solve the resulting optimization over a mixed discrete and continuous solution space, we propose a novel DRL method based on a parameterized advantage actor critic (A2C) algorithm. We provide a holistic design of the actor, including offloading and allocation policies empowered by trust region policy optimization (TRPO), along with a critic for the state-value training.
5) We conduct numerical experiments for both our consensus-based BFL and parameterized A2C schemes. The results reveal that our BFL scheme outperforms existing FL approaches in terms of model loss and accuracy convergence in both IID and non-IID data settings. Our proposed parameterized A2C scheme also helps lower the system latency by up to 38% compared with state-of-the-art DRL schemes. Moreover, our blockchain-empowered BFL scheme shows high robustness against model poisoning attacks.

C. Paper Organization

The remainder of this paper is organized as follows. Section II presents the BFL architecture and its different components, and analyze the BFL model training convergence. In Section III, we formulate the corresponding system latency minimization problem. To solve the formulated problem, we propose a DRL method based on a parameterized advantage actor critic (A2C) algorithm in Section IV. We present experiments comparing latency and accuracy obtained by our methodology against several baselines in Section V. Finally, Section VI concludes the paper. The key acronyms and notations used in this paper are summarized in Table II and Table III, respectively.

Due to space limitations, note that certain math details have been deferred to our online technical report [29].

II. SYSTEM MODEL

In this section, we describe our BFL system and detail the BFL task model, and then analyze its convergence properties.

A. Overall System Architecture

Our proposed BFL architecture is illustrated in Fig. 1. Each ES is located at a base station (BS) to provide computation

TABLE II

LIST OF KEY ACRONYMS

Acronym	Definition	Acronym	Definition
FL	Federated learning	ML	Machine learning
BFL	Blockchain-based federated learning	MD	Mobile device
ES	Edge server	P2P	Peer-to-peer
UAV	Unmanned aerial vehicle	IID	Independent and identically distributed
DRL	Deep reinforcement learning	A2C	Advantage actor critic
TRPO	Trust region policy optimization	SGD	Stochastic gradient descent
DNN	Deep neural network	DDPG	Deep deterministic policy gradient
MSPBE	Mean Squared Projected Bellman Error	KL	Kullback–Leibler

TABLE III

LIST OF KEY NOTATIONS

Notation	Definition	Notation	Definition
M	Number of ESs	N	Number of MDs
K	Number of sub-channels	D_n	MD's Data size
f_n^ℓ	MD's CPU workload	f_m	ES's CPU workload
p_n	MDs' transmit power	$b_{n,g}$	MD's bandwidth
ϑ	MD's model size	Ψ_n	MD's hash rate
$\hbar$	Hash amount of a block	$b_{m'}$	ES's bandwidth
$p_{m'}$	ES's transmit power	g	Wireless channel
$x_{n,m,g}$	MD's offloading decision	k	Global aggregation round
$\boldsymbol{w}$	ML model parameter	$\mathbf{x}_m$	ES's gradient
$T_{n,m}^{\text{off},(k)}$	MD's offloading latency	$T_m^{\text{exe},(k)}$	ES's execution latency
$T_n^{\text{loc},(k)}$	MD's local data processing latency	$T_{n,m}^{\text{up},(k)}$	MD's model uploading latency
$T_m^{\text{update},(k)}$	ES's model updating latency	$T^{\text{learn},(k)}$	Total learning latency
$T^{\text{cons},(k)}$	Total model consensus latency	$T_n^{\text{mine},(k)}$	Total mining latency

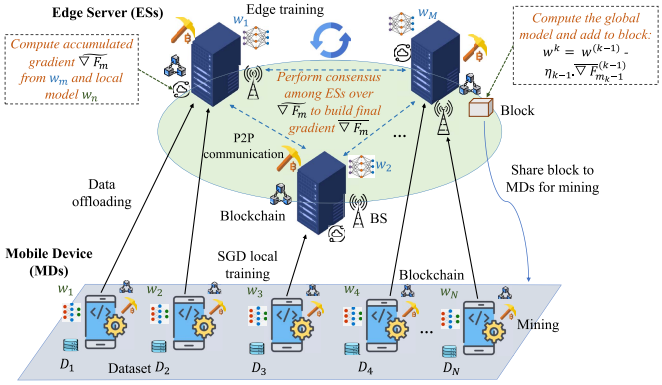


Fig. 1. Our proposed BFL architecture in multi-server edge computing.

services for multiple MDs concurrently. We consider N MDs gathered in the set $\mathcal{N}$, $N = |\mathcal{N}|$, which are connected to M ESs collected in the set $\mathcal{M}$, $M = |\mathcal{M}|$, in a multi-server edge computing network. The goal of the system is to train an ML model, with training proceeding through a series of global aggregation rounds collected via set $\mathcal{K}$. In each round $k \in \mathcal{K}$, each MD n possesses a dataset $\mathcal{D}_n^{(k)}$, which may vary from one round to the next, with size $D_n^{(k)} = |\mathcal{D}_n^{(k)}|$. Each dataset $\mathcal{D}_n^{(k)}$ contains multiple data samples, each consisting of an input feature vector and (e.g., supervised learning) a label. The MDs employ these datasets for local training in round k, formalized in Section II-B. The total dataset is given by the set $\mathcal{D}^{(k)} = \cup_{n\in\mathcal{N}} \mathcal{D}_n^{(k)}$ with size $D^{(k)} = \sum_{n\in\mathcal{N}} D_n^{(k)}$.

The interactions between MDs, ESs, and blockchain components of our BFL system are summarized as follows:

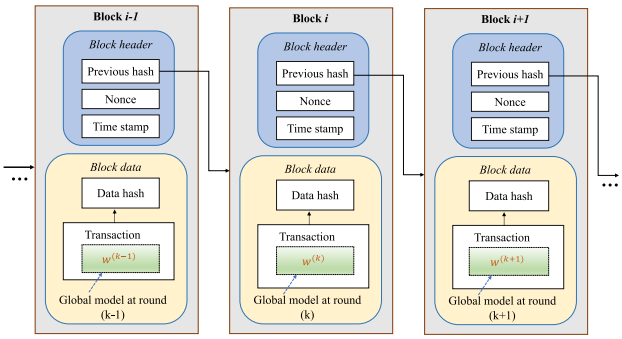


Fig. 2. A block architecture in our blockchain for global model sharing, where a global model $\boldsymbol{w}^{(k)}$ is embedded into the transaction of a block. The transaction is also used to transfer the local gradient $\nabla F_n^{\mathsf{C},(k)}$ of MDs and edge gradient $\mathbf{x}_m^{(l)}$ of ESs in the model uploading and model consensus processes, respectively.

- *MDs* participate in ML model training in a federated manner to serve their intelligence applications (e.g., object detection). Moreover, they work as blockchain nodes to mine blocks containing global models in each communication round to support BFL model sharing.
- *ESs* assist in MD model training by providing computation resources through the offloading process. They also coordinate the model aggregation process to build the global model that is shared with MDs via blockchain.
- *Blockchain* allows MDs to securely transmit their computed local model to ESs via blockchain. It also facilitates the model consensus process between ESs with a traceable data ledger. Moreover, blockchain enables secure global model sharing from ESs to MDs. After the model aggregation process completes, the global model is added to the blockchain, where mining is executed for secure model sharing. Specifically, an ES will be randomly sampled to function as a leader node and build an unverified block which contains the global model. This block is then shared with other ESs and MDs for mining. Subsequently, each MD downloads the verified block from the blockchain to extract the global model which is used for the next training round. As illustrated in Fig. 2, each block includes (i) a header, with a hash and cryptographic nonce, and (ii) the data part. To construct a block, an ES generates a *transaction* using its aggregated model and *hashes* it, resulting in an output of a fixed length. The block is then shared with the MDs for model training.

We let $\mathcal{G}$, $G = |\mathcal{G}|$, denote the set of available sub-channels for communication at a BS. We assume that uplink communications between MDs and ESs follow an OFDMA-based protocol, where each MD is assigned to a sub-channel $g \in \mathcal{G}$ to offload its data in each time slot, and thus each ES can serve at most G MDs in every offloading period. We define the data offloading policy, which incorporates the uplink sub-channel scheduling, through a binary variable $x_{n,m,g}^{(k)}$, $(n \in \mathcal{N}, m \in \mathcal{M}, g \in \mathcal{G})$, where $x_{n,m,g}^{(k)} = 1$ indicates that data $\mathcal{D}_n^{(k)}$ from MD n is offloaded to ES m via sub-channel g in round k, and $x_{n,m,g}^{(k)} = 0$ otherwise. Each dataset can be either trained locally at the MD or offloaded to the ES under a feasible offloading policy $\boldsymbol{X}^{(k)} = \{x_{n,m,g}^{(k)} | x_{n,m,g}^{(k)} \in \{0,1\}, \forall n \in \mathcal{N}, m \in \mathcal{M}, g \in \mathcal{G}\}$. In each training round, the BFL operation consists of four key steps, as depicted in Fig. 1:

1) *Data Offloading and Processing:* Depending on its available resource, each MD can choose to offload its data to one of the nearby ESs for edge learning, or learn the model locally via local data processing. We assume that the model learning process begins once the offloading phase is completed [14]. This assumption is realistic in practical scenarios, where each MD needs to obtain an offloading policy on whether it should offload the data to an ES or not, before it allows the associated ES to train its data. For both training cases, MDs and ESs conduct stochastic gradient descent (SGD) iterations and synchronize their model parameters through block mining. After model training, a MD uploads its ML model parameters to its nearby ES via blockchain. Considering an ES, if it receives data from MDs, it allocates its resources to conduct ML model training.
2) *Partial Model Aggregation and Consensus Update:* Each ES combines its computed ML model with models received from its associated MDs to perform a partial model aggregation. Then, ESs will join a consensus update process in which they conduct multiple rounds of blockchain-enabled *peer-to-peer (P2P)* communications to exchange their models. After the consensus update, an ES will be randomly sampled to work as a leader and build an unverified block that contains the aggregated model for mining.
3) *Block Mining:* The leader ES broadcasts the block via the blockchain to the connected ESs and MDs for mining. In this work, we are mostly interested in the mining latency from the users' perspective, and thus we analyze the mining process at the MDs. Given resource constraints at the MDs, we develop a resource trading strategy where MDs can purchase hash power from the edge/cloud service provider (ESP) (e.g., Amazon cloud services) to run the mining task [30].
4) *Block Generation:* After receipt of the unverified block, the MD that is the first to successfully verify the block will append it to the blockchain and receive a reward. Then, each MD downloads the verified block via its blockchain account [30] that contains the global model, and uses this for local model synchronization to begin the next round of model training.

B. Federated Learning Model

Let $f(\boldsymbol{w}, i) \in \mathbb{R}$ denote the loss function of the ML model (e.g., neural network) associated with data point i, where $\boldsymbol{w} \in \mathbb{R}^d$ is the parameter vector (e.g., weights on neurons). In this work, we consider a scenario of full ML training at MDs, i.e., each MD either keeps its entire data locally or offloads it completely to an ES. When the MD keeps its data local, it offloads the entire trained ML model to the ES. We measure the *online loss function* of each MD n at each global aggregation round k as

$$F_n^{(k)}(\boldsymbol{w}) = \frac{1}{D_n^{(k)}} \sum_{i \in \mathcal{D}_n^{(k)}} f(\boldsymbol{w}, i), \tag{1}$$

and subsequently define the *online global loss* as

$$F^{(k)}(\boldsymbol{w}) = \frac{1}{D^{(k)}} \sum_{n\in\mathcal{N}} D_n^{(k)} F_n^{(k)}(\boldsymbol{w}),\ D^{(k)} = \sum_{n\in\mathcal{N}} D_n^{(k)}. \quad (2)$$

In our BFL system, each ES processes datasets offloaded from a portion of MDs, and also receives the computed ML models uploaded from another portion of nearby MDs which had chosen local processing. We denote $\mathcal{N}_m^{\mathsf{off},(k)}$ and $\mathcal{N}_m^{\mathsf{loc},(k)}$ as the sets of MDs engaged in data offloading and model uploading with ES m, respectively ($\mathcal{N}_m^{\mathsf{off},(k)}, \mathcal{N}_m^{\mathsf{loc},(k)} \subseteq \mathcal{N}$ and $\mathcal{N}_m^{\mathsf{off},(k)} \cap \mathcal{N}_m^{\mathsf{loc},(k)} = \emptyset$). Letting $\mathcal{D}_m^{(k)} = \cup_{n\in\mathcal{N}_m^{\mathsf{off},(k)}} \mathcal{D}_n^{(k)}$ denote ES m's dataset received from MDs, with size $D_m^{(k)}$, the loss at ES m is given as

$$F_m^{(k)}(\boldsymbol{w}) = \frac{1}{D_m^{(k)}} \sum_{i\in\mathcal{D}_m^{(k)}} f(\boldsymbol{w}, i). \quad (3)$$

We now formalize the ML model training procedure at ESs and MDs. Each model training round k starts with the broadcast of a global model, $\boldsymbol{w}^{(k)}$, from one of the ESs. During round k, each ES m performs $e_m^{(k)}$ iterations of SGD over its local/offloaded dataset, which may vary from one ES to another, where the evolution of its local model parameters is given by

$$\boldsymbol{w}_m^{(k),e} = \boldsymbol{w}_m^{(k),e-1} - \frac{\eta_k}{B_m^{(k)}} \sum_{d\in\mathcal{B}_m^{(k),e}} \nabla f(\boldsymbol{w}_m^{(k),e-1}, d), \quad (4)$$

where $\eta_k > 0$ is the step-size and $e \in \{1, \cdots, e_n^{(k)}\}$ is the index of local iteration with $\boldsymbol{w}_m^{(k),0} = \boldsymbol{w}^{(k)}$. In (4), $\mathcal{B}_m^{(k),e}$ denotes the set of data points sampled at the e-th iteration from the local dataset of the respective MD to perform mini-batch SGD. We assume that the mini-batch size $B_m^{(k)} = |\mathcal{B}_m^{(k),e}|$, $\forall e$, is fixed during each local model training round k for each ES m. The local model training at each MD n is also similar to (4), where each MD n performs $e_n^{(k)}$ iterations of SGD with local updates as

$$\boldsymbol{w}^{(k),e} = \boldsymbol{w}_n^{(k),e-1} - \frac{\eta_k}{B_n^{(k)}} \sum_{d\in\mathcal{B}_n^{(k),e}} \nabla f_n(\boldsymbol{w}_n^{(k),e-1}, d) \quad (5)$$

After model training, each ES m computes its *cumulative gradient*:

$$\nabla F_m^{\mathsf{C},(k)} = \left(\boldsymbol{w}^{(k)} - \boldsymbol{w}_m^{(k),e_m^{(k)}}\right)/\eta_k. \quad (6)$$

