A Decentralized Framework for Learning Mobile Spectrum Data on Large Scales

Anonymous Author(s)

ABSTRACT

Crowdsourcing via mobile sensors has been an important paradigm for spectrum sensing on large scales; however, it usually requires participants to upload their collected data to a centralized cloud for further analysis, which introduces two deficiencies that limit its adoption. First, it needs to upload the spectrum data along with the meta-data, e.g. GPS info, which is sensitive for individuals in many cases so that they are reluctant to join the community. Second, a centralized cloud owned by a single entity is relatively difficult to avoid single point of failures and makes sharing the findings, e.g., learned models based on the data, unnecessary hard. In this paper, we present Fiesta, a decentralized framework for learning mobile spectrum data on large scales. Fiesta leverages two emerging concepts, federated learning and blockchain, to address the aforementioned deficiencies. We modify the ordinary federated learning to support asynchrony so that it not only permits sensitive (meta-)data to stay at the sensor but also is pragmatic to be deployed on large scales. We also design a blockchain structure to make sharing the learned model straightforward and robust to single point of failures. We have deployed Fiesta based on spectrum sensors connected to Android mobile devices in a mid-sized US city for more than 3 months and collected more than 600 hours of raw spectrum data with 220K records for two applications. Evaluation results show that Fiesta can achieve roughly 4dB accuracy for estimating TV channels' power across space and 5dB accuracy for reconstructing LTE band PSD data using machine learning models.

ACM Reference Format:

Anonymous Author(s). 2022. A Decentralized Framework for Learning Mobile Spectrum Data on Large Scales. In *Proceedings of ACM Conference (Conference'17).* ACM, New York, NY, USA, 14 pages. https://doi.org/10.1145/nnnnnnn.nnnnnnn

Conference'17, July 2017, Washington, DC, USA

ACM ISBN 978-x-xxxx-xxxx-x/YY/MM…$15.00
https://doi.org/10.1145/nnnnnnn.nnnnnnn

1 INTRODUCTION

The scarcity of wireless spectrum and the exploding mobile data traffic make the regulation authorities consider opening previously allocated bands for secondary users, e.g. TV whitespace [7, 14, 43] and CBRS band [37], in addition to leveraging existing unlicensed bands [10]. The success of such an endeavor highly depends on a deep understanding of how these bands are used by the primary users. This requires large-scale spectrum sensing measurements, across frequency, temporal, and spatial domains. In order to make it possible, crowdsensing has become a popular spectrum measurement paradigm. For example, Microsoft Spectrum Observatory [3] and Electrosense [33] have supported nationalwide or even continental-wise spectrum measurement with high-end and low-end SDR platforms, respectively.

Given a large amount of collected data in spectrum crowdsensing platforms, prior spectrum data analysis efforts usually require uploading data to a centralized cloud for further analysis. TxMiner [49] based on Microsoft Spectrum Observatory leverages Azure cloud service to analyze each data record and find active transmitters. BigSpec [45] also takes advantage of big data techniques to conducts analytics directly performs computations of multiple applications, e.g. energy detection, on the compressed data. Moreover, data is similarly uploaded to the cloud for further analysis in Electrosense, but here the uploaded data can be already processed initially at the sensors, for example, ADS-B IQ sample decoding can be performed at the sensor before data uploading [9, 44].

However, this centralized data analysis paradigm has two deficiencies. (1) *Privacy issue of data uploading for individuals.* In spectrum crowdsensing platforms, spectrum data is collected with meta-data, e.g. GPS information, including timestamp and location, which is sensitive for most participants. The spectrum data itself can also be sensitive. If the communication of participants or their neighborhoods is not properly encrypted or compressed, the decoded information from captured IQ data and processed PSD (Power Spectrum Density) data can also reveal sensitive information, e.g., occupation and online consumption habits of the users or their neighbors. Therefore, uploading data to a centralized cloud in spectrum crowdsensing platforms, in addition to the lack of proper incentive mechanisms for participants, results in a significant obstacle to wide adoption. (2) *It is difficult to avoid single point of failures, and it makes sharing the findings*

unnecessarily hard. Facebook, one of the largest global social networking service providers, experienced a major service outage on Oct. 4th, 2021 for about 6 hours globally [12] due to a DNS configuration error. This shows the possibility and potential of the risk that one single entity owns the infrastructure entirely. Furthermore, one entity owning the entire infrastructure makes it hard to share the findings based on the crowdsensed data. For example, as a data contributor to Electrosense, although it is easy to view and analyze the data collected by the sensor we own, access to data from other participants is possible only at lower spectral resolution.

To address these two deficiencies, we present Fiesta, a decentralized framework for learning mobile spectrum data at large scales. We especially aim at the case where sensors are mobile and attached to smartphones to enable large-scale sensing, yet the most challenging case given that GPS information is sensitive and the practical implementation on mobile has to address its constrained resources. We mainly focus on model learning based on spectrum data, given that data analysis tasks can be broadly divided into model learning and model inference, and model learning is computationally much more difficult than model inference. Fiesta is decentralized in two regards. First, *it leverages federated learning such that model learning based on spectrum data mainly takes place at the mobile devices with sensing capability and does not require uploading the raw data.* Although federated learning is not perfect in privacy preserving in the sense that data still can be partially inferred [13], it is significantly better than the centralized approach. Second, Fiesta has a distributed ledger, i.e. blockchain, to aggregate and record the updates of the global model. *This component makes it easy for organizations to join the platform and prevents single point of failures, and sharing the learned global model based on the spectrum data is inherently supported.* In addition, carefully designed reward mechanisms based on the blockchain can further promote the participation of both individuals and organizations.

Ordinary federated learning requires the participants to be synchronized in the sense that the model aggregation is performed after each participant completes a local training round. Unfortunately, it makes ordinary federated learning hard to be deployed on large scales. Spectrum changes continuously over time, frequency and space so that data needs to be collected and then learned in a distributed manner, therefore synchronous operations are hard to reach. This problem is exacerbated in Fiesta, given that participants can download different versions of the global model at the same time. We address this asynchronous learning problem by proposing a different approach for the aggregation method and empirically show that the updates on different versions of the global model can be combined without much effect on the convergence and accuracy of training.

Blockchain was originally proposed to realize decentralized digital cryptocurrencies. We take advantage of its decentralized manner to avoid single point of failures in ordinary federated learning. We design a block structure, which contains the current global model, so that sharing global model is naturally supported. In order to make it more efficient and pragmatic for large-scale deployment, we also solve several implementation issues, for example how to support oversized models and the evolution of the model structure.

We have implemented and deployed Fiesta based on mobile devices with spectrum sensor board attached to learn how TV and LTE bands are used in a mid-sized US city for 3 months with 600 hours of power spectrum density (PSD) data (220K records) collected. The data and code will be made available with the camera-ready version of this paper.

To summarize, we make the following contributions:

- To the best of our knowledge, this is the first work that proposes and designs a decentralized architecture for learning spectrum data instead of using a traditional centralized computing platform.
- We address the single point of failures in ordinary federated learning with the help of blockchain and tackle the consequent challenge of asynchronous learning. Simulation results show that Fiesta can reduce significant fluctuations of the learning curve in vanilla federated learning in asynchronous scenarios, and is robust to network failures, e.g. node failures and partition failures.
- We have deployed Fiesta for 3 months, collected 220K PSD data records in 600 hours using Android smartphones. We achieve roughly 4dB accuracy for estimating TV channels' power across space and 5dB accuracy for reconstructing LTE band PSD data using machine learning models. The dataset and codebase will be released for further research.

2 RELATED WORK

Spectrum sensing. Spectrum sensing is traditionally conducted via bulky and costly spectrum analyzers [39, 42, 45, 46]. Due to the high cost of setting up the sensing infrastructure, although it can cover a wide bandwidth with a high frequency resolution, previous works either mainly measure spatial usage patterns and ignore temporal variations [39, 42], or cover many locations but assume time invariance [46]. BigSpec covers both spatial and temporal domains by mounting the spectrum analyzer on a mobile vehicle, but the dataset collected is still sparse in the spatio-temporal domain. A more long-term spectrum measurement effort that has a wider coverage is Microsoft Spectrum Observatory [3]. However, its cost, with only tens of measurement locations, is bearable only for big companies like Microsoft, and the network is now deprecated. Despite the less frequency resolution and a narrower bandwidth, with the advent of low-cost sensing devices, e.g. RTL-SDR [4] and HackRF [2], crowdsensing, e.g.