Similarly, each MD n also obtains its gradient:

$$\nabla F_n^{\mathsf{C},(k)} = \left(\boldsymbol{w}^{(k)} - \boldsymbol{w}_n^{(k),e_n^{(k)}}\right)/\eta_k. \quad (7)$$

Subsequently, the MDs offload their cumulative gradients to their associated ES via blockchain. Each ES $m \in \mathcal{M}$ subsequently acquires its *aggregated gradient*, which is a scaled sum (with respect to the number of SGD iterations and the number of data points) of its cumulative gradient and that of its associated MDs, as

$$\nabla F_m^{\mathsf{A},(k)} = \sum_{n\in\mathcal{N}_m^{\mathsf{loc},(k)}} \frac{D_n^{(k)}}{D^{(k)} e_n^{(k)}} \nabla F_n^{\mathsf{C},(k)} + \frac{D_m^{(k)}}{D^{(k)} e_m^{(k)}} \nabla F_m^{\mathsf{C},(k)}. \quad (8)$$

The ESs then engage in P2P communications for cooperative consensus formation among their aggregated gradients. For this purpose, we assume that they exploit *linear distributed consensus* iterations [31], where during *training round* k, each ES $m \in \mathcal{M}$ conducts $\phi^{(k)} \in \mathbb{N}$ *consensus rounds* of P2P communications with its neighboring ESs. During each round $l = 0, \cdots, \phi^{(k)} - 1$ of P2P communications, the evolution of the local gradient of ES $m \in \mathcal{M}$ can be expressed as:

$$\mathbf{x}_m^{(l+1)} = \lambda_{m,m}^{(k)} \mathbf{x}_m^{(l)} + \sum_{m'\in\varrho(m)} \lambda_{m,m'}^{(k)} \mathbf{x}_{m'}^{(l)}, \quad (9)$$

where $\mathbf{x}_m^{(0)} = \nabla F_m^{\mathsf{A},(k)}$ is ES m's initial local aggregated gradient, and $\mathbf{x}_m^{\left(\phi^{(k)}\right)}$ denotes the local gradient after the consensus process concludes. In (9), $\varrho(m) \subseteq \mathcal{M}$ denotes the set of ESs in the neighborhood of ES m, and $\lambda_{m,m'}^{(k)} \in [0,1]$, $m' \in \{m\} \cup \varrho(m)$ are the *consensus weights* employed at m.

Let $\nabla F_m^{\mathsf{L},(k)} = \mathbf{x}_m^{\left(\phi^{(k)}\right)}$ denote the final local gradient at ES m after the P2P communication process for training round k concludes, which can be expressed as

$$\nabla F_m^{\mathsf{L},(k)} = \underbrace{\sum_{m\in\mathcal{M}} \nabla F_m^{\mathsf{A},(k)}}_{(a)} + \boldsymbol{c}_m^{(k)}, \quad (10)$$

where term (a) is the perfect average of the local aggregated gradients and $\boldsymbol{c}_m^{(k)}$ denotes the error of consensus caused by finite P2P rounds. The selected ES at aggregation round k, denoted by $m_k \in \mathcal{M}$, then adds a boosting coefficient to its local gradient $\nabla F_{m_k}^{\mathsf{L},(k)}$, forming the vector

$$\overline{\nabla F}_{m_k}^{(k)} = \sum_{m\in\mathcal{M}} \sum_{n\in\mathcal{N}_m^{(k)}} \frac{D_n^{(k)} e_n^{(k)}}{D^{(k)}} \nabla F_{m_k}^{\mathsf{L},(k)}, \quad (11)$$

and updates the global model parameter as follows:

$$\boldsymbol{w}^{(k+1)} = \boldsymbol{w}^{(k)} - \eta_k \overline{\nabla F}_{m_k}^{(k)}. \quad (12)$$

$\boldsymbol{w}^{(k+1)}$ is then broadcast across the ESs and MDs via block mining to begin the next round of local model training. The training procedure of our BFL scheme is summarized in Algorithm 1. At the beginning of each global communication round, each MD makes a training decision: edge model training at the ESs or local model training, which is an output of Algorithm 2 (line 5) that will be developed in Section IV. In the case of edge model training, the MD offloads its data to its associated ES (line 7); otherwise, the MD trains its data locally (lines 9-14). After receiving all models from local MDs, each ES performs edge model training (lines 17-21). Once the local training and edge training processes are completed, ESs collaborate to perform P2P-based consensus on aggregated gradients to update the global model (lines 23-35). Finally, the mining is executed, where an ES leader adds the global model parameter to an unverified block and broadcasts it to other ESs and MDs. In particular, each MD performs resource trading to purchase hash power from the ESP to run the Proof-of-Work-based mining. The confirmed block is then appended into the blockchain for global model sharing, where each MD downloads the block for the next round of training (lines 36-40).

C. Convergence Analysis of ML Model Training

We now study the convergence of our BFL scheme. We first make a few assumptions and define some quantities of interest. To obtain convergence guarantees for the distributed consensus process, we make the following assumptions on the consensus weights in (9):

Assumption 1 (Conditions on Consensus Weights [31], [32]): The consensus matrix $\Lambda^{(k)} = [\lambda^{(k)}_{m,m'}]_{m,m' \in \mathcal{M}^{(k)}}$ satisfies the following properties: (i) $\lambda^{(k)}_{m,m'} = 0$ if ESs m and m' are not connected, (ii) $\Lambda^{(k)}\boldsymbol{1} = \boldsymbol{1}$, (iii) $\Lambda^{(k)} = \Lambda^{(k),\top}$, and (iv) $\rho\left(\Lambda^{(k)} - \frac{\boldsymbol{1}\boldsymbol{1}^\top}{|\mathcal{M}^{(k)}|}\right) \leq \lambda^{(k)} < 1$, where $\boldsymbol{1}$ represents the 1s's vector and $\rho(\mathbf{A})$ defines $\mathbf{A}$'s spectral radius.

If $\lambda^{(k)}_{m,m} = 1 - d|\varrho(m)|$ and $\lambda^{(k)}_{m,m'} = d$, $m \neq m'$, $0 < d < 1/M$ [31], the conditions in Assumption 1 hold. These weights can be distributedly obtained at the ESs given a predefined d.

Definition 1 (Gradient Divergence): The divergence of local aggregated gradients across the ESs at global aggregation round k, denoted by $\Xi^{(k)}$, is defined as follows:

$$\left\| \nabla F_m^{\mathsf{A},(k)} - \nabla F_{m'}^{\mathsf{A},(k)} \right\| \leq \Xi^{(k)}, \ \forall m, m' \in \mathcal{M}. \tag{14}$$

Assumption 2 (Smoothness of the Loss Functions [33], [34]): For each MD n that conducts local model training during aggregation round k, the local loss function $F_n^{(k)}$ is β-smooth:

$$\|\nabla F_n^{(k)}(\boldsymbol{w}) - \nabla F_n^{(k)}(\boldsymbol{w}')\| \leq \beta \|\boldsymbol{w} - \boldsymbol{w}'\|, \ \forall \boldsymbol{w}, \boldsymbol{w}'. \tag{15}$$

Also, for each ES m that conducts model training, the loss function $F_m^{(k)}$ is assumed to be β-smooth, which verifies that the global loss function $F^{(k)}$ achieves β-smoothness.

Let $y \in \mathcal{N} \cup \mathcal{M}$ denote the index of an arbitrary MD/ES. Also, let $\mathcal{N}_m^{(k)} = \mathcal{N}_m^{\mathsf{loc},(k)} \cup \{m\}$ denote a set containing the devices which conduct local model training and offload their cumulative gradients to ES m as well as ES m itself. We measure the heterogeneity of data across the MDs/ESs via the following assumption [34]:

Assumption 3 (Bounded Dissimilarity of Local Loss Functions): The finite constants $\zeta_1 \geq 1$, $\zeta_2 \geq 0$ exist for which the following inequality defined on the gradient of the local loss in (2) *and* (3) *holds for any set of coefficients $\{a_y \geq 0\}$ where $\sum_{m \in \mathcal{M}} \sum_{y \in \mathcal{N}_m^{(k)}} a_y = 1$:*

$$\sum_{m \in \mathcal{M}} \sum_{y \in \mathcal{N}_m^{(k)}} a_y \|\nabla F_y^{(k)}(\boldsymbol{w})\|^2 \leq \zeta_1 \left\| \sum_{m \in \mathcal{M}} \sum_{y \in \mathcal{N}_m^{(k)}} a_y \nabla F_y^{(k)}(\boldsymbol{w}) \right\|^2 + \zeta_2, \ \forall k. \tag{17}$$

As can be seen, $\zeta_1 = 1$ and $\zeta_2 = 0$ in (17) correspond to a scenario in which the data across the MDs/ESs is completely homogeneous, and ζ_1 and ζ_2 increase as the heterogeneity across the datasets increases. We next quantify the heterogeneity of data inside each local dataset:

Definition 2 (Local Data Variability): The local data variability at each MD/ES y is denoted by $\Theta_y \geq 0$, which $\forall \boldsymbol{w}, k$ satisfies

$$\|\nabla f_y(\boldsymbol{w}, d) - \nabla f_y(\boldsymbol{w}, d')\| \leq \Theta_y \|d - d'\|, \ \forall d, d' \in \mathcal{D}_n^{(k)}. \tag{18}$$

We further define $\Theta = \max_{y \in \mathcal{N} \cup \mathcal{M}} \{\Theta_y\}$.

Additionally, we let $(\sigma_y^{(k)})^2$ denote the variance across the feature vectors of dataset $\mathcal{D}_y^{(k)}$ at ES/MD y which conducts local model training. We further quantify the dynamics of local datasets via their impact on the ML performance [26]:

Definition 3 (Model/Concept Drift): For each MD/ES y, we calculate the model/concept drift between two FL rounds $k-1$ and k, $\Delta_y^{(k)} \in \mathbb{R}$, which characterizes the local loss's variation induced by data arrival/departure at the MDs and data collection at the ESs, as follows:

$$\frac{D_y^{(k)}}{D^{(k)}} F_y^{(k)}(\boldsymbol{w}) - \frac{D_y^{(k-1)}}{D^{(k-1)}} F_y^{(k-1)}(\boldsymbol{w}) \leq \Delta_y^{(k)}, \forall \boldsymbol{w}. \tag{19}$$

We next present one of our main results, the general convergence behavior of BFL:

Theorem 1 (Convergence Characteristics): Let $\mathcal{N}^{(k)}$, $N^{(k)} = |\cup_{m \in \mathcal{M}} \mathcal{N}_m^{(k)}|$ denote the set of all MDs and ESs engaged in model training during the k-th local model training round. Also, let $e_{\mathsf{max}}^{(k)} = \max_{y \in \mathcal{N}^{(k)}} \{e_y^{(k)}\}$, $e_{\mathsf{avg}}^{(k)} =$

$$\begin{aligned}
&\frac{1}{K} \sum_{k=0}^{K-1} \mathbb{E}\left[\|\nabla F^{(k)}(\boldsymbol{w}^{(k)})\|^2 \right] \\
&\leq \frac{8\sqrt{e_{\mathsf{avg}}^{\mathsf{max}}}}{\alpha \hat{e}_{\mathsf{avg}}^{\mathsf{min}} \sqrt{K}} \left(F^{(0)}(\boldsymbol{w}^{(0)}) - F^{(k)^\star} \right) + \underbrace{\frac{8\sqrt{e_{\mathsf{avg}}^{\mathsf{max}}}}{\alpha \hat{e}_{\mathsf{avg}}^{\mathsf{min}} \sqrt{K}} \sum_{k=1}^{K-1} \Delta^{(k)}}_{(a)} \\
&\quad + \underbrace{\frac{80\beta^2\alpha^2}{K^2 e_{\mathsf{avg}}^{\mathsf{min}}} \sum_{k=0}^{K-1} \sum_{m \in \mathcal{M}} \sum_{y \in \mathcal{N}_m^{(k)}} \frac{D_y^{(k)}}{D^{(k)}} \left(e_y^{(k)} - 1 \right) \left(1 - \frac{B_y^{(k)}}{D_y^{(k)}} \right) \frac{(\sigma_y^{(k)})^2}{B_y^{(k)}} \Theta_y^2}_{(b)} + \underbrace{\frac{80\beta^2\alpha^2}{K^2 e_{\mathsf{avg}}^{\mathsf{min}}} \sum_{k=0}^{K-1} \zeta_2 \left(e_{\mathsf{max}}^{(k)} \right) \left(e_{\mathsf{max}}^{(k)} - 1 \right)}_{(c)} \\
&\quad + \underbrace{\frac{1}{K} \sum_{k=0}^{K-1} 24M(\lambda^{(k)})^{2\varphi^{(k)}} \left(\Xi^{(k)} \right)^2}_{(d)} + \underbrace{\frac{16\beta\alpha\hat{e}_{\mathsf{avg}}^{\mathsf{max}}}{K\sqrt{K}\sqrt{e_{\mathsf{avg}}^{\mathsf{min}}}} \sum_{k=0}^{K-1} \sum_{m \in \mathcal{M}} \sum_{y \in \mathcal{N}_m^{(k)}} \left(\frac{D_y^{(k)}}{D^{(k)} \sqrt{e_y^{(k)}}} \right)^2 \left(1 - \frac{B_y^{(k)}}{D_y^{(k)}} \right) \frac{(\sigma_y^{(k)})^2}{B_y^{(k)}} \Theta_y^2}_{(e)}.
\end{aligned} \tag{13}$$

Algorithm 1 Proposed BFL Algorithm

Input: Global communication rounds $\mathcal{K}$, local training round $e_n, \forall n \in \mathcal{N}$, edge training round $e_m, \forall m \in \mathcal{M}$, number of MDs $\mathcal{N}$, number of ESs $\mathcal{M}$
Initialization: Initialize global model $\boldsymbol{w}^{(0)}$
for each global communication round $k \in \mathcal{K}$ **do**
  for each MD $n \in \mathcal{N}$ **do**
    Determine each training decision $x_{n,m,g}$ via Algorithm 2
    if $x_{n,m,g} = 1$ **then**
      Offload dataset $D_n^{(k)}$ to ES m
    else
      Perform local model training on local dataset $D_n^{(k)}$
      for each local training epoch $e \in e_n$ **do**
        Update local parameters $\boldsymbol{w}_n^{(k),e}$ via (5)
      end for
      Compute the cumulative gradient via (7)
      Add the gradient to a transaction based on the blockchain framework defined in Fig. 2 to transfer to an ES
    end if
  end for
  for each ES $m \in \mathcal{M}$ **do**
    for each edge training epoch $e \in e_m$ **do**
      Update local parameters $\boldsymbol{w}_m^{(k),e}$ via (4)
    end for
    Compute its cumulative gradient via (6)
  end for
  Each ES $m \in \mathcal{M}$ computes an aggregated gradient: $\nabla F_m^{\mathsf{A},(k)}$ using $\nabla F_m^{\mathsf{C},(k)}$ and $\nabla F_m^{\mathsf{C},(k)}$ via (8)
  Set $\mathbf{x}_m^{(0)} = \nabla F_m^{\mathsf{A},(k)}$
  for each P2P consensus round $l \in \phi^{(k)}$ **do**
    for each ES $m \in \mathcal{M}$ **do**
      for each neighboring ES $m' \in \varrho(m)$ of ES m **do**
        Transmit the gradient $\mathbf{x}_{m'}^{(l)}$ via a transaction based on the blockchain framework defined in Fig. 2 to ES m
      end for
      ES m downloads the block of all transactions to obtain neighboring ESs' gradients and computes its gradient $\mathbf{x}_m^{(l+1)}$ via (9)
    end for
  end for
  Obtain the final local gradient at ES m: $\nabla F_m^{\mathsf{L},(k)} = \mathbf{x}_m^{\left(\phi^{(k)}\right)}$
  Add a boosting coefficient to $\nabla F_m^{\mathsf{L},(k)}$ to form $\overline{\nabla F}_{m_k}^{(k)}$ via (11)
  Update the global model parameter $\boldsymbol{w}^{(k+1)}$ via 12
  Add the global model parameter to an unverified block B by the ES leader and broadcast it to other ESs and MDs for mining
  Each MD trades hash resource $\Psi_n^{(k)}$ from the ESP to mine the block
  The fastest MD propagates the verified block to ESs and other MDs for confirmation
  Add the block to the blockchain for global model sharing
  Each MD downloads the block to obtain the global model for next round of training
end for
Output: Final global model $\boldsymbol{w}^{(K)}$

$\sum_{y\in\mathcal{N}^{(k)}} e_y^{(k)}/N^{(k)}$, *and* $\hat{e}_{\mathsf{avg}}^{(k)} = \sum_{y\in\mathcal{N}^{(k)}} D_y^{(k)} e_y^{(k)}/D^{(k)}$. *Assume that* $e_{\mathsf{avg}}^{\min} \le e_{\mathsf{avg}}^{(k)} \le e_{\mathsf{avg}}^{\max}$ *and* $\hat{e}_{\mathsf{avg}}^{\min} \le \hat{e}_{\mathsf{avg}}^{(k)} \le \hat{e}_{\mathsf{avg}}^{\max}$, $\forall k$, *where* $e_{\mathsf{avg}}^{\min}$, $e_{\mathsf{avg}}^{\max}$, $\hat{e}_{\mathsf{avg}}^{\min}$ *and* $\hat{e}_{\mathsf{avg}}^{\max}$ *are four finite positive constants. Further, let* $\Delta^{(k)} = \sum_{m\in\mathcal{N}} \sum_{y\in\mathcal{N}_m^{(k)}} \Delta_y^{(k)}$. *In conducting* K *global aggregation rounds, if the step size satisfies* $\eta_k = \frac{\alpha}{\sqrt{K e_{\mathsf{avg}}^{(k)}}}$, *where* α *is chosen such that* $\eta_k \le \min\left\{\frac{1}{2\beta}\sqrt{\frac{1}{(4\zeta_1+1)\left(e_{\mathsf{max}}^{(k)}\left(e_{\mathsf{max}}^{(k)}-1\right)\right)}}, \frac{D^{(k)}}{2\beta\sum_{y\in\mathcal{N}} D_y^{(k)} e_y^{(k)}}\right\}$, *then the convergence characteristics of the global loss function under BFL follow the bound in* (13)*, shown at the bottom of the previous page.*

Proof: See Appendix A in the technical report [29]. □

The bound in (13) reveals the impact of different device/network configurations on the ML model performance. In particular, the impact of model drift is captured in term (a). Also, the divergence of the global model caused by bias of the local models due to the heterogeneity of data across the MDs/ESs is captured via term (c) which encapsulates ζ_2. Larger daatset heterogeneity captured via ζ_1 imposes a stricter condition on the step size η_k described in the statement of the theorem as well. Terms (b) and (e) in (13) capture the impact of local data heterogeneity ($\{\Theta_y\}$) and mini-batch sizes ($\{B_y^{(k)}\}$) on the performance of the model. Finally, term (d) captures the impact of imperfect local aggregations caused by finite P2P rounds ($\phi^{(k)}$) and divergence of gradients $\Xi^{(k)}$, where the consensus matrix's spectral radius across the ESs ($\lambda^{(k)} < 1$) defined in Assumption 1 determines the rate under which the consensus error decays to zero with respect to the number of P2P rounds $\phi^{(k)}$.

We derive Theorem 1 to obtain conditions for the online global gradient under BFL converges:

Corollary 1 (Convergence Under Proper Choice of Mini-Batch Size and P2P Communication Rounds): Besides conditions in Theorem 1, we assume that (i) the model/concept drift is small enough such that $\Delta^{(k)} \le \frac{\Upsilon}{K}$, $\forall k$, *for some positive constant* Υ*, (ii) the choice of mini-batch size* $B_y^{(k)}$ *at each MD/ES* y *ensures a unified bound* $\left(1 - \frac{B_y^{(k)}}{D_y^{(k)}}\right)\frac{(\sigma_y^{(k)})^2}{B_y^{(k)}}\Theta_y^2 \le \vartheta$, $\forall k$, *where* ϑ *is a finite positive constant, and (iii)* $e_{\mathsf{max}}^{(k)} \le e_{\mathsf{max}}$, $\forall k$ *for some positive constant* e_{max}*. If the number of P2P communications among the ESs at each global aggregation round* k*, i.e.,* $\phi^{(k)}$*, satisfies* $\varphi^{(k)} \ge \frac{1}{2}\left[\log_{\lambda^{(k)}}\left(\frac{\xi}{\sqrt{K}\left(\Xi^{(k)}\right)^2 m^{(k)}}\right)\right]^+$ *for some positive constant* ξ*, where* $\lambda^{(k)}$ *is defined in Assumption 1, the gradient of the global loss under BFL satisfies the upper bound in* (16)*, shown at the bottom of the page, which implies* $\frac{1}{K}\sum_{k=0}^{K-1}\mathbb{E}\left[\|\nabla F^{(k)}(\boldsymbol{w}^{(k)})\|^2\right] = \mathcal{O}\left(\frac{1}{\sqrt{K}}\right)$*, and thus* $\lim_{K\to\infty}\frac{1}{K}\sum_{k=0}^{K-1}\mathbb{E}\left[\|\nabla F^{(k)}(\boldsymbol{w}^{(k)})\|^2\right] \to 0$.

Proof: See Appendix B in the technical report [29]. □

III. System Latency Model

In this section, we analyze the latencies of the model training and block mining processes in detail, and present our latency optimization problem.

$$\frac{1}{K}\sum_{k=0}^{K-1}\mathbb{E}\left[\|\nabla F^{(k)}(\boldsymbol{w}^{(k)})\|^2\right] \le \frac{8\sqrt{e_{\mathsf{avg}}^{\max}}}{\alpha\hat{e}_{\mathsf{avg}}^{\min}\sqrt{K}}\left(F^{(0)}(\boldsymbol{w}^{(0)}) - F^{(k)^\star}\right) + \frac{8\Upsilon\sqrt{e_{\mathsf{avg}}^{\max}}}{\alpha\hat{e}_{\mathsf{avg}}^{\min}\sqrt{K}} + \frac{80\beta^2\alpha^2}{Ke_{\mathsf{avg}}^{\min}}\left(e_{\mathsf{max}}-1\right)\vartheta + \frac{80\beta^2\alpha^2}{Ke_{\mathsf{avg}}^{\min}}\zeta_2\left(e_{\mathsf{max}}\right)\left(e_{\mathsf{max}}-1\right) + \frac{24\xi}{\sqrt{K}} + \frac{16\beta\alpha\hat{e}_{\mathsf{avg}}^{\max}}{\sqrt{K}\sqrt{e_{\mathsf{avg}}^{\min}}}\vartheta \tag{16}$$

A. Latency of Model Training in BFL

In our BFL system, an MD can offload its data to the ES for edge learning or choose to train the ML model locally in each global aggregation $k \in \mathcal{K}$. In the case of offloading ($x^{(k)}_{n,m,g} = 1$), the latency for model training consists of data communication latency and execution latency at the ES. The data communication latency of MD n when offloading its data $D^{(k)}_n$ to ES m is given by

$$T^{\mathsf{off},(k)}_{n,m} = \sum_{g\in\mathcal{G}} x^{(k)}_{n,m,g} \frac{D^{(k)}_n}{R^{(k)}_{n,m}}, \forall n \in \mathcal{N}, \tag{20}$$

where $R^{(k)}_{n,m}$ is the transmission rate (in bits/s) from MD n to ES m, which is given by $R^{(k)}_{n,m} = \sum_{g\in\mathcal{G}} x^{(k)}_{n,m,g} b^{(k)}_{n,g} \log_2\left(1+\frac{p^{(k)}_n h^{(k)}_{n,m,g}}{\sigma^2+\sum_{j\in\mathcal{N}} \setminus \mathcal{N}^{\mathsf{off}}_m \left(x^{(k)}_{j,m,g} p^{(k)}_j h^{(k)}_{j,m,g}\right)}\right)$, under interference $\sum_{j\in\mathcal{N}} \setminus \mathcal{N}^{\mathsf{off}}_m \left(x^{(k)}_{j,m,g} p^{(k)}_j h^{(k)}_{j,m,g}\right)$ caused by a group of MDs ($\mathcal{N} \setminus \mathcal{N}^{\mathsf{off}}_m$) that are associated with other ESs on sub-channel g. Here, $b^{(k)}_{n,g}$ is the bandwidth (in Hz) allocated to channel g under the policy $\boldsymbol{B}^{(k)} = \{b^{(k)}_{n,g} | 0 < b^{(k)}_{n,g} \leq W, \forall n \in \mathcal{N}, g \in \mathcal{G}, k \in \mathcal{K}\}$, where W is the maximum system bandwidth constraint. Further, $p^{(k)}_n$ is transmission power (in Watts) of MD n in the offloading subject to the power constraint P_n under a policy $\boldsymbol{P}^{(k)} = \{p^{(k)}_n | 0 < p^{(k)}_n \leq P_n,\ n \in \mathcal{N}^{\mathsf{off}}_m,\ \forall m \in \mathcal{M}\}$, and $h^{(k)}_{n,m,g}$ is the gain of wireless channel between the MD m and ES n on sub-channel g. Accordingly, the energy consumption for data offloading at aggregation round k at MD n is given as $E^{\mathsf{off}(k)}_{n,m} = \sum_{g\in\mathcal{G}} x^{(k)}_{n,m,g} p^{(k)}_n T^{\mathsf{off},(k)}_{n,m}$.

After the offloading process, ES m receives a combined dataset $\mathcal{D}^{(k)}_m$ (as defined in Section II-B) with size $D^{(k)}_m = |\mathcal{D}^{(k)}_m|$ from MDs engaged in data offloading. Then, the ES allocates its computational resource to perform the SGD-based model training. The data processing latency at ES m is thus given by

$$T^{\mathsf{exe},(k)}_m = \sum_{g\in\mathcal{G}} x^{(k)}_{n,m,g} \frac{C_m D^{(k)}_m \varrho^{(k)}_m e^{(k)}_m}{f_m}, m \in \mathcal{M}, \tag{21}$$

where C_m represents how many CPU cycles is required to calculate the gradient per data point at ES m, and $\varrho^{(k)}_m \in (0, 1]$ is the mini-batch size ratio, from which the size of SGD mini-batches can be written as $B^{(k)}_m = \varrho^{(k)}_m D^{(k)}_m$. Moreover, $e^{(k)}_m$ is the number of SGD iterations, and f_m is the fixed computational capability of ES m (in CPU cycles/s).

On the other hand, each MD n can also choose to locally process its data, implying $x^{(k)}_{n,m,g} = 0, \forall m, g$. We denote $f^{\ell,(k)}_n$ as the computational capability of MD n (in CPU cycles/s) allocated to train the data given maximum capacity F_n, which is represented via a policy: $\boldsymbol{F}^{(k)} = \{f^{\ell,(k)}_n | 0 < f^{\ell,(k)}_n \leq F_n, \forall n \in \mathcal{N}\}$. The latency of model training over $e^{(k)}_n$ SGD iterations at MD n is given by

$$T^{\mathsf{loc},(k)}_n = \left(1 - \sum_{g\in\mathcal{G}} x^{(k)}_{n,m,g}\right) \frac{C_n D^{(k)}_n \varrho^{(k)}_n e^{(k)}_n}{f^{\ell,(k)}_n}, \tag{22}$$

where $\varrho^{(k)}_n \in (0, 1]$ is the SGD mini-batch ratio, and C_n is the number of CPU cycles required to compute the gradient per data point at MD n. Also, the local energy consumption of MD n is given by $E^{\mathsf{loc},(k)}_n = \left(1 - \sum_{g\in\mathcal{G}} x^{(k)}_{n,m,g}\right) \kappa (f^{\ell,(k)}_n)^2 C_n$, where κ is the energy coefficient depending on the chip architecture. After completing the local model training, MD n then transmits the computed model $\nabla F^{\mathsf{A},(k)}_n$ to its associated ES via blockchain. The latency of this model uploading can be given as

$$T^{\mathsf{up},(k)}_{n,m} = \left(1 - \sum_{g\in\mathcal{G}} x^{(k)}_{n,m,g}\right) \frac{\vartheta}{R^{(k)}_{n,m}}, \forall n \in \mathcal{N}, m \in \mathcal{M}, \tag{23}$$

where ϑ is the gradient/model size (in bits) that is the same across the MDs. Based on above formulations, the total latency of model training of the BFL system at each global aggregation k is

$$T^{\mathsf{learn},(k)} = \sum_{n\in\mathcal{N}} \sum_{m\in\mathcal{M}} T^{\mathsf{off},(k)}_{n,m} + \sum_{m\in\mathcal{M}} T^{\mathsf{exe},(k)}_m + \sum_{n\in\mathcal{N}} T^{\mathsf{loc},(k)}_n + \sum_{n\in\mathcal{N}} \sum_{m\in\mathcal{M}} T^{\mathsf{up},(k)}_{n,m}, \forall k \in \mathcal{K}. \tag{24}$$

After the model training process, the ESs in the edge layer perform the P2P-based consensus on the computed model. An ES performs the model updating where the neighbours exchange their aggregated gradients defined in (8) via P2P communications. The latency of parameter updating at ES m in each P2P communication round l is determined by

$$T^{\mathsf{update},(k)}_m(l) = \max_{m'\in\Xi(m)} \frac{\vartheta}{R^{(k)}_{m,m'}(l)}, \tag{25}$$

where $R^{(k)}_{m,m'}(l)$ is the transmission rate from ES m' to ES m via wired communications. Thus the updating latency of each ES m over φ consensus rounds is $T^{\mathsf{cons},(k)}_m = \sum_{l=1}^{\varphi} T^{\mathsf{update},(k)}_m(l)$. Finally, the latency of the model consensus is determined by the slowest ES as below:

$$T^{\mathsf{cons},(k)} = \max_{m\in\mathcal{M}} T^{\mathsf{cons},(k)}_m = \max_{m\in\mathcal{M}} \left[\sum_{l=1}^{\varphi} T^{\mathsf{update},(k)}_m(l)\right]. \tag{26}$$

B. Latency of Block Mining in BFL

After model consensus among ESs, an ES will be selected to work as a leader that builds an unverified block B and broadcasts it to all connected ESs and participating MDs for universal block mining. This selection can be based on the ESs' reputation in the previous aggregation round from the P2P collaboration process, where the reputation evaluation methodology can be used to quantify each ES's contribution to block generation [35]. Inspired by our previous work [36], we adopt mining latency as the reputation metric: an ES which generates the fastest block in the previous mining round will

have the highest reputation and is selected as the leader for mining coordination in the current mining round.