[15, 28, 33, 36], is possible to cover both spatial and temporal domains. Among the previous efforts, Electrosense [33] has enabled a large-scale and long-term spectrum observatory. It overcomes the deficiency of narrow bandwidth by installing a frequency extension board. Nevertheless, due to the static nature of the sensors, its coverage on the spatio-temporal domain is still not dense enough so that the spectrum usage at any timestamps and locations can be estimated based on the data collected. In order to make the crowdsensing platform have even more adoption and good mobility support, low-cost and convenient sensing devices based on mobile phones or tablets have attracted many research efforts recently [24, 28, 47].

Spectrum data analysis. Although spectrum can be measured with a high frequency resolution using high-cost spectrum analyzers, traditionally spectrum data analysis is performed on data at channel level. For example, they can analyze channel level occupancy and power [38, 42, 46]. Converting the frequency bin level data to channel level data (e.g. from bins of 10 kHz to channels of 5 MHz) significantly reduces the size of spectrum data and removes essential information for applications beyond waterfalls plots, such as anomaly detection and technology classification. Recently, with the success of big data techniques and machine learning, spectrum data can be analyzed on the frequency bin level to do fine-grained analysis. Latest advances include classification of signals [34], detection of transmitters [49], and detection of anomalies [35, 45]. Nevertheless, as mentioned earlier, these efforts require uploading the spectrum data to a centralized cloud, which has its own deficiencies.

Federated learning. Federated learning [18–20] is a promising alternative to traditional centralized learning. It addresses the problem of collaborative machine learning among multiple clients without uploading their sensitive local training data to a centralized cloud. It has attracted large attention recently and can be applied to various applications, e.g. recommendation system [29], human activity recognition [30], and keyboard prediction on mobile devices [8]. Nevertheless, an open problem of federated learning is the potential single point of failures introduced by the central server [13], which aggregates the model updates from clients. While large organizations or companies can play this role, it may be not always reliable and relatively unwilling to collaborate if a third party wants to join the system. Therefore, a fully decentralized version of federated learning is necessary. A few recent works tackle this problem by leveraging blockchain, e.g. [16, 32, 40, 48]. However, they are mostly theoretical works, which lack the ability of wide adoption. More specifically, they do not consider the asynchronous learning problem introduced by blockchain. Because the different blocks in the blockchain contain different versions of the global model, the asynchronous learning problem is unavoidable in order to have a pragmatic system. Furthermore, the heterogeneity of mobile devices and the different speeds of data collection in a time period by various clients exacerbates the problem. In addition to the problem of asynchronous learning, these prior works also ignore some practical issues, for example how to deal with machine learning models with large size and how to support model structure evolution, which are critical for the long-term success of the system. Moreover, [41] tries to tackle the asynchronous learning problem but ignores single point of failures and requires participants to upload updates per epoch rather than per round (of multiple epochs), which is impractical.

3 SYSTEM DESIGN

In this section, we present the design of Fiesta. The core idea of Fiesta is that we *use federated learning instead of centralized learning so that the spectrum data and GPS information do not leave mobile devices*, and we also *leverage blockchain to record the update to the global model in a distributed fashion to avoid single point of failures.* In the following, we first present the architecture of Fiesta, next explain how the learning process works in Fiesta, then discuss the reward mechanisms to encourage participants, and finally illustrate how can one join (or leave) Fiesta without affecting the whole system.

3.1 Architecture

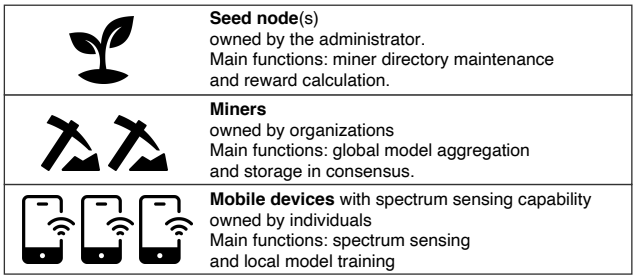


Figure 1: Layered architecture of Fiesta.

There are three different roles in Fiesta, as shown in Fig. 1, *mobile devices*, *(ordinary) miners*, and *seed node(s)*.

The mobile devices in Fiesta have spectrum sensing capability and are owned by *individuals*. They mainly operate in two modes, data collection mode and learning mode. When in the data collection mode, they record the spectrum readings and the corresponding meta-data, i.e. latitude, longitude, and timestamp. When in the learning mode, they compute the local updated model based on the current global model and the collected local spectrum data, then communicate with miners for model aggregation.

The miners in Fiesta, owned by different *organizations*, e.g. companies like Microsoft and non-profit associations like Electrosense, could in the future batch a bundle of updates from mobile devices into a block and calculate the aggregated global model. They also let the mobile devices know the up-to-date global model. All the versions of the global model and

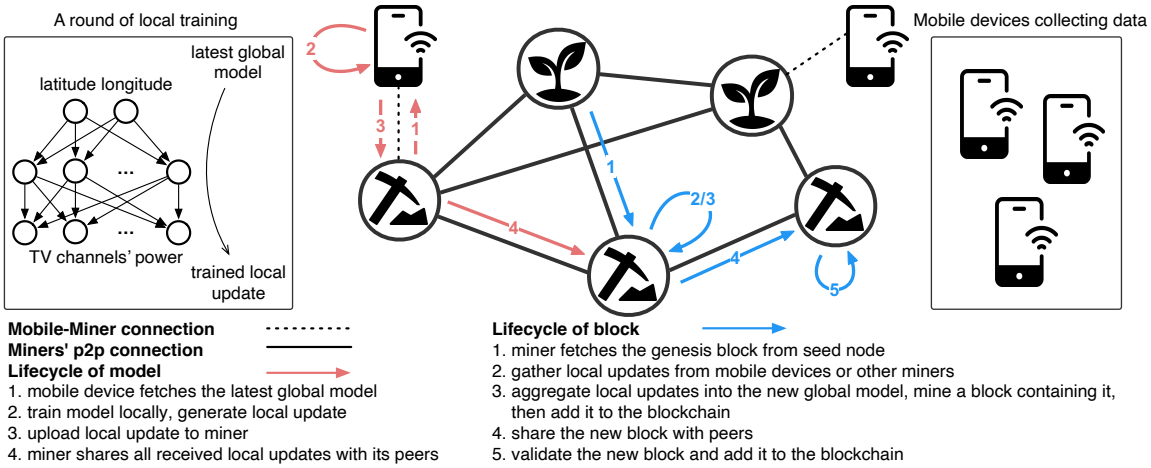


Figure 2: Learning process of Fiesta.

the corresponding updates from mobile devices to get the global model are recorded in the form of a blockchain. The Proof-of-Work (PoW) mechanism of the blockchain solves the consensus problem within the miners.

The seed node(s) in Fiesta are owned by the *administrator* of the community (e.g. the government or the regulation authority like the FCC), and they mainly have two additional functions compared with ordinary minors. First, they maintain a directory of working miners such that new mobile devices and miners joining the community know where to contact the existing miners. Second, they periodically scan the blockchain to decide the reward to each participating device/individual and miner/organization.

3.2 Learning process

Fig. 2 illustrates the model learning process in Fiesta. Compared with ordinary federated learning, Fiesta needs to handle the challenge of *asynchronous learning*. Due to the heterogeneity of the resources within the mobile devices, the different amounts of data each mobile device collects, and the latency and throughput of the wireless connections during model learning, it is common to have a variety of speeds per local round of training. Furthermore, due to the block propagation delay within the miners' network and the possible forking of the blockchain, mobile clients can download different versions of the global model at the same time. The openness of Fiesta also makes it possible to have mobile devices joining a long time after others have started the learning process, or devices joining only for a limited time, which exacerbates the challenge of asynchronous learning. Moreover, because the mobile devices switch between learning mode and data collection mode, the amount of data on each mobile device is increasing over time (although stays the same during training), which is also different from ordinary federated learning.

Local training at mobile devices. The main idea is to use a single machine learning model to describe the spectrum data according to the exact application. The mobile device starts training when the individual enables the learning mode and only keeps training if the blockchain is extended with a new block since the last pulling from the miner. If there are no new blocks, the mobile device will wait for an average block generation time before it starts pulling for the newest block. If there is a new block or training mode just enabled, then one round of local training is performed based on the global model within, for example, train a neural network for a round based on local PSD data (frequency information) and GPS readings (spatial information) to estimate TV channels' power at different locations, as shown in Fig. 2. Then, the mobile device uploads the base block id, the updated model, the number of data recorded locally, the training loss before and after local training for performance monitoring and reward calculation. All this information is signed using its private key for identity verification.