After leader selection, the mining process is executed. Similar to existing works [8], [9], we focus on *mobile mining* analysis, where the mining latency at MDs is considered, and thus the mining analysis at ESs is ignored. We adopt the popular Proof-of-Work mining mechanism [6], [7] for our BFL, where MDs compete to mine the block. The adoption of other mining mechanisms in BFL will be considered in future works. Conceptually, the mining latency at each MD n mainly consists of block generation latency and block propagation latency [24]. Due to resource constraints, we allow MDs to implement resource trading to buy hash power from the ESP to assist their mining. Accordingly, MDs compete with each other to gain the maximum hash power allocation from the ESP's hash resource pool, aiming to increase the probability of becoming the mining winner for gaining rewards. In every global aggregation round k, each MD n specifies its mining demand to trade hash resource, denoted as $\Psi_n^{(k)}$ (Hash/sec) subject to the constraint of the total hash power of the BFL system Ψ_{max} under a policy $\boldsymbol{\Psi}^{(k)} = \{\Psi_n^{(k)} | 0 < \Psi_n^{(k)} \leq \Psi_{\mathsf{max}}, \forall n \in \mathcal{N}\}$. Let $\hbar^{(k)}$ denote the hash amount (in hash) required to mine the block B (which also represents the size) in aggregation round k, the block generation latency at MD n can be specified as $T_n^{\mathsf{gen},(k)} = \frac{\hbar^{(k)}}{\Psi_n^{(k)}}$. Thus, the energy consumption for block generation at MD n can be given as $E_n^{\mathsf{gen},(k)} = \Xi_n \hbar^{(k)}$, where Ξ_n is the power efficiency of the mining rig of MD n (J/hash).

After generating a block, MD n will propagate it to ESs and other MDs for confirmation. The latency of this block propagation process can be determined as $T_n^{\mathsf{prop},(k)} = \xi(B|L^{M+N-1}|)^{(k)}$, where ξ is a parameter to quantify the efficiency of block verification. Further, $B|L^{M+N-1}|$ represents the average delay of the repeated verification on the block B of all entities except MD n [36]. Therefore, the mining latency at an MD can be given as $T_n^{\mathsf{mine},(k)} = T_n^{\mathsf{gen},(k)} + T_n^{\mathsf{prop},(k)}$.

However, in the block mining process, there is the possibility that a MU n generates the block and propagates it slower than other miners which will discard this block from blockchain. This issue is called forking and such a block is called an orphaned one. The forking probability can be determined as $P_{\mathsf{fork}} = 1 - e^{-\iota \phi(s_n)}$, where ι is set to $\iota = 1/600(sec)$ [30]. Moreover, s_n represents how many transactions are included in the block mined by MD n, and $\phi(s_n)$ is a function of block size. Therefore, the mining latency at MD n can be rewritten as

$$T_n^{\mathsf{mine},(k)} = \zeta(T_n^{\mathsf{gen},(k)} + T_n^{\mathsf{prop},(k)}), \forall k \in \mathcal{K}, \tag{27}$$

where ζ is the number of forking occurrences in each global training round. Therefore, the total mining latency of BFL system can be given as $T^{\mathsf{mine},(k)} = \sum_{n \in \mathcal{N}} T_n^{\mathsf{mine},(k)}$.

C. Formulation of System Latency Problem

Our objective is to optimize the total latency of the BFL system from the user perspective as the sum of model training latency, model consensus latency and block mining latency at a certain aggregation round k:

$$\underset{\boldsymbol{X},\boldsymbol{P},\boldsymbol{B},\boldsymbol{F},\boldsymbol{\Psi}}{\text{minimize}} \frac{1}{K} \sum_{k=0}^{K-1} \left(T^{\mathsf{learn},(k)} + T^{\mathsf{cons},(k)} + T^{\mathsf{mine},(k)} \right) \tag{28a}$$

$$\text{s.t. } x_{n,m,g}^{(k)} \in \{0,1\}, \forall n \in \mathcal{N}, m \in \mathcal{M}, g \in \mathcal{G}, \tag{28b}$$

$$\sum_{m \in \mathcal{M}} \sum_{g \in \mathcal{G}} x_{n,m,g}^{(k)} \leq 1, n \in \mathcal{N}, \tag{28c}$$

$$0 < p_n^{(k)} \leq P_n, \forall n \in \mathcal{N}_n, \tag{28d}$$

$$0 < b_{n,g}^{(k)} \leq W, \forall n \in \mathcal{N}, g \in \mathcal{G}, \tag{28e}$$

$$0 < f_n^{\ell,(k)} \leq F_n, \forall n \in \mathcal{N}, \tag{28f}$$

$$0 < \Psi_n^{(k)} \leq H, \forall n \in \mathcal{N}_n, \tag{28g}$$

$$0 < E_n^{\mathsf{learn},(k)} + E_n^{\mathsf{gen},(k)} \leq E_n^{\mathsf{max},(k)}, \forall n \in \mathcal{N}_n, \tag{28h}$$

$$\frac{1}{K} \sum_{k=0}^{K-1} \mathbb{E}\left[\|\nabla F^{(k)}(\boldsymbol{w}^{(k)})\|^2 \right] \leq \epsilon, \tag{28i}$$

where $E_n^{\mathsf{learn},(k)} = \sum_{g \in \mathcal{G}} x_{n,m,g}^{(k)} E_{n,m}^{\mathsf{off},(k)} + \left(1 - \sum_{g \in \mathcal{G}} x_{n,m,g}^{(k)}\right) E_n^{\mathsf{loc},(k)}$ is the energy consumption for model training. Here, constraints (28b) and (28c) imply that each dataset can be either trained locally or offloaded to at most one ES via a sub-channel. (28d) ensures the transmit power constraint of each MD. Constraint (28e) guarantees that each MD n is allocated a feasible bandwidth resource for data offloading. The MD also allocates a positive computational resource to train its ML model with respect to the maximum CPU capability F_n, as indicated in (28f). Constraint (28g) guarantees that the hash power allocated to each MD is limited by the total system hash resource. Further, constraint (28h) implies that the energy consumption of MD n for model training and blockchain mining is limited by its battery energy level. Finally, the global loss function should be less than a desirable value ϵ to ensure the required training quality, as indicated in constraint (28i).

The optimization problem in (28) is non-convex with respect to the mixed discrete offloading and continuous allocation variables. Due to the time-varying nature of system states, such as channel condition and computational availability, it is challenging to directly solve the formulated problem via conventional optimization approaches such as Lyapunov optimization [24]. Therefore, we will propose to use a learning-based approach, where a new DRL algorithm is developed to well capture the dynamics of system and integrate them into the solution design.

IV. DRL Design With Parameterized A2C for BFL

Different from the existing DRL algorithms which consider either purely discrete [22], [27], [37] or purely continuous actions [12], [28], [38], we here study a more practical DRL setting with a hybrid discrete-continuous action for improving the training performance. Even though such a hybrid action setting has been previously mentioned in a few related works such as [39], a holistic investigation on the sampling of discrete and continuous actions has not been given.

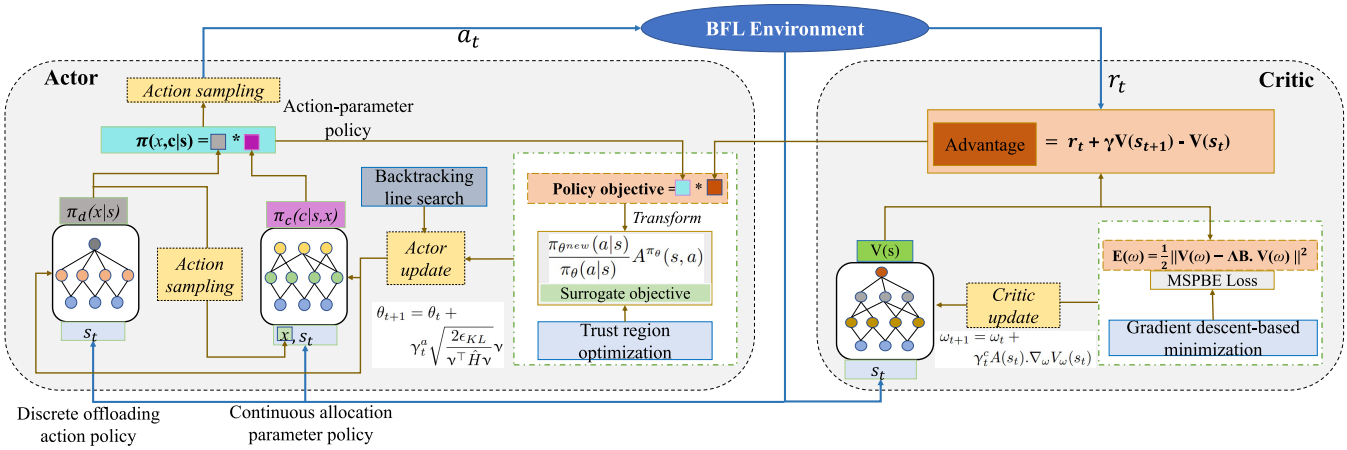


Fig. 3. The proposed parameterized A2C architecture for our BFL environment.

Therefore, we propose a parameterized advantage actor critic (A2C) algorithm to optimize the system latency, as illustrated in Fig. 3. We consider a hybrid discrete and continuous action space, where the resource allocation variables in (28) are continuous, while the offloading decision variables are discrete. The actor is designed to train both offloading and resource allocation policies, which we demonstrate in Section IV-C. The critic is then designed to evaluate the efficiency of the actor policy training, which we present in Section IV-D. We also include the advantage function in the critic design as it has been shown to reduce the variance in actor policy training compared with conventional actor-critic approaches [28], [40]. Finally, we develop a training procedure to optimize the long-term system utility of our developed A2C algorithm (Section IV-E), which corresponds to minimizing the long-term BFL system latency.

A. DRL Formulation

We consider a single-agent DRL problem setting [12], [28], where a virtual centralized agent interacts with the BFL environment induced by the interaction between MDs and ESs during model training and block mining. Our DRL scheme aims to optimize the total system latency in (28a) consisting of model training latency, consensus latency and block mining latency. To handle the formulated optimization problem, we build a centralized agent which defines its reward based on the total utility, and restricts its action space based on the resource allocation and learning constraints in (28). With a comprehensive view of the model training and block mining processes, the agent can obtain the system state and employ it to take efficient actions via well-trained data offloading and resource allocation policies that are observed to minimize the system latency. We consider a parameterized Markov Decision process characterized by $\mathcal{M} =< \mathcal{S}, \mathcal{A}, r >$. Here, $\mathcal{S} = \{s_1, \ldots, s_N\}$ and $\mathcal{A} = \{a_1 \ldots, a_N\}$ are the finite sets of state and parameterized action, respectively. Further, $r(s,a) : \mathcal{S} \times \mathcal{A} \rightarrow [-R_o, R_o]$, $(R_o \in \mathbb{R}_{>0})$ denotes the bounded system reward. We also denote $P(s'|s,a)$ as the probability of transition executed by action a of the agent from s to s'.

In our BFL latency optimization problem, the offloading decisions for ML model training at MDs are closely associated with resource allocation variables. For instance, to offload the data to an ES, an MD needs to tune its transmit power and determine channel availability; or to execute data locally, it should determine its computational utilization. We consider a parameterized action space: a finite set of discrete offloading action $a \in \mathcal{A}_a = \{a_1, a_2, \ldots, a_n\}$ defined via a discrete-action policy $\pi^d(a|s)$ and each a_n has a set of continuous parameters $\{c \in \mathcal{A}_c\}$ defined via an action-parameter policy $\pi^c(c|s,x)$. Thus, the joint action is given by conditional probability $(a,c) \sim \pi_\theta(a,c|s) = \pi^d_{\theta_d}(a|s)\pi^c_{\theta_c}(c|s,a)$, where θ is the parameter of the overall action policy, and θ_d and θ_c are parameters of discrete action policy and parameter policy, respectively, with $\theta = [\theta_d, \theta_c]$. To simplify the notation, $\pi_\theta(a,c|s)$ is expressed by $\pi_\theta(a,s)$ in the rest of the paper.

At each time step t, the agent at state s_t takes an action a_t to transition to the next state s_{t+1} and observe a reward $r_{t+1} \leftarrow r(s_t, a_t, s_{t+1})$. As a result, the training data for DRL is produced in a form of *trajectory* where each data point on the trajectory can be represented by tuple $\{s_t, a_t, r_t, s_{t+1}\}$. Given a policy π, the long-term discounted system reward is characterized by the state-value function $V^{\pi_\theta}(s) : \mathcal{A} \rightarrow \mathbb{R}$ and the action-value function $Q^{\pi_\theta}(s,a) : \mathcal{S} \times \mathcal{A} \rightarrow \mathbb{R}$, which are $V^{\pi_\theta}(s) = \mathbb{E}^{\pi_\theta}[\sum_{t=0}^{\infty} \gamma_t r_{t+1} | s_t = s]$ and $Q^{\pi_\theta}(s,a) = \mathbb{E}^{\pi_\theta}[\sum_{t=0}^{\infty} \gamma_t r_{t+1} | s_t = s, a_t = a]$, where $\gamma_t \in [0,1]$ is the discount value and $\mathbb{E}^{\pi_\theta}[.]$ represents expectation of the executed reward function under policy π. By using the Bellman optimality equation, the Q-value associated with a state-action pair can also be expressed by $Q^{\pi_\theta}(s_t, a_t) = \mathbb{E}^{\pi_\theta}[r_{t+1} + \gamma V^{\pi_\theta}(s_{t+1})]$. In this paper, we focus on the A2C model that can be characterized by the advantage function $A^{\pi_\theta}(s,a)$

$$\begin{aligned} A^{\pi_\theta}(s_t, a_t) &= Q^{\pi_\theta}(s_t, a_t) - V^{\pi_\theta}(s_t) \\ &= r_{t+1} + \gamma V^{\pi_\theta}(s_{t+1}) - V^{\pi_\theta}(s_t), \forall s \in \mathcal{S}, a \in \mathcal{A}. \end{aligned} \tag{29}$$

It is worth noting that the proposed A2C algorithm training is performed in a certain FL aggregation round k to allow the agent to obtain an optimal latency-aware offloading and allocation strategy for MDs. Based on the well trained A2C model, we then deploy it into the BFL environment to guide the FL training across aggregation rounds. Therefore, the index k is dropped for the simplicity of notations in our DRL formulation. In the following, we define state, action and reward for our DRL algorithm.

1) State: In our BFL environment, the system state consists of five components: data state $S_{\mathsf{data}}(t)$, channel state $S_{\mathsf{channel}}(t)$, bandwidth state $S_{\mathsf{band}}(t)$, computation state $S_{\mathsf{comp}}(t)$, and hash power state $S_{\mathsf{hash}}(t)$. Therefore, the system state is defined as:

$$s(t) = \{S_{\mathsf{data}}(t), S_{\mathsf{channel}}(t), S_{\mathsf{band}}(t), S_{\mathsf{comp}}(t), S_{\mathsf{hash}}(t)\}. \tag{30}$$

Here, $S_{\mathsf{data}}(t)$ is defined as $S_{\mathsf{data}}(t) = \{D_n(t)\}_{n\in\mathcal{N}}$. Further, $S_{\mathsf{channel}}(t) = \{q_{n,g}(t)\}_{n\in\mathcal{N},g\in\mathcal{G}}$ indicates whether the sub-channel g is used by MD n at time slot t. If yes, $q_{n,g}(t) = 1$, otherwise $q_{n,g}(t) = 0$. The bandwidth state $S_{\mathsf{band}}(t)$ can be given $S_{\mathsf{band}}(t) = \{b_{n,g}(t)\}_{n\in\mathcal{N},g\in\mathcal{G}}$ under the total radio system bandwidth W. The computation state $S_{\mathsf{comp}}(t)$ contains the information of current computational resource $S_{\mathsf{comp}}(t) = \{f_n^{\ell}(t)\}_{n\in\mathcal{N}}$. Lastly, the hash power state $S_{\mathsf{hash}}(t)$ presents the current hash power Ψ_n of all MDs: $S_{\mathsf{hash}}(t) = \{\Psi_1(t), \Psi_2(t), \ldots, \Psi_N(t)\}$.

2) Action: Our BFL system features a parameterized action space with offloading or local execution, as elaborated below:

- **Offloading** *(transmit power, channel bandwidth allocation, hash power allocation)*: When MD n chooses the offloading mode $x_{n,m,g} = 1$, it must determine relevant parameters, i.e., transmit power p_n and channel bandwidth $b_{n,g}$ that are needed for offloading. Also, MDs perform mining regardless of their learning status, and thus we also involve a hash power allocation parameter Ψ_n. Therefore, the joint action on each MD can be expressed by $a_n = \{x_{n,m,g}, p_n, b_{n,g}, \Psi_n\}$ and the complete system action in this mode is $a(t) = x_{n,m,g}(t), p_n(t), b_{n,g}(t), \Psi_n(t), \forall n \in \mathcal{N}, m \in \mathcal{M}, g \in \mathcal{G}$.
- **Local Execution** *(computational allocation, hash power allocation)*: When MD n chooses the local execution mode $x_{n,m,g} = 0$, it must specify its necessary parameters to execute the model training, e.g., f_n^{ℓ}. Moreover, similar to the offloading mode, the parameter of hash power allocation is also involved to support the block mining task executed after the model learning. Therefore, the joint action on each MD is $a_n = \{x_{n,m,g}, f_n^{\ell}, \Psi_n\}$ and the complete action for the BFL system in this local execution mode is given by $a(t) = x_{n,m,g}(t), f_n^{\ell}(t), \Psi_n(t), \forall n \in \mathcal{N}, m \in \mathcal{M}, g \in \mathcal{G}$.

3) System Reward Function: The reward in our BFL system comes from the joint model learning and block mining, by maximizing the system returns in the long run. However, in the optimization problem (28), our objective is to minimize the system latency in a certain aggregation round, which requires a negative multiplication before being used as the reward, i.e., we define the reward as $r(s_t, a_t) = -\left(T^{\mathsf{learn}} + T^{\mathsf{cons}} + T^{\mathsf{mine}}\right)$. For better presentation, we transform the latency minimization problem into a system utility optimization problem by using a simple exponential equation:

$$U = \left[e^{\left(1 - \frac{\left(T^{\mathsf{learn}} + T^{\mathsf{cons}} + T^{\mathsf{mine}}\right)}{\tau}\right)} - 1 \right], \tag{31}$$

where τ denotes an upper bound of the system latency. (31) implies that the lower system latency results in a higher system utility. Therefore, instead of minimizing the latency, we are keen on maximining the system utility which better characterizes the efficiency of our algorithm. Accordingly, we transform (28) to the following optimization problem:

$$\underset{\boldsymbol{X},\boldsymbol{P},\boldsymbol{B},\boldsymbol{F},\boldsymbol{\Psi}}{\text{maximize}} \quad U \tag{32a}$$

$$\text{s.t.} \quad 28b - \ 28i. \tag{32b}$$

Thus, we re-define the system reward as a result of executing the action with given states as $r(s_t, a_t) = U(t)$.

B. Policy Gradient Update for A2C

We first analyze the policy gradient update necessary for the design of actor and critic components that will be elaborated later. In the BFL environment, the agent tries to search among the set of parameterized offloading policies to obtain the optimal policy $\pi_\theta^*(a, s)$ that can return the maximum reward, i.e., system utility. However, in practice the search space may be very large and the agent may not be able to find the optimal policy. Thus, we restrict the policy set by a vector $\theta \in \mathbb{R}^z$ for some integer $z > 0$ and perform the optimization over the group of the parameterized policies $\pi_\theta(a, s)$. To facilitate our analysis, the following assumptions are introduced.

Assumption 4: The policy π_θ and $\mathcal{P}(s'|s, a)$ guarantee an irreducible and aperiodic Markov chain described by $\mathcal{P}^{\pi_\theta}(s'|s)$, $\forall\theta$. Therefore, there exists a stationary distribution defined as $d^{\pi_\theta}(s)$ on policy θ.

Assumption 4 is a common assumption for actor-critic algorithms [28], [40]. Accordingly, we define $J(\pi_\theta) = r(\pi_\theta)$, where r is the system reward defined in Section IV-A.3, as the performance function with respect to the policy parameterized by θ. Accordingly, the objective function with linear Q-value function approximation is given by

$$J(\pi_\theta) = \sum_{s\in\mathcal{S}} d^{\pi_\theta}(s) Q^{\pi_\theta}(s, a). \tag{33}$$

Next, similar to [40], we make an assumption on the policy $\pi_\theta(a, s)$.

Assumption 5: The following assumptions are made on the policy function:

- *Positive policy function:* $\pi_\theta(a|s) > 0, \forall\theta \in \mathbb{R}^d$
- *Bounded policy gradient*:

$$||\nabla \log \pi_\theta(a|s)||_2 < G_\pi, \forall\theta, \forall s, \forall a, G_\pi > 0$$

- *ℓ-Lipschitz policy gradient*:

$$||\nabla \log \pi_{\theta_1} - \nabla \log \pi_{\theta_2}||_2 \leq \ell||\theta_1 - \theta_2||_2, \forall\theta_1, \forall\theta_2$$

Here, the regularity conditions in Assumption 5 can be satisfied by using the Gibbs softmax distribution network, e.g., in deep neutral networks, for action selection in the actor. Under this assumption, the policy gradient π_θ can be updated as $\nabla J(\pi_\theta) = \mathbb{E}_{s\sim d^{\pi_\theta}(.), a\sim\pi_\theta(.|s)} [Q^{\pi_\theta}(s, a) \nabla \log \pi_\theta(a|s)]$.

C. Actor Design

In our A2C-based DRL algorithm, the actor aims to update the parameter θ over time-step iterations to find the optimal policy π_θ^* that characterizes the best trajectory for our system utility optimization problem. In other words, the actor is expected to make optimal model learning and block mining decisions in a fashion that the long-term reward (i.e., system utility) is maximized. In doing so, the actor needs to use the gradient $\nabla J(\pi_\theta)$ to optimize its policy

$$\max_{\theta \in \mathbb{R}^d} J(\pi_\theta) = \mathbb{E}_{s\sim d^{\pi_\theta}(.)}\left[\pi_\theta(a|s) A^{\pi_\theta}(s,a)\right], \tag{34}$$

Traditionally, the policy is optimized via a vanilla policy gradient algorithm by direct policy search over the entire exploration space which is known to be inefficient. Instead, we use trust region policy optimization (TRPO) to improve policy optimization by maximizing a surrogate objective over a trust-region [41]. Accordingly, (34) can be re-written

$$\begin{aligned} &\max_{\theta \in \mathbb{R}^d} J(\pi_\theta) = \mathbb{E}_{\pi_\theta}\left[\frac{\pi_{\theta^{\mathsf{new}}}(a|s)}{\pi_\theta(a|s)} A^{\pi_\theta}(s,a)\right], \\ &\textit{subject to} \quad \mathbb{E}_s\left[KL\left(\pi_\theta(.|s)||\pi_{\theta^{\mathsf{new}}}(.|s)\right)\right] \le \epsilon_{KL}, \end{aligned} \tag{35}$$

for some $\epsilon_{KL} > 0$. In (35), $\mathbb{E}_s$ represents the state visitation distribution induced by π_θ, and θ and θ^{new} are the vectors of policy parameters before and after each update, respectively. By enforcing a Kullback-Leibler (KL)-divergence constraint KL, the probability distributions of the policy before and after the update will be kept closely in the parameter space to avoid divergence in the gradient update. Considering our parameterized action space including the offloading actions and their parameters, the optimization objective function can be expressed as $\mathbb{E}_{\pi_\theta}\left[\frac{\pi^d_{\theta^{\mathsf{new}}_d}(a|s)\pi^c_{\theta^{\mathsf{new}}_c}(c|s,a)}{\pi^d_{\theta_d}(a|s)\pi^c_{\theta_c}(c|s,a)} A^{\pi_{\theta'}}(s,a)\right]$.

To solve (35), we construct a closed form solution for computing the KL-divergence between the distributions of the discrete offloading action policy and the action-parameter policy for our BFL problem as follows:

$$\begin{aligned} &\mathbb{E}_s\left[KL\left(\pi_\theta(a,c|s)||\pi_{\theta^{\mathsf{new}}}(a,c|s)\right)\right] \\ &= \mathbb{E}_s\left[KL\left(\pi^d_{\theta^{\mathsf{new}}_d}(a|s)||\pi^d_{\theta_d}(a|s)\right)\right. \\ &\quad \left. + KL\left(\pi^c_{\theta^{\mathsf{new}}_c}(c|s,a)||\pi^c_{\theta_c}(c|s,a)\right)\right] \\ &= \mathbb{E}_s\left[KL\left(\pi^d_{\theta^{\mathsf{new}}_d}(a|s)||\pi^d_{\theta_d}(a|s)\right)\right] \\ &\quad + \mathbb{E}_s \mathbb{E}_{a\sim\pi^d_{\theta^{\mathsf{new}}_d}(a|s)}\left[KL\left(\pi^c_{\theta^{\mathsf{new}}_c}(c|s,a)||\pi^c_{\theta_c}(c|s,a)\right)\right]. \end{aligned} \tag{36}$$

By using the analytical form of the discrete offloading action policy $\pi^d_{\theta^{\mathsf{new}}_d(a|s)}$ via its trajectory probability, we can further reduce the variance of the KL-divergence in the last term of (36) as

$$\begin{aligned} &\mathbb{E}_s\left[KL\left(\pi_\theta(a,c|s)||\pi_{\theta^{\mathsf{new}}}(a,c|s)\right)\right] \\ &\approx \mathbb{E}_s\left[KL\left(\pi^d_{\theta^{\mathsf{new}}_d}(a|s)||\pi^d_{\theta_d}(a|s)\right)\right] \\ &\quad + \mathbb{E}_s\left[\left(-\log(\pi^d_{\theta^{\mathsf{new}}_d(a|s)})\right) KL\left(\pi^c_{\theta^{\mathsf{new}}_c}(c|s,a)||\pi^c_{\theta_c}(c|s,a)\right)\right]. \end{aligned} \tag{37}$$

Based on the above approximation steps, the optimization of the original policy in (34) is transformed into a conjugate gradient form which allows for estimating the expectations of the policy objective in (35) with policy improvement guarantees. In this regard, the update direction can be approximated by $\nu \approx \hat{H}^{-1}\nabla_\theta J(\pi_\theta)$, where $\hat{H}^{-1}$ is the Hessian-vector product of sampled KL-divergence. Finally, the actor parameter is updated via backtracking line search [42] subject to the KL constraint as follows:

$$\theta_{t+1} = \theta_t + \gamma_t^a \sqrt{\frac{2\epsilon_{KL}}{\nu^\top \hat{H}\nu}}\nu, \tag{38}$$

where $\gamma_t^a \in (0,1)$ is the backtracking step-size parameter which controls the line search for guaranteeing the conjugate gradient improvement given the KL divergence constraint. It is desirable to set up a fairly large step size γ_t^a to initialize the line search space on the policy, and gradually shrink γ_t^a until a Armijo-Goldstein condition [42] is satisfied where a critical (optimal) point is obtained.

D. Critic Design

The role of the critic is to estimate the state-value function $V^{\pi_\theta}(s)$ to guide the update of the actor by approximating its state-value function. Specifically, a feature function $\phi : \mathcal{S} \mapsto \mathbb{R}^n$ is created as a full-ranked matrix with I dimensions to create i-dimensional features ($i \le I$) for any state $s \in \mathcal{S}$, i.e., $\phi(s) = \left(\phi^1(s), \ldots, \phi^i(s)\right)^\top$. Given a state s, the state-value function is thus approximated by a linear function $V_\omega(s) \approx \omega\phi(s)^\top$, where $\omega \in \mathbb{R}^m$ is a parameter vector used to update the state-value function. By function approximation, the critic provides an inexact temporal difference (TD) solution to the value function $V^{\pi_\theta}(s)$ under policy π_θ. This naturally results in the minimization of TD error as a loss function, i.e., the Mean Squared Projected Bellman Error (MSPBE) defined by

$$E_\theta(\omega) = \frac{1}{2}||V_\omega - \Lambda B_\theta V_\omega||^2, \tag{39}$$

where $\Lambda = \phi^\top(\phi M \phi^\top)^{-1}\phi H$ is the projector with $H \in \mathbb{R}^{|\mathcal{S}|\times|\mathcal{S}|}$ being a diagonal matrix whose elements are within the stationary state distribution d^{π_θ} generated according to the policy π_θ when the entire state space is irreducible. Also, B_θ denotes the Bellman operator implied by $B_\theta V(s) \leftarrow r(s,a) + \gamma P_\theta(s,s',a)V(s)$, where $V(s)$ is the state value, $r(s,a)$ is reward with discount $\gamma \in (0,1)$, and $P_\theta(s,s',a)$ is the transition probability as defined in IV-A. During the value function evaluation process, the critic aims to minimize the loss function in (39) to obtain a fixed point of the projected Bellman operator by gradient TD learning, where the Lipschitz continuity property of the MSPBE loss function in (39) is significant to guarantee a successful policy-parameter update. Note that the loss function in is quadratic and thus convex with respect to θ.