Updates aggregation at miners. As shown in Fig. 2, miners receive updates, forward to neighbors, batch them into a block, and compute the new global model. We use a modified version of FedAvg [25] to deal with asynchronous model aggregation. FedAvg is the most popular model aggregation method in the synchronous setting. It computes the weighted average of the model updates with the weights equal to the number of data records, and it is guaranteed to converge with non-iid (independent and identically distributed) data for strongly convex and smooth problems [23], although slower than the iid case. Formally, suppose client $i \in \{1, \ldots, C\}$ receives the same copy of the global model $\mathbf{w}_t$ at the beginning of round $t+1 (t \geq 0)$. Then, after local training, all clients (or only a fraction of all clients selected by the central server) generate their local update $\mathbf{w}_{t+1}^i$ using their local n_i number of data records. To get the new global model

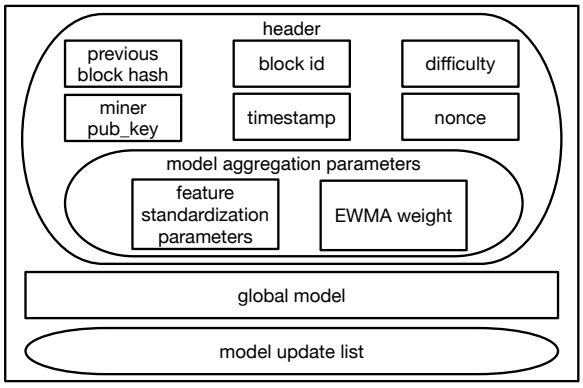


Figure 3: Block structure of Fiesta.

after round $t+1$, the central server in FedAvg computes

$$\mathbf{w}_{t+1} = \sum_{i=1}^{C} \frac{n_i}{N} \mathbf{w}^i_{t+1}, \tag{1}$$

where $N = \sum_{i=1}^{C} n_i$. Finally, this new global model $\mathbf{w}_{t+1}$ is pushed to all clients for the next round of training.

However, due to the asynchronous nature of Fiesta, it is possible that $block_{t+1}$ contains the model updates, which are based on the global model outputted in $block_t$, from clients with normal end-to-end latency only. In $block_{t+2}$, it may contain model updates, which are still based on the global model outputted in $block_t$, from straggling clients. We call these updates in $block_{t+2}$ based on $block_t$ 1 block delayed. In reality, the updates may be several blocks delayed instead of only 1. Due to this delay, when we aggregate model updates in $block_{t+2}$, we should also take the global model outputted in $block_{t+1}$ into consideration. As a result, we use the exponential weighted moving average (EWMA) of the global model in Fiesta model aggregation. Formally, in $block_{t+1}$, the miner first aggregates the model updates within this block same as FedAvg, and gets the temporary model, say $\mathbf{w}_{t+1}$. Then, the global model can be computed as

$$\mathbf{w}^{global}_{t+1} = (1-\alpha)\mathbf{w}^{global}_{t} + \alpha \mathbf{w}_{t+1}, \alpha \in (0,1). \tag{2}$$

Note that the round notion in the synchronous setting does not apply here anymore, we define *async round* as each time a new block is generated with a new global model within. In addition, if miners receive multiple updates from the same mobile device in an async round (say $\mathbf{w}^i_{t-1}$ and $\mathbf{w}^i_t$), they should only use the most recent one ($\mathbf{w}^i_t$) for aggregation so that too stale updates will be ignored.

Block structure. Fig. 3 shows the structure of a block. The principle of block structure design is that all parameters that affect global model aggregation should be recorded in the block, besides standard blockchain required parameters. For example, we include the parameters for feature standardization[1] and the exponential weight for EWMA during model aggregation in the block. Local training related parameters are not included in the blockchain, e.g. optimizer type and the learning rate.

Mining difficulty and block generation speed. In a blockchain, blocks are chained by containing the hash of the previous block and it requires the hash of each block to contain at least d leading zeros, which is the mining difficulty. The mining difficulty also decides the block generation speed accordingly. The mining difficulty balances a tradeoff between the first confirmation time (the time taken to generate the next valid block) and the amount of work wasted due to blockchain forking. A lower mining difficulty means a higher block generation speed, which leads to a higher possibility of blockchain forking and more work wastage but also a shorter time for first confirmation. Because we currently aim at a city-scale deployment, it usually takes up to tens of milliseconds to propagate a block within the miner network. Therefore, as long as the average time to generate a block is on the order of 10 seconds, we will waste roughly 0.1% of the work only with an acceptable confirmation time compared with a round of local training time at mobile devices.

Block verification. Before adding the new block received from a neighbor to the blockchain and forwarding it to other neighbors, a miner needs to verify its validity. In addition to checking the PoW validity and block generation time and size, which are similar to bitcoin [27], the followings also need to be verified. First, the relatively stable parameters are the same as the previous block, e.g. feature standardization parameters. Second, the new global model is correctly computed based on the local updates within the block. Third, there are no multiple updates from the same mobile device within a block. This last one is important because it not only affects model aggregation result but also the reward allocation, which will be shown later.

3.3 Reward mechanisms

Rewards within a certain time period are calculated by the administrator based on the information recorded in the blockchain. We distinguish two types of rewards, one for individuals and one for organizations. Note that the end time of the time period cannot be too close to the current time, in which case the blockchain may not reach a consensus due to forking. In addition, we assume the information reported by each individual, e.g. number of data collected and the loss value before and after local training, are trustworthy. We leave the untrustworthy scenario as future work and will discuss more in § 6.

Individual reward: Previous work on generalized blockchain and federated learning [16] utilizes a reward mechanism that is proportional only to the number of data records each device contributes. However, this approach is not suitable for spectrum sensing. For instance, for learning spatial variations of a TV channel, suppose there is a stationary device

[1] Feature standardization makes the values of each feature in the data have zero-mean and unit variance, which is a common data preprocessing step before actual model training.

that keeps sensing the channel. If the reward is purely based on the number of data records, this device will tend to gain more reward without providing much insight into how this channel is utilized across the area, which is undesirable. Thus, we decide that the reward of an update for each mobile device/individual should be proportional to its contribution in loss function minimization. Suppose a model update from client i is trained on local data set D_i with $|D_i| = n_i$, and it uses $block_t$'s global model $\mathbf{w}_t^{global}$ as the initial parameters, and outputs $\mathbf{w}_{t+1}^i$. Denote the average loss on each data record as $l(\mathbf{w}; D_i)$. The individual reward is calculated as follows:

$$r_{t+1}^i = n_i \cdot \left[l(\mathbf{w}_t^{global}; D_i) - l(\mathbf{w}_{t+1}^i; D_i) \right]. \quad (3)$$

Since we assume the mobile clients report these values with the model update to the miners faithfully, the individual rewards can be calculated reliably at seed node(s). In order to gain more rewards with the same n_i, one can enlarge the difference between $l(\mathbf{w}_t^{global}; D_i)$ and $l(\mathbf{w}_{t+1}^i; D_i)$ by measuring the locations that are not captured by the existing global model, e.g. for the exemplary application of estimating TV channel's power at different locations, rather than visiting the locations that are already measured. This is also beneficial from the entire community's perspective. Furthermore, this reward mechanism also motivates individuals to use appropriate types of optimizer and parameters to decrease the loss quickly, and to have high computational power and low network latency to finish more rounds of local training within a certain time period. Note that the reward is for a fixed number of epochs (as a local round). Increasing reward by conducting local training for too many epochs before uploading can be harmful for convergence [23].

Organization reward: It is desirable that the reward to each organization is proportional to the contribution in the computing power that aggregates updates from mobile devices and generates the new global model. Therefore, the reward to each miner/organization is proportional to the number of blocks generated in the blockchain within the time period, similar to bitcoin.

We envision that the funding for the reward mechanisms can come from two resources, including the administrator itself, and the revenue generated by selling the learned model or the right to query the learned model. The allocation of the funding between the individual reward and the organization reward can be dynamically adjusted to advocate more mobile devices with spectrum sensing capability or more miners.

3.4 Joining and leaving Fiesta

Due to the peer-to-peer nature of the blockchain protocol, it is easy to join Fiesta for both individuals and organizations, which is impossible for ordinary federated learning. Given the joining requests from organizations' miners and individuals' mobile devices, the seed node(s) reply with a list of miners they are aware of. The joining miners can then establish connections with a group of other miners. Different from miners, the individuals' mobile devices can select the miner with the smallest latency within this group before it enters the learning mode. As the result, the latency between the mobile device and the miner is minimized. In addition, the mobile devices also can obtain the recommended optimizer type and parameters, e.g. Adam [17] optimizer, the learning rate, and the decay rates of the momentum during their joining process, although not forced to use them. Moreover, the freedom of joining for both organizations and individuals makes sharing the model easy. As long as one is contributing to the system, it inherently has access to the up-to-date global model.

Similarly, miners and mobile devices can leave the system whenever they like, which overcomes the drawback of the single point of failures in ordinary federated learning.

4 IMPLEMENTATION & DEPLOYMENT

We implement the three roles of Fiesta's layered architecture using different hardware and software stack.

Mobile device. A mobile device is connected with an external sensor board through a USB cable to gain spectrum sensing ability, which is shown in Fig. 4(a). The sensor board we use is from [24]. It is equipped with an AD9361 RF transceiver and a ZYNQ-7020 FPGA chip, featuring strong spectrum sensing capability ranging from 70MHz to 6GHz with a 56MHz sampling rate and high programmability. The mobile device we adopt in our platform is Google Pixel 2 running Android 11. We also use Galaxy Tab A running Android 11 as an alternative. Users can check the status and interact through an Android app, as shown in Fig. 4(a). The app will start the sensing mode when the sensor board is attached, and will start the learning mode when the mobile device is charging with WiFi access (our devices do not support cellular data plan). In terms of the software stack, usb-serial-for-android [5] and deeplearning4j [11] are integrated into the Android app to facilitate serial communication with the sensor board and on-device machine learning.

Miner. Miners are realized using Flask [1], a lightweight web application framework based on Python, and all the communications between mobile devices and miners use HTTP protocol for simplicity. An ordinary blockchain node using Prove-of-Work (PoW) consensus algorithm written in Python consists of the main body of each miner. Having received several local weight updates from mobile devices, the miner will integrate them into a new block and compute the new global model, described using PyTorch [31], for the next async round of training. For our experiments and deployment, several miner servers are deployed on different host machines including a 16-core 16 GB RAM personal desktop and an 8-core 8 GB RAM lab workstation.

Seed node. Seed node is also a Flask server and it uses HTTP to communicate with other roles as well. The seed node is backed up with two tables, one noting down the members registered for Fiesta and the other recording the contribution and reward for each member. In our design, there are a limited number of seed nodes and each of them should be reserved for only the administrator. For our experiments and deployment, a seed node is running on a 16-core 16 GB RAM desktop with a public domain name to allow other roles to query the seed node and register themselves through the Internet.

4.1 Implementation Issues

We resolve several practical issues during implementation to make Fiesta's implementation more efficient and easier to be deployed.

When to start the learning mode on mobile devices. Model training on mobile devices is not recommended by most popular frameworks like Tensorflow [6] and PyTorch [31] because it is both time and energy consuming. In order to minimize its impact on the daily usage of mobile devices, we only perform local training at stationary environment with power cable plugged (which also means the sensor board is detached) as well as WiFi access, and start the task only after a significant amount of new sensing data is gathered.

Model compatibility. As mentioned previously, we use deeplearning4j on Android mobile devices while using PyTorch on miner servers to realize the machine learning model, due to no training support of PyTorch for java. This makes it impossible to directly use the local model updates from mobile devices to generate the new model at the miner. We implement two helper functions to (de)serialize the model weights and translate between the machine learning models in deeplearning4j and PyTorch.

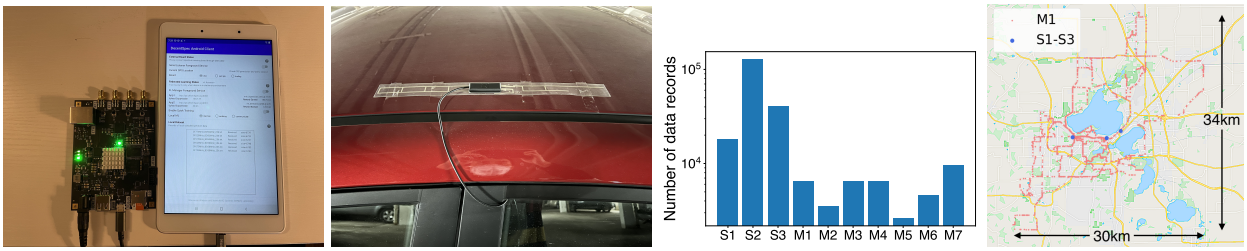
Minor update suppression. When the model converges and the pattern of the local data is already learned, the mobile device should not upload the local model to save the network bandwidth of both the mobile device and the miners as well as the computing resources at the miners. Given the reward mechanism we propose, this can be implemented as the mobile device stops uploading the update if the individual reward calculated based on local training loss reduction is smaller than the minimal unit of the reward, i.e. when the local training loss does not decrease significantly, or the reward is non-positive.

Oversized models. Model size can be large, which makes block size large. This leads to two consequences: a heavy burden on the network bandwidth during miner communication and on the computation workload to the hash function when doing PoW. To address the former, we only keep full information of a block in its generator miner and another archive server (e.g. seed node), and share only the shrank block with detailed local models substituted by a single global model to its peer miners. To address the latter, we perform a pre-hash over those local updates, and use the generated fixed-length hash string to further generate the hash of the whole block when finding the nonce during PoW. As a result, hashing of a long model is replaced with hashing a shorter and fixed length string. Note that here we only address the impact of large model size on the blockchain component of the system instead of the mobile devices. Our model size (<2 MB) can be easily stored in the mobile device's memory (4 GB for Google Pixel 2), and we only start learning when having WiFi and power access. As such we leave addressing the impacts of large model size on mobile devices as future work, which can be mitigated through methods proposed in [20, 22, 29].

Model structure evolution. The structure of the model may need to grow more complex to better fit the increasing number of data records. At this time, seed node(s) and only seed node(s) will send out a broadcast message with their signature, which then will be verified by miners and other seed node(s), such that the miners start to work on a new blockchain upon receipt of this message. The genesis block of the new blockchain contains the new model structure. In order to fully utilize the data already obtained, there could be a knowledge transfer process before announcing the new structure, which means the global model in the genesis block of the new blockchain may not be randomly initialized. All updates for the old blockchain from the mobile devices will be ignored by the miners after receiving this message, and will be replied with the new model structure immediately. However, the old blockchain is still stored by the seed node(s) until the rewards are settled. In addition, this reseeding process can also be used to handle the scenarios where some stable parameters of the blockchain need to be changed, e.g. changing feature standardization parameters because the sensing area is changed.

4.2 Deployment and data collection

We deploy 10 sensor boards with mobile devices in a mid-sized US city for 3 months. 3 sensor boards are static (S1-S3), and the rest are deployed on personal vehicles of volunteers (M1-M7). Fig. 4(b) shows an antenna mounted on a vehicle. Due to the frequency constraint of the antennas, we configure the sensor boards to capture Power Spectrum Density (PSD) readings of 256 bins from 500MHz to 800MHz on a 50MHz basis, among which 500-700MHz is primarily for TV broadcast and 700-800MHz is mainly for LTE usage. PSD data is collected at the rate of one data record per minute for each 50MHz band. Fig. 4(c) shows the number of data records per sensor. Overall, we have gathered more than 220K data records of 600 hours. Fig. 4(d) illustrates the locations of measurements. It shows the static nodes and the GPS trace of one mobile node, only for illustration.



(a) Sensor board controlled by an Android app on a mobile device. (b) Antenna mounted on a vehicle. (c) No. of data records per device. (d) Locations of measurements.

Figure 4: Implementation and deployment of Fiesta.

In terms of servers, we launch 5 miners, among which 3 are always on and the rest can go offline at the maintainer's will to introduce dynamics into the miner network instead of having a constant one. We also have two seed nodes using a 1+1 configuration to avoid single point of failures of the seed node. This can be further optimized using DNS-based load balancing.