Lemma 1: Given the feature vector $\phi(s) = \left(\phi^1(s), \ldots, \phi^i(s)\right)^\top$, *the* $E_\theta(\omega)$*'s gradient is* ℓ*-Lipschitz with* $\ell = (1+\gamma)^2 \max_i ||\phi^i||_2^2$, *where* $i \in \mathcal{I}$.

Proof: See Appendix C in the technical report [29]. □

Algorithm 2 Training Procedure of Our A2C Algorithm

1: **Input:** Time budget T, discount factor γ, BFL environment env
2: **Output:** Optimal parameterized action policy π_θ^* and maximum reward R
3: **Initialization:** Initialize offloading policy parameters θ_d, parameterized allocation policy parameter θ_c, critic parameter ω, actor's learning rate γ^a, critic's learning rate γ^c, discount parameter γ, initial reward $R = 0$
4: **for** each episode **do**
5: Set up initial state s_0
6: **for** timestep $t = 1, 2, \ldots, T$ **do**
7: Perform action sampling using $\pi^d(.|s)$ and $\pi^c(.|s)$
8: **for** each MD $n \in \mathcal{N}$ **do**
9: Sample the discrete offloading action with policy $x_n \sim \pi^d_{n\theta_d}(.|s)$
10: Sample continuous allocation parameters for the selected offloading action x_n

$$c_n = \begin{cases} (p_n, b_{n,g}, \Psi_n) \sim \pi^c_{\theta_c}(.|s, x_n), & \text{if } x_{n,m,g} = 1 \\ (f^\ell_n, \Psi_n) \sim \pi^c_{\theta_c}(.|s, x_n), & \text{if } x_{n,m,g} = 0 \end{cases}$$

11: **end for**
12: Execute the parameterized action $a_t \leftarrow (x_{n,m,g,t}, c_{n,t}), \forall n \in \mathcal{N}$ in the BFL environment
13: Obtain reward r_t and next state s_{t+1}: $(r_t, s_{t+1}) \leftarrow env.step(a_t)$
14: Calculate the accumulated reward $R_{t+1} \leftarrow r_t + \gamma_t * R_t$
15: Compute the advantage function $A^{\pi_\theta}(s, a)$ using (29) based on r_t and $V^{\pi_\theta}(s)$
16: Estimate the policy gradient for the actor: $\nabla J(\pi_\theta) = \mathbb{E}_{s\sim d^{\pi_\theta}(.), a\sim\pi_\theta(.|s)} [A^{\pi_\theta}(s, a) \nabla_\pi \log \pi_\theta(a|s)]$
17: Optimize the actor policy via TRPO in (35) with computed advantage value and update its parameter with KL constraint in (38): $\theta_{t+1} = \theta_t + \gamma^a_t \sqrt{\frac{2\epsilon_{KL}}{\mathbf{v}^\top H \mathbf{v}}} \mathbf{v}$
18: Calculate the MSPBE loss for critic via (39)
19: Optimize V_ω for the critic and update its policy in (40) via TD error σ_t: $\omega_{t+1} = \omega_t + \gamma^c_t A^{\pi_\theta}(s_t, a_t).\nabla_\omega V_\omega(s_t)$
20: **end for**
21: **end for**

Based on the stochastic gradient descent on $E_\theta(\omega)$ with respect to parameter ω, the critic update can be achieved by

$$\omega_{t+1} = \omega_t + \gamma^c_t A^{\pi_\theta}(s_t, a_t) \nabla_\omega V_\omega(s_t), \tag{40}$$

where $\gamma^c_t \in (0, 1)$ is a step-size parameter.

E. Training Procedure of Parametrized A2C Algorithm in BFL

The training procedure of our A2C algorithm is summarized in Algorithm 2. To prepare for the training, we build a BFL environment where multiple MDs participate in the FL training and block mining with a set of ESs connected via wireless links. The objective function in the system utility optimization problem built in (32) is selected as a system DRL reward that is obtained via iterative training with parameterized actions and system states for an optimal offloading policy π_θ^* to maximize the reward R in the long run with a initial system state (line 5). We build an actor that consists of a deep neural network (DNN) for offloading decision sampling and another DNN for parameter sampling. At each timestep, we randomly sample a set of discrete offloading actions for all MDs via a DNN-empowered policy $\pi^d(.|s)$. Then we also sample a set of continuous allocation parameters based on the sampled offloading decision using another policy $\pi^c_{\theta_c}(.|s, x_n)$ as defined in IV-A (lines 7-11). With the result of the action sampling step, we generate a complete set of parameterized actions for all MDs which is ready to be executed in the BFL environment (line 12). This allows us to obtain the reward, i.e., system utility, to calculate the long-term return and the agent moves to the next state needed for the following step of training (lines 13-14). Then, the actor and critic updates their policy (lines 15-19): (i) the actor computes the policy gradient via its advantage function and TRPO to optimize its policy, and (ii) the critic computes the MSPBE loss function and updates its gradient via TD error learning.

TABLE IV
SIMULATION PARAMETERS

Parameter	Value
Number of ESs M	5
Number of MDs N	[20-100]
Number of sub-channels at each ES K	5
Data size D_n	[0.5-2] MB
CPU workload of MDs and ESs	[0.7-1.1] Gcyles
MDs' transmit power p_n	[10-30] dBm
MD's computational capability f^ℓ_n	[0,2-2] GHz
ES's computational capability f_m	5 GHz
Background noise variance	-100 dBm
Maximum system bandwidth W	20 MHz
MD's model size ϑ	5 KB
MD's energy coefficient κ	$5 * 10^{-27}$
MD's hash rate Ψ_n	[100-1000] GHash/s
MD's mining power efficiency Ξ_n	$5 * 10^{-8}$ J/hash
Hash amount of a block $\hbar$	50 GHash
Block broadcasting rate ξ	0.005
Number of forking occurrences ζ	3
MD's maximum latency τ_n	3 sec
Noise power spectral density N_0	-174 dBm/Hz
ES's bandwidth $b_{m'}$	5 MHz
ES's transmit power $p_{m'}$	[100-120] dBm

V. SIMULATIONS AND PERFORMANCE EVALUATION

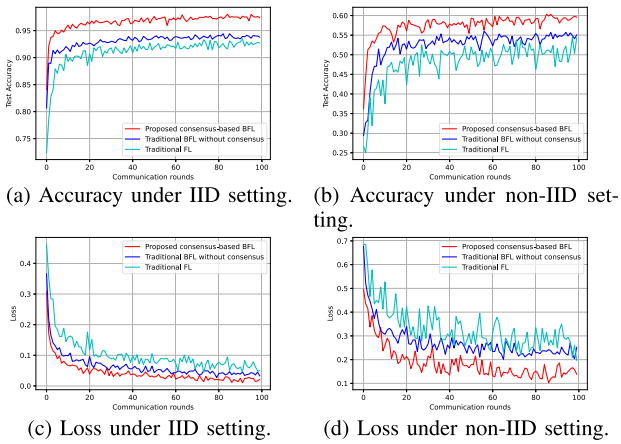
A. Parameter Settings

We conduct numerical experiments to verify our method under various parameter settings. Inspired by related works [12], [14], [15], [24], [30], [36], we set up all necessary parameters for our BFL environment as listed in Table IV.

We consider a BFL system with 3 ESs and 10 MDs which aim to collaboratively train two popular image datasets: SVHN[1] (including 73,257 training instances and testing 26,032 instances) and Fashion-MNIST[2] (including 60,000 training instances and testing 10,000 instances), where each dataset contains 10 labels/classes. For the SVHN dataset, we deploy a convolutional neural network (CNN) with two 2-D convolutional layers followed by two hidden layers (the first with 256 units and the second with 72 units) with ReLU activation. The CNN architecture used for the Fashion-MNIST dataset is similar, with the first hidden layer with 320 units and the second hidden layer with 50 units. We investigate the FL performance under both IID and non-IID data settings. In IID data setting, each MD possesses datapoints from all the 10 labels, while in non-IID data setting, each MD contains data samples from three of 10 labels for the SVHN dataset and two of 10 labels for the Fashion-MNIST dataset. We employ the Adam optimizer with mini-batch size of 25 and 10 SGD iterations.

In our A2C algorithm, the actor has two DNNs, one for the discrete offloading policy with two hidden neural layers {64 and 32 in sizes} and one for the continuous allocation

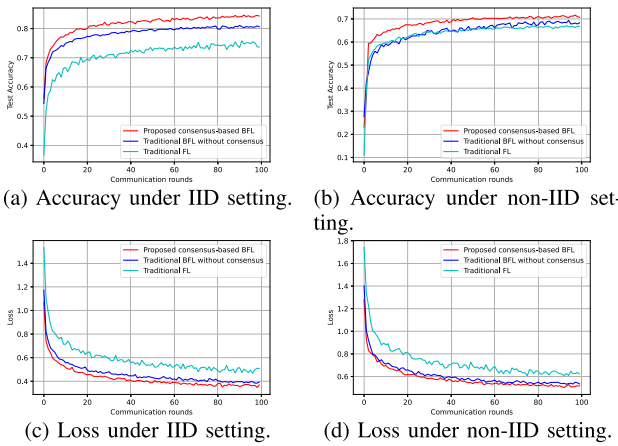
[1]http://ufldl.stanford.edu/housenumbers/
[2]https://github.com/zalandoresearch/fashion-mnist



(a) Accuracy under IID setting. (b) Accuracy under non-IID setting.

(c) Loss under IID setting. (d) Loss under non-IID setting.

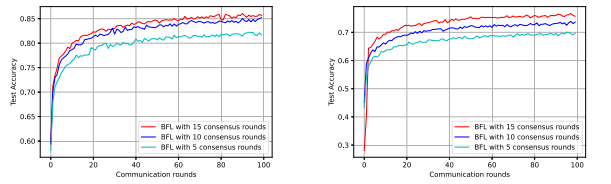
Fig. 4. Comparison of different FL approaches on SVHN dataset.



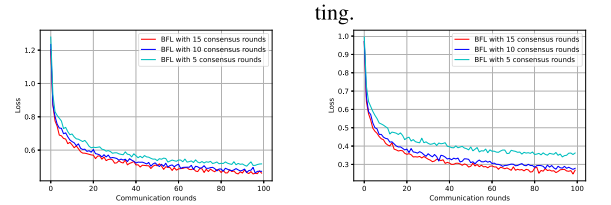
(a) Accuracy under IID setting. (b) Accuracy under non-IID setting.

(c) Loss under IID setting. (d) Loss under non-IID setting.

Fig. 5. Comparison of different FL approaches on Fashion-MNIST dataset.



(a) Accuracy under IID setting. (b) Accuracy under non-IID setting.



(c) Loss under IID setting. (d) Loss under non-IID setting.

Fig. 6. Comparison of BFL performance with different consensus rounds on Fashion-MNIST dataset.

parameter policy with two layers {128 and 64 in sizes}. For the output layers, we used Softmax to generate offloading decisions for MDs and adopted Tanh to produce allocation parameters given the discrete offloading policy. The KL divergence constraint ϵ_{KL} is set to 0.01 for the TRPO-based actor policy optimizer [41]. The critic was also built by a DNN that contains two hidden layers of sizes {200, 100} to train the state-value function of our A2C scheme with the Adam optimizer. To implement our parameterized A2C algorithm, we set up a virtual agent that is allowed to interact with a pre-defined BFL environment to learn the parameterized offloading policy for all MDs and observe the return, i.e., system utility, after each iteration. We trained the agent over 10000 episodes with 100 timesteps per each episode. All numerical results are averaged over 10 independent simulation runs.

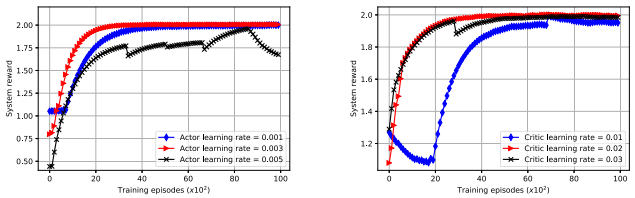
B. Evaluation of FL Performance

We evaluate the classification accuracy and loss performance of our proposed BFL scheme (with 5 consensus rounds) and compare it with two related schemes. The first one is a traditional FL scheme [1], where ESs work independently and each only aggregates the models of its associated MDs and then broadcasts the resulting model across devices. The second one is a traditional BFL without P2P consensus [4], in which each ES runs an averaging algorithm by collaborating with its MDs to build its aggregated model, and then a random ES is selected as a leader that builds a global model based on its local aggregated model without conducting P2P consensus. Fig. 4 illustrates the performance when training the SVHN dataset, showing the considerable improvements in terms of higher accuracy and lower loss compared with the counterparts. Although the performance degrades when the dataset becomes non-IID, our consensus-based BFL scheme still outperforms other algorithms. The BFL scheme without consensus achieves a better training performance than the traditional FL scheme since its randomized leader ES selection avoid local model bias across the clients. Moreover, the performance gap between our scheme and the others becomes larger in the non-IID case which demonstrates the benefit of consensus-based model aggregation over existing approaches. The advantages of our scheme are also verified on the Fashion-MNIST dataset in both IID and non-IID settings, as indicated in Fig. 5. For example, in the IID setting, our consensus-based BFL scheme improves the accuracy rate by 8% and 14% in comparison with the traditional BFL scheme without consensus and the traditional FL scheme, respectively.

Fig. 6 investigates the impact of P2P consensus rounds (i.e., 5, 10, and 15 rounds) on the learning performance. We can see that the increase of consensus rounds significantly improves the performance. Under the non-IID setting, the role of consensus on the model training becomes more significant with a larger performance gap, for example, between 5 rounds and 15 rounds, which shows the efficiency of our consensus-based BFL design for federated model training.