We have two different machine learning applications during the deployment to demonstrate the functionality of our design. **App1**: This application is to learn how the TV spectrum is used in the spatial domain through a neural network model predicting the signal power of TV channels at different locations. The model is a feed-forward neural network with latitude and longitude as inputs and signal power of TV channels as output, and can be used to not only estimate the spatial usage pattern but also to detect spatial domain anomalies [45]. **App2**: This application is learning how the LTE band is used in the frequency domain by training an autoencoder on the PSD data. It takes the PSD data as input and the output of the autoencoder tries to reconstruct the input as much as possible. The reconstruction error can be used to detect frequency domain anomalies, similar to [35].

5 EVALUATION AND RESULTS

We first benchmark Fiesta's performance using simulations and compare with baselines based on available spectrum data in [45], which contains TV band PSD data with spatial variations in a city. It is important to verify the system's correctness and robustness before real-world deployment. We then report the evaluation results using the measurement data from real-world deployment.

5.1 Microbenchmarks

In order to verify the correctness and robustness of Fiesta, we use the following simulation setting. We have 5 miners and 10 mobile devices in the system, which is the same as the real-world deployment. The learning mode of mobile devices is simulated through Python code. One round of local training in a mobile device contains 10 epochs. We do not simulate the data collection mode of mobile devices so the data on each mobile device is fixed during simulations. All miners and mobile devices processes run in parallel in a single 28-core machine so that each process occupies a separated core. We use **App1** for benchmarking. For simplicity, we only use the readings of channel 19 (500-506 MHz) in [45]. The data are randomly sampled with 10% probability as the validation set first. For the rest 90%, they are sorted by timestamp and then allocated to the mobile devices as 10 consecutive time periods of a same number of data records. Although the number of data records on each mobile device is the same, there is a certain degree of heterogeneity in terms of spatial distribution.

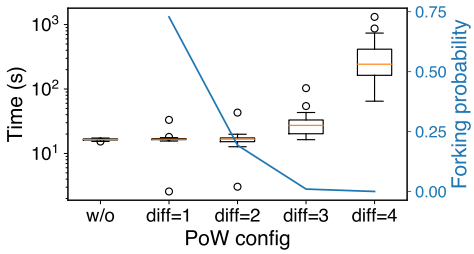
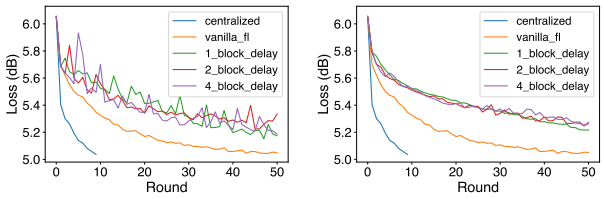


Figure 5: Impact of mining difficulty.

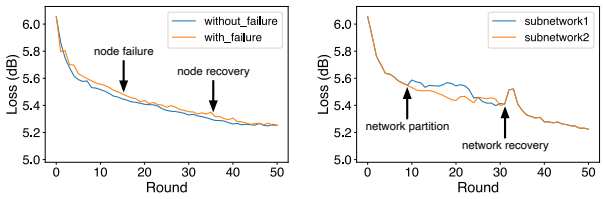
5.1.1 Mining difficulty. We first decide the mining difficulty to balance the first confirmation time and the forking probability of blockchain. We calculate forking probability as the total number of block replacements due to knowing a longer chain from other miners divided by the total number of blocks in the blockchain. In this benchmark, we use the synchronous setting, i.e. the miners must batch all model updates from the 10 mobile devices into one block, to simplify the configurations.

We vary the difficulty and compare it with the baseline, where PoW is disabled in the system. Fig. 5 shows the results. When the PoW is disabled, we can see that the first confirmation time is 11 seconds, without many variations in the distribution. The major factor of this first confirmation time is attributed to the time spent in the mobile devices for a round of local training. Network latency is negligible in this case. When the difficulty is no more than 2, although there are some outliers, the distribution of the first confirmation time does not change much compared with the result when



(a) Without EWMA. (b) With EWMA.

Figure 6: Impact of model aggregation with EWMA on model convergence and accuracy.



(a) Miner node failure. (b) Miner network partition failure.

Figure 7: Impact of failures on model training.

PoW is disabled. However, when the difficulty is greater than 2, the first confirmation time starts increasing exponentially, which means the time spent mining for a valid block becomes the major bottleneck. Moreover, the distribution of the latency is not concentrated to the mean value anymore, which is different from the baseline. As for the result of forking probability, we can see from Fig. 5, if the difficulty is no more than 2, we have a significant non-zero possibility of forking. Nevertheless, if the difficulty is greater than 2, we can reduce the forking probability to a value close to zero.

Considering the results of first confirmation time and forking probability, we fix the difficulty as 3 for the rest of our simulations and the real-world deployment. Note that forking probability is also related to the scale of the miner network. The larger scale of the miner network, the higher the forking probability it can get. We use a fixed difficulty rather than a dynamic one because the scale of our miner network does not increase significantly during the experiment.

5.1.2 Model convergence and accuracy. We study the effect of model aggregation with EWMA introduced in § 3.2 on model convergence and accuracy. We show in this subsection that the proposed variant in the model aggregation is very useful in asynchronous settings.

We introduce asynchrony mainly from two aspects in this benchmark. First, we simulate that the speed to perform a round of local training on mobile devices can vary significantly due to the heterogeneity of mobile devices. Second, because the block size of blockchain usually has an upper bound, each block can only contain model updates from a portion of the mobile devices, which makes the asynchrony due to different speeds to perform local training inextricable. As a result, we let the blockchain contain 5 model updates in each block precisely and divide the 10 mobile devices into two groups. There are 5 faster mobile devices that upload their updates immediately after their local training round is finished, i.e. no block delay. The other 5 mobile devices are the slower ones, whose updates are at least $b (b >= 1)$ blocks delayed (specified in the legend of Fig. 6). We use the loss on the validation set, which we randomly choose from the training set in advance, to check the convergence and accuracy.

Fig. 6 illustrates the results. If EWMA is not used when doing model aggregation, we can see from Fig. 6(a) that the loss would fluctuate significantly, for example, the 4-block-delay setting from round 3 to 6 and the 2-block-delay setting from round 13 to 15. However, Fig. 6(b) shows that if EWMA is used when doing model aggregation, the learning curves are very smooth without sacrificing accuracy, compared with not using EWMA. The initial loss is a little bit higher compared with Fig. 6(a) but it still converges quickly after several rounds. Having large fluctuations in the learning curve is even more undesirable if we consider the possibility of mobile devices joining/leaving, the heterogeneous data distribution on each device, and new data collected. Additionally, Fig. 6 also contains the results of centralized learning and vanilla federated learning in the synchronous setting. From Fig. 6, we can observe that in the synchronous setting the model converges the fastest and has the best estimation accuracy using the centralized learning, and vanilla federated learning in the synchronous setting converges slightly faster than the results of the asynchronous settings and has a marginally better estimation accuracy.

Note that we set the α in EWMA as 0.5 for all the simulations and the real-world deployment because we weigh the past and the current equally. This parameter can be fine tuned if one desires.

5.1.3 Robustness to failures. We study the robustness introduced by blockchain, which does not exist in centralized learning and ordinary federated learning. We use an asynchronous setting where each block contains 5 to 10 model updates with EWMA enabled.

We first study whether a miner node failure can affect the system, which is equivalent to single point of failures in centralized learning and ordinary federated learning. Fig. 7(a) illustrates the validation loss as a function of the number of rounds with miner node failure and recovery. A randomly chosen miner node fails after round 15 is finished and is back online after round 35 is finished. Obviously, the training can continue during the period when the miner node goes offline. We are more interested in whether this failure can affect the learning curve, compared with when there is no failure. We

note from Fig. 7(a) that it converges to the close accuracy with the approximately same speed. One may notice that the no-failure curve does not perfectly overlap the with-failure curve. This fluctuation is introduced by the randomness between experiments, like HTTP respond arrival order, and does not compromise our demonstration of the system robustness. Consequently, we believe Fiesta is robust to single point of failures.

By using a distributed miner network, another type of network failure is possible, which is network partitions. Similar to miner node failure, after block 10 is generated, the miner network is partitioned into two subnetworks, with one subnetwork containing 2 nodes and the other containing 3 nodes. After block 31, the two subnetworks merge back into one. Further, the validation loss still gradually goes down, almost the same as the no failure case in Fig. 7(a). Thus, we conclude Fiesta is robust to network partition failures.

5.2 Real-world evaluation results

After the demonstration with 10 simulated mobile devices on real spectrum data, we deploy Fiesta using physical Android devices along with sensor boards. These devices present a wide variety in terms of their models (tablets and phones) and mobility status (stationary or portable), which cover the majority of real-world scenarios.

5.2.1 Overhead on mobile devices. Machine learning on mobile devices may be of low efficiency and time-consuming. Hence, profiling of its overhead is necessary.

CPU and memory. Fiesta Android client occupies less than 200MB RAM, and 50% cpu utilization on Google Pixel 2 when running two training threads in the background.

Network traffic. In order to fetch the global model from a miner and upload the trained local model to a miner, a limited bandwidth is occupied by 220KB for **App1** and 1.5MB for **App2** during one local round of training.

Training speed. A local training round (contains 10 epochs) with a training set with 4k data records takes 1.3 minutes for **App1** and 2 minutes for **App2** on Samsung Galaxy Tablet A 8.0, and slightly faster on Google Pixel 2. The difference between applications comes from the variance of their neural network structures. We also observe that one round of training time is proportional to the size of the local dataset and the number of epochs.

Energy consumption. Although local training is triggered only when the device gets plugged with a charger, we still test energy consumption based on battery. Google Pixel 2 whose battery has a capacity of 2700mAh can hold up to 5 hours of continuous training, while Samsung Galaxy Tablet A 8.0, with a 6250mAh battery could support up to 8 hours.

When it comes to sensing mode, where the device is linked with a sensor board, tablets can last for 4 hours and only 2 hours for Google Pixel 2, because not only the Android device needs to power the board through OTG, GPS service also drains significant energy. In this case, battery life can be extended via external powering for the sensor board and using coarse network location to replace GPS.

5.2.2 Convergence and learned models of the two applications. We track the convergence during the whole federated learning procedure. Unlike an off-the-shelf training set with a fixed test set in our microbenchmarks, for the real-world deployment, it is hard to evaluate the trained model with a specific test set directly, considering the training set itself is gathered in a streaming manner, not to mention privacy concern. As a compromise, we define *AppLoss* using the weighted average of training loss among local updates within $block_t$.

$$AppLoss = \frac{1}{\sum_{i \in block_t} n_i} \sum_{i \in block_t} n_i \cdot l(w_t^i; D_i). \tag{4}$$

$l(w_t^i; D_i)$ is the loss after the last epoch of the local training round from mobile device i with local dataset D_i, where $|D_i| = n_i$. After calculation of the *AppLoss* for each block, "convergence curves" as functions of date for both apps are shown in Fig. 8. Note that the convergence curves contain at most 1 point per 24 hours for clarity. From the convergence curves we can notice that before Feb. 20th both apps have a similar convergence pattern. This shows that the convergence pattern is largely decided by when the participants start to enable training on their mobile devices.

Before Feb. 1st, there are only 5 sensor nodes deployed, including 3 static and 2 mobile. In the meantime, we require each block contains at least 4 local updates. In this setup, the federated learning works in a semi-synchronous way, hence slow reduction of both app losses could be observed. Starting from Feb. 1st, 5 additional sensor nodes are deployed and the losses are reduced rapidly. Further, the asynchronous nature of this bigger system emerges and leads to inconsistent compositions between blocks. As a result, noticeable fluctuations are observed from both apps afterward. There is another distinct leap of **App1** loss because we online upgraded the machine learning model through transfer learning on Feb. 20th. Detail is discussed in § 5.2.3. Finally, the losses of **App1** and **App2** converge to 4dB and 5dB respectively.

We are also interested in what those models have learned. Fig. 9 visualizes the estimated power of TV channel 49 (680-686 MHz) using the model of **App1**. From this heatmap and combining our prior knowledge with the city's building distribution, we notice that the areas with a higher density of high-rise buildings tend to have lower TV power and vice versa, which is expected. Fig. 10 shows the reconstruction result of a sample without obvious signals using the autoencoder trained for **App2**. The non-flat noise floor of the sensor including a spike around 720MHz is successfully revealed after reconstruction. Fig. 11 shows a signal not learned by the model, which is considered a frequency domain anomaly.

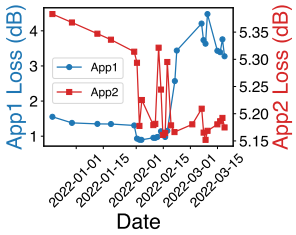


Figure 8: Model convergence over time.

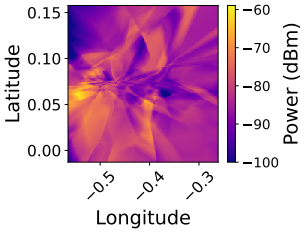


Figure 9: Estimated TV ch. 49 power distribution.

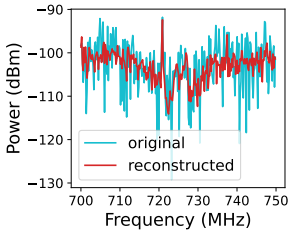


Figure 10: Example of reconstructed PSD data.

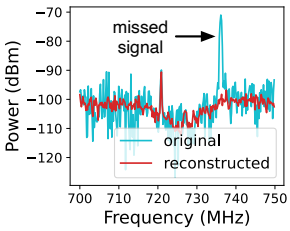


Figure 11: Example anomaly of LTE band.

5.2.3 Model structure evolution. On Feb. 20th, we online upgrade the structure of the model for **App1** to output multiple TV channels' power instead of only one. We achieve knowledge transfer by simply using the parameters of the last layer's neuron for the single TV channel power estimation to initiate multiple neurons corresponding to the multiple TV channels, and keep the hidden layers' parameter unchanged in the meantime. Although the loss increases, considering estimating multiple TV channels' power is more challenging compared with a single one with relatively less spatial variation, Fig. 8 still shows the ease of use of Fiesta when one needs to modify the machine learning model structure.

5.2.4 Effectiveness of model update suppression. As mentioned in § 4, when the pattern of local data has already been fully learned, i.e. when the local loss does not decrease significantly, mobile devices will retreat from uploading. Fig. 12 shows how frequently this kind of upload suppression happens among devices. From Fig. 12 we observe that for the mobile devices which have trained their gathered data more times, the upload suppression scheme is more likely to be triggered with non-zero probability. This is desirable since information within those data has already been fully absorbed. Note that the opposite may not hold considering when a mobile device that rarely trains the data it gathered, the knowledge within may already be obtained from another device with a large amount of data and frequently performs training. The outlier point on the left top of **App2** in Fig. 12 is an example of this case.

5.2.5 Effectiveness of the mechanism to deal with oversized models. Machine learning models with thousands or even millions of parameters will be a huge burden for the network and local storage. Fig. 13 shows our block compression policy with respect to the number of local updates within a block. Considering we replace the giant parameter Python dictionary of each local weight with a much smaller unit, the compression ratio is almost linear when the number of local updates per block is small. It converges to a relatively constant number as the number of local updates per block keeps increasing. And the more local updates a block contains, the higher storage we could save from storing blocks onto the hard disk.

We also show the CDF of the number of local updates within a block. We notice that due to the relatively small mining difficulty, most blocks contain only 4 local updates, the minimal number we set. Though we do not constrain the upper bound, the max number of local updates a block contains is 8, which is less than 10 due to asynchrony. **App1** blocks contain more local updates than **App2** blocks. It is reasonable considering the machine learning model of **App2** contains much more parameters than that of **App1**, hence requiring a longer time for local training. The more swiftly the local training completes the more local updates can be included in one block given a certain mining difficulty.

5.2.6 Reward mechanisms. We focus on individual rewards on mobile devices and omit the details of organization rewards from miners because miners' rewards are proportional to their computational capabilities as demonstrated in cryptocurrency systems. As mentioned in § 3, the reward mechanism previous work [16] adopts fails to reflect the actual contribution of each member. To demonstrate the effectiveness of our proposed reward mechanism shown in Formula 3, we compare it with the naive size-based reward mechanism as the baseline in Fig. 14 using 4 representative devices, including two stationary and two mobile. All the reward values are normalized by the maximum among devices.