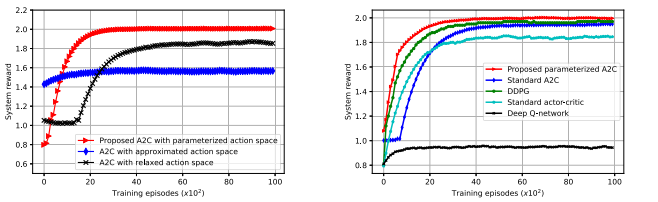
C. Evaluation of DRL Training Performance

We first investigate the training system reward (i.e., system utility as defined in (32)) performance by changing the learning rates at both actor and critic, which is important to determine a reliable parameter set for our later simulations, as shown in Fig. 7. If the learning rate is too small, the policy training probably requires long time to achieve an optimal solution. However, if we set a large learning rate, the training may become unstable and possibly diverge. In A2C algorithms, since the actor updates slower than the critic with a timestep, we set up the actor with a learning rate smaller than the critic's learning rate. We first evaluate the impacts of the actor learning rate, i.e., the backtracking step-size parameter γ_t^a which controls the policy update in the trust region in each iteration of TRPO under the KL divergence constraint.



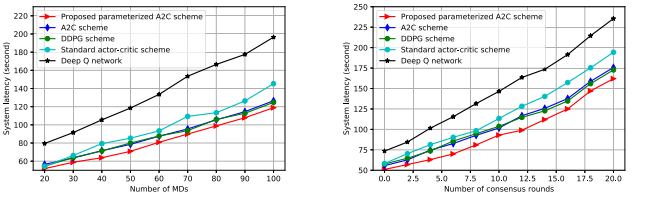
(a) System rewards with different actor learning rates. (b) System rewards with different critic learning rates.

Fig. 7. Evaluation of the training performance.



(a) System reward with different action spaces. (b) System reward with different learning schemes.

Fig. 8. Comparison of system reward via learning curves.



(a) System latency with different numbers of MDs. (b) System latency versus consensus rounds.

Fig. 9. Comparison of system latency between different schemes .

Fig. 7(a) reveals that the learning rate of 0.003 exhibits the highest system reward with the fastest convergence, compared to the case of $\gamma_t^a = 0.001$ which shows the slowest convergence rate. However, when the learning rate is relatively high ($\gamma_t^a = 0.005$), the learning process becomes unstable and diverges. Similar to the actor part, we also set up a learning procedure by considering various critic learning rates. As indicated in Fig. 7(b), the learning rate $\gamma_t^c = 0.02$ has the most stable training performance with a quick convergence, compared to other learning settings. Thus, we will use learning rates $\gamma_t^a = 0.003$ and $\gamma_t^c = 0.02$ for the following simulations.

Next, we compare the reward performance obtained by our proposed parameterized A2C algorithm against two baseline schemes: A2C with relaxed action space [36] and A2C with approximated action space [37]. *For the first baseline scheme*, we first relax the discrete offloading vector into a continuous set by defining a relaxed space as $\mathcal{A}_a = \{f(x_1,), f(x_2), \ldots, f(x_N)\}$, where $f(.)$ is a probability softmax function, and then re-normalize them to approximate discrete offloading vectors for execution. Compared with our proposed parameterized scheme, this method significantly increases the sampling complexity on the joint action space. *For the second baseline scheme*, we discretize each of continuous allocation vectors into a discrete subset. For example, the transmit power variable of MD n is discretized into Z levels as $\boldsymbol{P}_n = \left[0, p_{min}, p_{min}.\left(\frac{p_{max}}{p_{min}}\right)^{\frac{1}{|Z|-2}}, \ldots, p_{max}\right]$, where $p_n = 0$ implies the local execution mode. This discretization process creates a large number of quantization levels for convenient action sampling but also results in quantization noise.

From Fig. 8(a), our proposed scheme can achieve the best reward performance compared with baseline approaches, thanks to a flexible parameterized action sampling solution where both discrete offloading decisions and allocation variables are directly trained without relaxation or approximation. Meanwhile, the A2C scheme with relaxed action space suffers a reward decrease with high training variance since the offloading action selection must be converted into a continuous space of allocation vectors, leading to an extremely high complexity in the action sampling and thus making the training inefficient. Moreover, its complex action sampling requires longer time to reach convergence, i.e., after 4000 episodes compared to 2500 episodes in our proposed scheme. The lowest reward gain is observed at the approximated scheme, where the approximation of action space introduces the quantization error which degrades the policy training.

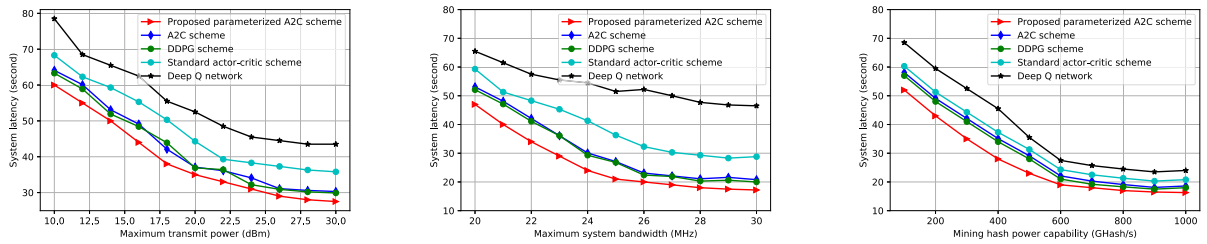
We then compare our scheme with state-of-the-art DRL approaches: (i) A2C [12], as in our method but only using the vanilla gradient update at the actor without policy optimization improvement; (ii) deep deterministic policy gradient (DDPG), using actor-critic with deterministic policy training over the continuous action space [36]; (iii) standard actor-critic [28], i.e., without using advantage function in the policy estimation; and (iv) deep Q-network [22], which uses action sampling approximation. From Fig. 8(b), we find that our proposed parameterized A2C method is better than baselines in terms of system reward and convergence rate and stability. This is due to the efficient action sampling and improved policy optimization based on trust region enforcement which helps avoiding possible gradient divergence. Thus, a faster and more reliable policy search is achieved toward an optimal solution. The DDPG scheme has a lower reward convergence speed due to the relaxation of the discrete offloading action space. This increases action sampling complexity and thus makes the policy training less efficient compared with our parameterized A2C.

D. Evaluation of System Latency Performance

In this subsection, we evaluate the system latency performance under networking scenarios. Based on the system reward (utility) computed via the above training process, it is straightforward to calculate the system latency via their mathematical relation as mentioned in (31) and (32).

We investigate the latency performance when varying the numbers of MDs from 20 to 100 in Fig. 9(a). The system latency obtained by our proposed parameterized A2C scheme is the lowest across the considered methods. As expected, the latency of each method increases with the number of MDs, due to a higher offloading and mining latency caused by a higher competition on bandwidth and hash allocation among MDs, our proposed scheme still achieves the best latency performance when the number of MDs increases. For instance, with 100 MDs, the system latency of our scheme is 11%, 13%, 25% and 38% lower than that of the DDPG, A2C, actor-critic, and deep Q-network schemes, respectively.

We then investigate the impact of P2P rounds on the system latency, as indicated in Fig. 9(b). We change the number of



(a) System latency versus transmit power allocation.

(b) System latency versus bandwidth allocation.

(c) System latency versus mining hash allocation.

Fig. 10. Comparison of system latency with different resource allocation settings.

P2P rounds from 0 to 20, where 0 implies the traditional BFL scheme without consensus. Although the use of consensus procedure at the edge layer enhances the model training performance, it potentially increases the system latency due to the additional delay of P2P model aggregation among ESs. However, with our advanced DRL design, the proposed parameterized A2C scheme achieves the minimal latency compared with other baselines. The increase of P2P rounds requires more time for model aggregation, leading to a higher system latency, but our approach still has the best performance.

We also investigate the latency performance under different resource allocation scenarios in Fig. 10 in a BFL environment with 5 ESs and 30 MDs. We first vary the maximum transmit power P_n at each MD n between 10 and 30 dBm and then measure the latency of offloading and mining. As shown in Fig. 10(a), the latency of all schemes decreases with respect to the increasing amount of allocated transmit power. This is because a larger transmit power improves the data transmission rate that helps reduce the data offloading latency. Notably, compared with other schemes, our method achieves the most significant and stable latency decrease due to our efficient and robust parameterized policy training to obtain the better offloading trajectory.

Moreover, we investigate the latency trends of different algorithms when the maximum system bandwidth W varies from 20 to 30 MHz in Fig. 10(b). Similar to the power allocation scenario, more bandwidth would help mitigate the offloading latency. In our simulation setting, the impact of bandwidth on the system latency is significant before the threshold of 25 MHz, where our scheme shows the best latency savings. Compared with DDPG, A2C and actor-critic schemes, our parameterized A2C scheme can reduce the latency by 7%, 8% and 17%, respectively, and achieves a 60% lower latency compared with the deep Q-network baseline.

Finally, we compare the latency performance with respect to the changes of mining hash power capability $\Psi_{\max}$. By increasing the hash budget, each MD has a higher chance to obtain sufficient hash power to run the block mining. As can be seen in Fig. 10(c), our proposed scheme outperforms other baselines in terms of system latency in each hash allocation setting. The simulation results thus demonstrate the efficiency of our proposed A2C algorithm design in the system latency minimization.

E. Attack Evaluation

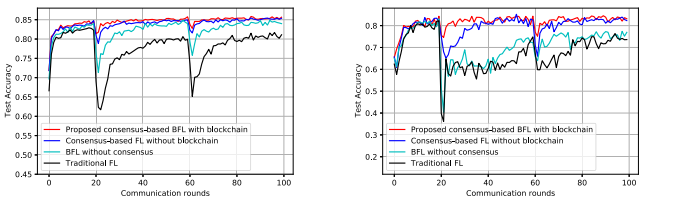
1) Attack Model and Security Analysis With Blockchain: A major concern in conventional FL with a single centralized aggregation server is that the server may become compromised via model poisoning attacks launched by adversaries. Model poisoning attacks mainly consist of untargeted model poisoning attacks and targeted model poisoning attacks [43]. The former category aims to degrade the accuracy of the global model training, whereas the latter aims to control the model deviation towards their target. Here, we focus on untargeted model poisoning attacks on our BFL system.

We assume that at a certain global FL round k, an ES can be compromised by a model poisoning attack following two steps. First, the attacker injects certain random noise $v^{(k)}$ to the aggregated global model $\boldsymbol{w}^{(k)}$ obtained via 12 to manipulate the global model update $\boldsymbol{w}^{(k)} \leftarrow \boldsymbol{w}^{(k)} + v^{(k)}$. Here we adopt Gaussian noise, i.e., $v^{(k)} = -\varpi \boldsymbol{q}^{(k)}$, where ϖ is a scaling factor which characterizes the magnitude of the compromised model, and $\boldsymbol{q}^{(k)}$ is a random vector sampled from the Gaussian distribution $\mathcal{N}(\boldsymbol{0}, \boldsymbol{I})$. Next, the attacked ES broadcasts the compromised global model to local MDs.

In our BFL system, blockchain replaces the centralized authority in FL with a decentralized tamper-proof data ledger to monitor the model consensus process as well as mitigate single-point-of-failure. By deploying a blockchain over the edge layer, any model update event over consensus rounds at a certain ES is automatically traced by other ESs. If a poisoning attack takes place at an ES, other ESs can detect this behavior via transaction logs. Here, we adopt the attack detection score metric [44], which characterizes the abnormal gradient deviation caused by the poisoning attacker at a certain FL round, to detect the occurrence of a model attack at a certain ES.

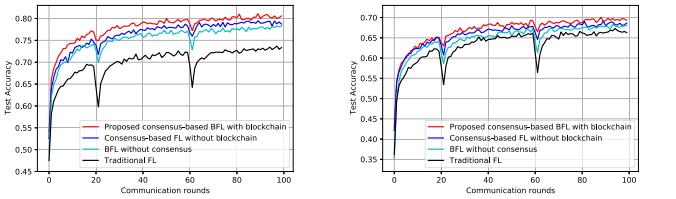
2) Attack Simulation: We investigate the training performance of different FL methods under model poisoning attacks on the two considered datasets. We consider an attack scenario where an adversary compromises an ES and deploys model poisoning attacks by injecting random noise at the global rounds of 20 and 60 during the model consensus process among ESs. Our proposed consensus-based BFL scheme with blockchain is compared with other three baselines: consensus-based FL without blockchain, BFL without consensus, and traditional FL. For the consensus-based schemes, we consider that there are 5 consensus rounds and the attacker poisons the model at round 3. For the BFL scheme without consensus, the attacker poisons one of the ESs during the global model aggregation process. For the traditional FL scheme, the attacker poisons the centralized aggregator.

As illustrated in Fig. 11 for the SVHN dataset, the accuracy performance of all schemes is dropped when the attack occurs (i.e., at aggregation rounds 20 and 60). However, our



(a) Accuracy under IID setting. (b) Accuracy under non-IID setting.

Fig. 11. Comparison of different FL approaches under model poisoning attacks (SVHN dataset) .



(a) Accuracy under IID setting. (b) Accuracy under non-IID setting.

Fig. 12. Comparison of different FL approaches under model poisoning attacks (Fashion-MNIST dataset).

consensus-based BFL scheme achieves the best accuracy rate and highest robustness against the poisoning attack in both IID and non-IID data distribution settings. For instance, from Fig. 11(a), we see that the accuracy of our scheme at the onset of an attack only drops by 3.7%, as opposed to 7.6%, 16.9%, and 32.1% in the consensus-based FL without blockchain, BFL without consensus, and traditional FL schemes, respectively. The traceable data ledger of blockchain records all the global updating behaviors of ESs across consensus rounds, which allows blockchain to rapidly detect the poisoning attack. Since all ESs are connected on the shared ledger, the blockchain disregards this ES from the consensus process to protect model training. The consensus-based FL scheme without blockchain has the second highest performance. The poisoned model is involved in the consensus process, which degrades the overall model aggregation, though the impact is dampened since the poisoned model is only one participating in the consensus process. The other two non-consensus-based methods show lower accuracy and less robustness against the attack. The BFL scheme without consensus shows lower training robustness since the attack at one of the ESs significantly affects the global model aggregation at that particular server and its associated MDs. In the traditional FL scheme with a centralized server, the model training is degraded after the attack since the attacker directly poisons the only model aggregator. We repeat these experiments on the Fashion-MNIST dataset in Fig. 12 which indicates qualitatively similar results.

VI. Conclusion

This paper studied a decentralized BFL system in multi-server edge computing with a holistic design of both offloading-assisted ML model training and mobile block mining schemes. A model aggregation solution has been developed via P2P-based consensus among ESs to build a global model that is shared with MDs via Blockchain for reliable model learning empowered by block mining. We aimed to minimize the system latency by a parameterized A2C algorithm with a careful design of actor and critic. A comprehensive analysis of the convergence properties of our proposed BFL model was given. Numerical simulations verified the superior performance of our proposed consensus-based BFL scheme over state-of-the-art schemes in terms of higher accuracy and lower loss. The proposed parameterized A2C algorithm exhibited the faster convergence rate and lower system latency, compared with the existing DRL schemes. Our blockchain-empowered BFL scheme also achieved high robustness against model poisoning attacks.