Stationary device 1 performs local training on a large but roughly geographically homogeneous dataset considering it is deployed in a stationary environment. On the contrary, mobile device 1 and 2 have relatively smaller but more heterogeneous data because they are deployed on moving vehicles. From Fig. 14, we observe that the baseline mechanism grants higher rewards of the stationary device regardless to the little information within, while the proposed mechanism values the data diversity significantly. The difference is more distinct when it comes to **App1**, which tries to learn the spatial distribution of TV channels' power. One may also notice that stationary device 1 obtains much more reward than stationary device 2, although it collects less data as shown in Fig. 4(c). This is because stationary device 2 rarely trains the model and has submitted merely dozens of local updates along with the number of local data records, while stationary device 1 contributes over thousands of updates.

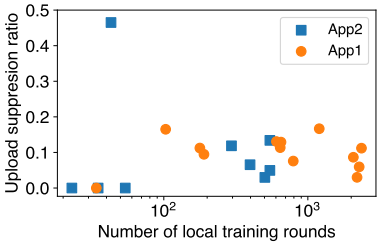


Figure 12: Upload suppression ratio.

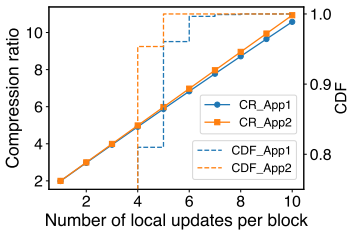


Figure 13: Compression and distribution of block size.

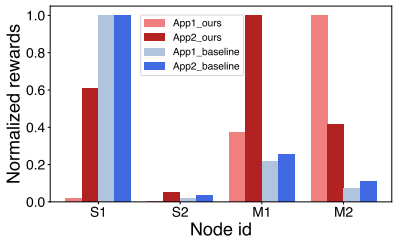


Figure 14: Comparison of different reward policies.

6 DISCUSSION AND FUTURE WORK

Privacy leakage. Although the data does not leave the mobile devices, there is still a possibility of privacy leakage in federated learning. Previous global models and the gradient updates from an individual can be used to infer the local training data held by that user [13], and these two pieces of information are available in Fiesta. Recent work on privacy-preserving federated learning [26] advocates running local training at mobile devices and secure model aggregation at servers in Trusted Execution Environments to reduce leakage. Nevertheless, whether it is fully compatible with Fiesta, especially with the openness and the joining freedom of the miners, still needs further exploration.

Adversarial robustness. In this work, we have assumed that both the mobile clients and the miners function with no malicious intent. Although it is a valid assumption currently given that we have full control of the hardware and software being used, it is probably no longer valid as the deployment scale enlarges. Taking advantage of the openness of Fiesta, malicious individuals can introduce data poisoning attacks and model updates can also be compromised by adversarial miners. How to deal with them remains an open question in the federated learning research area [13].

Model inference. Although in this paper we mainly focus on model learning, model inference is naturally supported by Fiesta, given that model inference is just a forward propagation during the training process. Not only model inference is much less computationally intensive, but it is also more well-supported than model training in popular machine learning frameworks e.g. PyTorch [31] and TensorFlow [6] on current mobile devices. We envision these popular frameworks will continue to evolve and better support model training on mobile devices in the future. Moreover, before model inference is enabled at mobile devices, the learned global model may need to be personalized due to non-IID distribution of data [21] for better performance.

Mobile device communication efficiency. As mentioned earlier, in Fiesta, we do not consider the communication efficiency of mobile devices as a major bottleneck due to the relatively small size of our models for spectrum related applications, compared with state-of-the-art deep learning models. If the size of machine learning models becomes very large so that mobile devices' communication inefficiency is significant, we can apply techniques proposed in previous works e.g. [20, 22, 29] to address it.

Application to other domains. In this paper, we show the efficacy of Fiesta via two PSD data based applications that can be formulated as self-supervised regression problems using gradient descent based optimizers. IQ data based applications can also be supported as long as they can be formulated as self-supervised regression problems. Classification problems, e.g. signal type classification, can be potentially supported as well. However, it needs a fraction of training data is labeled and then semi-supervised learning can be performed to leverage the whole data set. This needs manually labeling some data, which requires uploading some of the data to a group of domain experts. Therefore, ideally the data that needs to be labeled should come from administrators or organizations who can collect significantly more amount of representative data than ordinary participants with mobile devices and have more domain knowledge to label the data by themselves. Moreover, clustering problems are naturally supported because they do not require labeling, but how to interpret the clustering results requires domain expertise and validation on a fraction of raw data. The techniques proposed in this paper, although some are optimized for mobile spectrum sensing, can also be applied to learning other crowdsourcing data, like static spectrum sensing or even unrelated to spectrum.

7 CONCLUSION

A key hurdle for the wide adoption of spectrum crowdsensing systems is the centralized data analysis paradigm. We have presented Fiesta, a decentralized framework for learning (mobile) spectrum data on large scales. Fiesta addresses the two deficiencies of the centralized data analysis paradigm, namely privacy concerns as a result of uploading spectrum data from participants and single point of failure due to centralized analysis, by working at the intersection of federated learning and blockchain. We are also releasing the code and data set used in this paper to welcome further contribution and wider adoption from the community.

REFERENCES

[1] flask. https://github.com/pallets/flask.

[2] Hackrf. https://greatscottgadgets.com/hackrf/.

[3] Microsoft spectrum observatory. http://spectrum-observatory.cloudapp.net.

[4] Rtl-sdr. https://www.rtl-sdr.com.

[5] usb-serial-for-android. https://github.com/mik3y/usb-serial-for-android.

[6] M. Abadi, P. Barham, J. Chen, Z. Chen, A. Davis, J. Dean, M. Devin, S. Ghemawat, G. Irving, M. Isard, et al. Tensorflow: A system for large-scale machine learning. In *12th {USENIX} symposium on operating systems design and implementation ({OSDI} 16)*, pages 265–283, 2016.

[7] P. Bahl, R. Chandra, T. Moscibroda, R. Murty, and M. Welsh. White space networking with wi-fi like connectivity. *ACM SIGCOMM Computer Communication Review*, 39(4):27–38, 2009.

[8] K. Bonawitz, H. Eichner, W. Grieskamp, D. Huba, A. Ingerman, V. Ivanov, C. Kiddon, J. Konečnỳ, S. Mazzocchi, H. B. McMahan, et al. Towards federated learning at scale: System design. In *SysML*, 2019.

[9] R. Calvo-Palomino, H. Cordobés, M. Engel, M. Fuchs, P. Jain, M. Liechti, S. Rajendran, M. Schäfer, B. Van den Bergh, S. Pollin, et al. Electrosense+: Crowdsourcing radio spectrum decoding using iot receivers. *Computer Networks*, 174:107231, 2020.

[10] E. Chai, K. Sundaresan, M. A. Khojastepour, and S. Rangarajan. Lte in unlicensed spectrum: Are we there yet? In *Proceedings of the 22nd Annual International Conference on Mobile Computing and Networking*, pages 135–148, 2016.

[11] eclipse. deeplearning4j. https://github.com/eclipse/deeplearning4j, 2019.

[12] Facebook. More details about the october 4 outage. https://engineering.fb.com/2021/10/05/networking-traffic/outage-details/, 2021.

[13] P. Kairouz, H. B. McMahan, B. Avent, A. Bellet, M. Bennis, A. N. Bhagoji, K. Bonawitz, Z. Charles, G. Cormode, R. Cummings, et al. Advances and open problems in federated learning. *arXiv preprint arXiv:1912.04977*, 2019.

[14] M. M. Kassem, M. Kheirkhah, M. K. Marina, and P. Buneman. Whitehaul: an efficient spectrum aggregation system for low-cost and high capacity backhaul over white spaces. In *Proceedings of the 18th International Conference on Mobile Systems, Applications, and Services*, pages 338–351, 2020.

[15] M. Khaledi, M. Khaledi, S. Sarkar, S. Kasera, N. Patwari, K. Derr, and S. Ramirez. Simultaneous power-based localization of transmitters for crowdsourced spectrum monitoring. In *Proceedings of the 23rd Annual International Conference on Mobile Computing and Networking*, pages 235–247, 2017.

[16] H. Kim, J. Park, M. Bennis, and S.-L. Kim. Blockchained on-device federated learning. *IEEE Communications Letters*, 24(6):1279–1283, 2019.

[17] D. P. Kingma and J. Ba. Adam: A method for stochastic optimization. In *ICLR*, 2015.