References

[1] S. Wang et al., "Adaptive federated learning in resource constrained edge computing systems," *IEEE J. Sel. Areas Commun.*, vol. 37, no. 6, pp. 1205–1221, Jun. 2019.

[2] D. C. Nguyen, M. Ding, P. N. Pathirana, A. Seneviratne, J. Li, and H. V. Poor, "Federated learning for Internet of Things: A comprehensive survey," *IEEE Commun. Surveys Tuts.*, vol. 23, no. 3, pp. 1622–1658, 3rd Quart., 2021.

[3] C. Ma et al., "On safeguarding privacy and security in the framework of federated learning," *IEEE Netw.*, vol. 34, no. 4, pp. 242–248, Jul./Aug. 2020.

[4] F. Ayaz, Z. Sheng, D. Tian, and Y. L. Guan, "A blockchain based federated learning for message dissemination in vehicular networks," *IEEE Trans. Veh. Technol.*, vol. 71, no. 2, pp. 1927–1940, Feb. 2022.

[5] V. Mothukuri, R. M. Parizi, S. Pouriyeh, A. Dehghantanha, and K.-K.-R. Choo, "FabricFL: Blockchain-in-the-loop federated learning for trusted decentralized systems," *IEEE Syst. J.*, vol. 16, no. 3, pp. 3711–3722, Sep. 2022.

[6] Y. He, K. Huang, G. Zhang, F. R. Yu, J. Chen, and J. Li, "Bift: A blockchain-based federated learning system for connected and autonomous vehicles," *IEEE Internet Things J.*, vol. 9, no. 14, pp. 12311–12322, Jul. 2022.

[7] Q. Hu, Z. Wang, M. Xu, and X. Cheng, "Blockchain and federated edge learning for privacy-preserving mobile crowdsensing," *IEEE Internet Things J.*, early access, Nov. 16, 2021, doi: 10.1109/JIOT.2021.3128155.

[8] Y. Zhao et al., "Privacy-preserving blockchain-based federated learning for IoT devices," *IEEE Internet Things J.*, vol. 8, no. 3, pp. 1817–1829, Feb. 2021.

[9] D. C. Nguyen et al., "Federated learning meets blockchain in edge computing: Opportunities and challenges," *IEEE Internet Things J.*, vol. 8, no. 16, pp. 12806–12825, Aug. 2021.

[10] W. Shi, S. Zhou, Z. Niu, M. Jiang, and L. Geng, "Joint device scheduling and resource allocation for latency constrained wireless federated learning," *IEEE Trans. Wireless Commun.*, vol. 20, no. 1, pp. 453–467, Jan. 2021.

[11] Q. Ma, Y. Xu, H. Xu, Z. Jiang, L. Huang, and H. Huang, "FedSA: A semi-asynchronous federated learning mechanism in heterogeneous edge computing," *IEEE J. Sel. Areas Commun.*, vol. 39, no. 12, pp. 3654–3672, Dec. 2021.

[12] H. Yang, J. Zhao, Z. Xiong, K.-Y. Lam, S. Sun, and L. Xiao, "Privacy-preserving federated learning for UAV-enabled networks: Learning-based joint scheduling and resource management," *IEEE J. Sel. Areas Commun.*, vol. 39, no. 10, pp. 3144–3159, Oct. 2021.

[13] C. W. Zaw, S. R. Pandey, K. Kim, and C. S. Hong, "Energy-aware resource management for federated learning in multi-access edge computing systems," *IEEE Access*, vol. 9, pp. 34938–34950, 2021.

[14] Z. Ji, L. Chen, N. Zhao, Y. Chen, G. Wei, and F. R. Yu, "Computation offloading for edge-assisted federated learning," *IEEE Trans. Veh. Technol.*, vol. 70, no. 9, pp. 9330–9344, Sep. 2021.

[15] Z. Yang, M. Chen, W. Saad, C. S. Hong, and M. Shikh-Bahaei, "Energy efficient federated learning over wireless communication networks," *IEEE Trans. Wireless Commun.*, vol. 20, no. 3, pp. 1935–1949, Mar. 2021.

[16] C. Hu, J. Jiang, and Z. Wang, "Decentralized federated learning: A segmented gossip approach," 2019, *arXiv:1908.07782*.

[17] H. Xing, O. Simeone, and S. Bi, "Decentralized federated learning via SGD over wireless D2D networks," in *Proc. IEEE 21st Int. Workshop Signal Process. Adv. Wireless Commun. (SPAWC)*, May 2020, pp. 1–5.

[18] F. P.-C. Lin, S. Hosseinalipour, S. S. Azam, C. G. Brinton, and N. Michelusi, "Semi-decentralized federated learning with cooperative D2D local model aggregations," *IEEE J. Sel. Areas Commun.*, vol. 39, no. 12, pp. 3851–3869, Dec. 2021.

[19] S. Hosseinalipour et al., "Multi-stage hybrid federated learning over large-scale D2D-enabled fog networks," *IEEE/ACM Trans. Netw.*, vol. 30, no. 4, pp. 1569–1584, Aug. 2022.
[20] H. Kim, J. Park, M. Bennis, and S.-L. Kim, "Blockchained on-device federated learning," *IEEE Commun. Lett.*, vol. 24, no. 6, pp. 1279–1283, Jun. 2020.
[21] S. R. Pokhrel and J. Choi, "Federated learning with blockchain for autonomous vehicles: Analysis and design challenges," *IEEE Trans. Commun.*, vol. 68, no. 8, pp. 4734–4746, Aug. 2020.
[22] Y. Lu, X. Huang, K. Zhang, S. Maharjan, and Y. Zhang, "Communication-efficient federated learning and permissioned blockchain for digital twin edge networks," *IEEE Internet Things J.*, vol. 8, no. 4, pp. 2276–2288, Feb. 2021.
[23] M. Aloqaily, I. A. Ridhawi, and M. Guizani, "Energy-aware blockchain and federated learning-supported vehicular networks," *IEEE Trans. Intell. Transp. Syst.*, early access, Aug. 17, 2021, doi: 10.1109/TITS.2021.3103645.
[24] X. Deng et al., "On dynamic resource allocation for blockchain assisted federated learning over wireless channels," 2021, *arXiv:2105.14708*.
[25] B. Ganguly et al., "Multi-edge server-assisted dynamic federated learning with an optimized floating aggregation point," 2022, *arXiv:2203.13950*.
[26] S. Hosseinalipour et al., "Parallel successive learning for dynamic distributed model training over heterogeneous wireless networks," 2022, *arXiv:2202.02947*.
[27] X. Xiong, K. Zheng, L. Lei, and L. Hou, "Resource allocation based on deep reinforcement learning in IoT edge computing," *IEEE J. Sel. Areas Commun.*, vol. 38, no. 6, pp. 1133–1146, Jun. 2020.
[28] S. Chen, Z. Yao, X. Jiang, J. Yang, and L. Hanzo, "Multi-agent deep reinforcement learning-based cooperative edge caching for ultra-dense next-generation networks," *IEEE Trans. Commun.*, vol. 69, no. 4, pp. 2441–2456, Apr. 2021.
[29] D. C. Nguyen, S. Hosseinalipour, D. J. Love, P. N. Pathirana, andC. G. Brinton, "Latency optimization for blockchain-empowered federated learning in multi-server edge computing," 2022, *arXiv:2203.09670*.
[30] D. C. Nguyen, P. N. Pathirana, M. Ding, and A. Seneviratne, "Privacy-preserved task offloading in mobile blockchain with deep reinforcement learning," *IEEE Trans. Netw. Service Manag.*, vol. 17, no. 4, pp. 2536–2549, Dec. 2020.
[31] L. Xiao and S. Boyd, "Fast linear iterations for distributed averaging," *Syst. Control Lett.*, vol. 53, no. 1, pp. 65–78, 2004.
[32] L. Xiao, S. Boyd, and S.-J. Kim, "Distributed average consensus with least-mean-square deviation," *J. Parallel Distrib. Comput.*, vol. 67, no. 1, pp. 33–46, Jan. 2007.
[33] C. T. Dinh et al., "Federated learning over wireless networks: Convergence analysis and resource allocation," *IEEE/ACM Trans. Netw.*, vol. 29, no. 1, pp. 398–409, Feb. 2021.
[34] J. Wang, Q. Liu, H. Liang, G. Joshi, and H. V. Poor, "Tackling the objective inconsistency problem in heterogeneous federated optimization," in *Proc. Adv. Neural Infor. Process. Syst.*, vol. 33, 2020, pp. 7611–7623.
[35] C. Huang et al., "ZkRep: A privacy-preserving scheme for reputation-based blockchain system," *IEEE Internet Things J.*, vol. 9, no. 6, pp. 4330–4342, Mar. 2022.
[36] D. Nguyen, M. Ding, P. Pathirana, A. Seneviratne, J. Li, and V. Poor, "Cooperative task offloading and block mining in blockchain-based edge computing with multi-agent deep reinforcement learning," *IEEE Trans. Mobile Comput.*, early access, Oct. 14, 2021, doi: 10.1109/TMC.2021.3120050.
[37] F. Meng, P. Chen, L. Wu, and J. Cheng, "Power allocation in multi-user cellular networks: Deep reinforcement learning approaches," *IEEE Trans. Wireless Commun.*, vol. 19, no. 10, pp. 6255–6267, Oct. 2020.
[38] H. Lu, X. He, M. Du, X. Ruan, Y. Sun, and K. Wang, "Edge QoE: Computation offloading with deep reinforcement learning for Internet of Things," *IEEE Internet Things J.*, vol. 7, no. 10, pp. 9255–9265, Oct. 2020.
[39] M. Akbari, M. R. Abedi, R. Joda, M. Pourghasemian, N. Mokari, and M. Erol-Kantarci, "Age of information aware VNF scheduling in industrial IoT using deep reinforcement learning," *IEEE J. Sel. Areas Commun.*, vol. 39, no. 8, pp. 2487–2500, Aug. 2021.
[40] S. Qiu, Z. Yang, J. Ye, and Z. Wang, "On finite-time convergence of actor-critic algorithm," *IEEE J. Sel. Areas Inf. Theory*, vol. 2, no. 2, pp. 652–664, Jun. 2021.
[41] L. Shani et al., "Adaptive trust region policy optimization: Global convergence and faster rates for regularized MDPS," in *Proc. AAAI Conf. Artif. Intell.*, 2020, vol. 34, no. 4, pp. 5668–5675.
[42] S. Vaswani et al., "Painless stochastic gradient: Interpolation, line-search, and convergence rates," in *Proc. Adv. Neural Infor. Process. Syst.*, vol. 32, 2019, pp. 3732–3745.
[43] M. Fang et al., "Local model poisoning attacks to Byzantine-robust federated learning," in *Proc. 29th USENIX Secur. Symp.*, 2020, pp. 1605–1622.
[44] A. Mondal, H. Virk, and D. Gupta, "BEAS: Blockchain enabled asynchronous & secure federated machine learning," in *Proc. 3rd AAAI WRKSH Privacy-Preserving Artif. Intel.*, 2022, pp. 1–15.

Dinh C. Nguyen (Member, IEEE) received the Ph.D. degree from Deakin University, Australia, in 2022. He is currently a Post-Doctoral Associate at the School of Electrical and Computer Engineering, Purdue University, USA. His research interests include federated learning, blockchain, and edge computing. He is an Associate Editor of the IEEE OPEN JOURNAL OF THE COMMUNICATIONS SOCIETY.

Seyyedali Hosseinalipour (Member, IEEE) received the B.S. degree (Hons.) in electrical engineering from the Amirkabir University of Technology, Tehran, Iran, in 2015, and the M.S. and Ph.D. degrees in electrical engineering from North Carolina State University, Raleigh, NC, USA, in 2017 and 2020, respectively. He was a Post-Doctoral Researcher at Purdue University, West Lafayette, IN, USA, from 2020 to 2022. He is currently an Assistant Professor at the Department of Electrical Engineering, University at Buffalo (SUNY), Getzville, NY, USA. His research interests include analysis of modern wireless networks, synergies between machine learning methods and fog computing systems, distributed machine learning, and network optimization. He was a recipient of the ECE Doctoral Scholar of the Year Award in 2020 and the ECE Distinguished Dissertation Award from North Carolina State University in 2021. He has served as the TPC Co-Chair and a TPC Member for numerous artificial intelligence- and machine learning-related workshops held in conjunction with IEEE INFOCOM 2021, IEEE GLOBECOM 2021, IEEE ICC 2021, and a Program Committee Member for IEEE MSN 2021–2022.

David J. Love (Fellow, IEEE) received the B.S. (Hons.), M.S.E., and Ph.D. degrees in electrical engineering from The University of Texas at Austin in 2000, 2002, and 2004, respectively.

Since 2004, he has been with the School of Electrical and Computer Engineering, Purdue University, where he is now the Nick Trbovich Professor of electrical and computer engineering. He holds 31 issued U.S. patents. Along with his coauthors, he won Best Paper Awards from the IEEE Communications Society (2016 IEEE Communications Society Stephen O. Rice Prize), the IEEE Signal Processing Society (2015 IEEE Signal Processing Society Best Paper Award), and the IEEE Vehicular Technology Society (2009 IEEE Transactions on Vehicular Technology Jack Neubauer Memorial Award). He currently serves as a Senior Editor for *IEEE Signal Processing Magazine* and previously served as an Editor for the IEEE TRANSACTIONS ON COMMUNICATIONS, an Associate Editor for the IEEE TRANSACTIONS ON SIGNAL PROCESSING, and a Guest Editor for special issues of the IEEE JOURNAL ON SELECTED AREAS IN COMMUNICATIONS and the *EURASIP Journal on Wireless Communications and Networking*.

Pubudu N. Pathirana (Senior Member, IEEE) was a Visiting Professor at Yale University in 2009. He is currently a Full Professor and the Director of the Networked Sensing and Control Group, School of Engineering, Deakin University, Geelong, Australia. His current research interests include bio-medical assistive device design, mobile/wireless networks, and rehabilitation robotics.

Christopher G. Brinton (Senior Member, IEEE) received the M.S. and Ph.D. (Hons.) degrees from Princeton University in 2013 and 2016, respectively, both in electrical engineering. He is currently an Assistant Professor of ECE at Purdue University. His research interest is at the intersection of machine learning and networked systems, specifically in distributed machine learning, fog/edge network intelligence, and data-driven network optimization. He was a recipient of the NSF CAREER Award, the ONR Young Investigator Program (YIP) Award, and the DARPA Young Faculty Award (YFA). He currently serves as an Associate Editor for IEEE TRANSACTIONS ON WIRELESS COMMUNICATIONS in the ML and AI for wireless area.