[18] J. Konečnỳ, B. McMahan, and D. Ramage. Federated optimization: Distributed optimization beyond the datacenter. In *NIPS Optimization for Machine Learning Workshop*, 2015.

[19] J. Konečnỳ, H. B. McMahan, D. Ramage, and P. Richtárik. Federated optimization: Distributed machine learning for on-device intelligence. *arXiv preprint arXiv:1610.02527*, 2016.

[20] J. Konečnỳ, H. B. McMahan, F. X. Yu, P. Richtárik, A. T. Suresh, and D. Bacon. Federated learning: Strategies for improving communication efficiency. In *NIPS Workshop on Private Multi-Party Machine Learning*, 2016.

[21] V. Kulkarni, M. Kulkarni, and A. Pant. Survey of personalization techniques for federated learning. In *2020 Fourth World Conference on Smart Trends in Systems, Security and Sustainability (WorldS4)*, pages 794–797. IEEE, 2020.

[22] A. Li, J. Sun, P. Li, Y. Pu, H. Li, and Y. Chen. Hermes: An efficient federated learning framework for heterogeneous mobile clients. In *Proceedings of the 27th Annual International Conference on Mobile Computing and Networking*, 2021.

[23] X. Li, K. Huang, W. Yang, S. Wang, and Z. Zhang. On the convergence of fedavg on non-iid data. In *ICLR*, 2020.

[24] Y. Li, Y. Zeng, and S. Banerjee. Enabling wideband, mobile spectrum sensing through onboard heterogeneous computing. In *Proceedings of the 22nd International Workshop on Mobile Computing Systems and Applications*, pages 85–91, 2021.

[25] B. McMahan, E. Moore, D. Ramage, S. Hampson, and B. A. y Arcas. Communication-efficient learning of deep networks from decentralized data. In *Artificial Intelligence and Statistics*, pages 1273–1282. PMLR, 2017.

[26] F. Mo, H. Haddadi, K. Katevas, E. Marin, D. Perino, and N. Kourtellis. Ppfl: Privacy-preserving federated learning with trusted execution environments. In *Proceedings of the 19th Annual International Conference on Mobile Systems, Applications, and Services*, MobiSys '21, page 94–108, New York, NY, USA, 2021. Association for Computing Machinery.

[27] S. Nakamoto. Bitcoin whitepaper. *URL: https://bitcoin. org/bitcoin. pdf*, 2008.

[28] A. Nika, Z. Zhang, X. Zhou, B. Y. Zhao, and H. Zheng. Towards commoditized real-time spectrum monitoring. In *Proceedings of the 1st ACM Workshop on Hot Topics in Wireless*, pages 25–30, 2014.

[29] C. Niu, F. Wu, S. Tang, L. Hua, R. Jia, C. Lv, Z. Wu, and G. Chen. Billion-scale federated learning on mobile clients: a submodel design with tunable privacy. In *Proceedings of the 26th Annual International Conference on Mobile Computing and Networking*, pages 1–14, 2020.

[30] X. Ouyang, Z. Xie, J. Zhou, J. Huang, and G. Xing. Clusterfl: a similarity-aware federated learning system for human activity recognition. In *Proceedings of the 19th Annual International Conference on Mobile Systems, Applications, and Services*, pages 54–66, 2021.

[31] A. Paszke, S. Gross, F. Massa, A. Lerer, J. Bradbury, G. Chanan, T. Killeen, Z. Lin, N. Gimelshein, L. Antiga, et al. Pytorch: An imperative style, high-performance deep learning library. *Advances in neural information processing systems*, 32:8026–8037, 2019.

[32] Y. Qi, M. S. Hossain, J. Nie, and X. Li. Privacy-preserving blockchain-based federated learning for traffic flow prediction. *Future Generation Computer Systems*, 117:328–337, 2021.

[33] S. Rajendran, R. Calvo-Palomino, M. Fuchs, B. Van den Bergh, H. Cordobés, D. Giustiniano, S. Pollin, and V. Lenders. Electrosense: Open and big spectrum data. *IEEE Communications Magazine*, 56(1):210–217, 2017.

[34] S. Rajendran, W. Meert, D. Giustiniano, V. Lenders, and S. Pollin. Deep learning models for wireless signal classification with distributed low-cost spectrum sensors. *IEEE Transactions on Cognitive Communications and Networking*, 4(3):433–445, 2018.

[35] S. Rajendran, W. Meert, V. Lenders, and S. Pollin. Saife: Unsupervised wireless spectrum anomaly detection with interpretable features. In *2018 IEEE International Symposium on Dynamic Spectrum Access Networks (DySPAN)*, pages 1–9. IEEE, 2018.

[36] B. Reynders, F. Minucci, E. Perenda, H. Sallouha, R. Calvo-Palomino, Y. Lizarribar, M. Fuchs, M. Schäfer, M. Engel, B. Van den Bergh, et al. Skysense: terrestrial and aerial spectrum use analysed using lightweight sensing technology with weather balloons. In *Proceedings of the 18th International Conference on Mobile Systems, Applications, and Services*, pages 352–363, 2020.

[37] S. Sarkar, M. Buddhikot, A. Baset, and S. K. Kasera. Deepradar: a deep-learning-based environmental sensing capability sensor design for cbrs. In *Proceedings of the 27th Annual International Conference on Mobile Computing and Networking*, pages 56–68, 2021.

[38] L. Shi, P. Bahl, and D. Katabi. Beyond sensing: Multi-ghz realtime spectrum analytics. In *12th {USENIX} Symposium on Networked Systems Design and Implementation ({NSDI} 15)*, pages 159–172, 2015.

[39] T. M. Taher, R. B. Bacchus, K. J. Zdunek, and D. A. Roberson. Long-term spectral occupancy findings in chicago. In *2011 IEEE international symposium on dynamic spectrum access networks (DySPAN)*, pages 100–107. IEEE, 2011.

[40] M. H. ur Rehman, K. Salah, E. Damiani, and D. Svetinovic. Towards blockchain-based reputation-aware federated learning. In *IEEE INFOCOM 2020-IEEE Conference on Computer Communications Workshops (INFOCOM WKSHPS)*, pages 183–188. IEEE, 2020.

[41] C. Xie, S. Koyejo, and I. Gupta. Asynchronous federated optimization. *arXiv preprint arXiv:1903.03934*, 2019.

[42] S. Yin, D. Chen, Q. Zhang, M. Liu, and S. Li. Mining spectrum usage data: a large-scale spectrum measurement study. *IEEE Transactions on Mobile Computing*, 11(6):1033–1046, 2012.

[43] X. Ying, J. Zhang, L. Yan, G. Zhang, M. Chen, and R. Chandra. Exploring indoor white spaces in metropolises. In *Proceedings of the 19th annual international conference on Mobile computing & networking*, pages 255–266, 2013.

[44] Y. Zeng, R. Calvo-Palomino, D. Giustiniano, G. Bovet, and S. Banerjee. Adaptive uplink data compression in spectrum crowdsensing systems. In *2021 IEEE International Symposium on Dynamic Spectrum Access Networks (DySPAN)*, Dec. 2021.

[45] Y. Zeng, V. Chandrasekaran, S. Banerjee, and D. Giustiniano. A framework for analyzing spectrum characteristics in large spatio-temporal scales. In *Proceedings of the 25th Annual International Conference on Mobile Computing and Networking*, pages 1–16, 2019.

[46] T. Zhang, N. Leng, and S. Banerjee. A vehicle-based measurement framework for enhancing whitespace spectrum databases. In *Proceedings of the 20th annual international conference on Mobile computing and networking*, pages 17–28. ACM, 2014.

[47] T. Zhang, A. Patro, N. Leng, and S. Banerjee. A wireless spectrum analyzer in your pocket. In *Proceedings of the 16th International Workshop on Mobile Computing Systems and Applications*, pages 69–74. ACM, 2015.

[48] Y. Zhao, J. Zhao, L. Jiang, R. Tan, D. Niyato, Z. Li, L. Lyu, and Y. Liu. Privacy-preserving blockchain-based federated learning for iot devices. *IEEE Internet of Things Journal*, 8(3):1817–1829, 2020.

[49] M. Z. Zheleva, R. Chandra, A. Chowdhery, P. Garnett, A. Gupta, A. Kapoor, and M. Valerio. Enabling a nationwide radio frequency inventory using the spectrum observatory. *IEEE Transactions on Mobile Computing*, 17(2):362–375, 2017.