# iFLBC: On the Edge Intelligence Using Federated Learning Blockchain Network

Ronald Doku and Danda B. Rawat
Data Science and Cybersecurity Center (DSC$^2$), EECS Department
Howard University, Washington, DC 20059, USA
rdoku@bison.howard.edu.com, danda.rawat@howard.edu

***Abstract*—Lately there has been an increase in the number of Machine Learning (ML) and Artificial Intelligence (AI) applications ranging from recommendation systems to face to speech recognition. At the helm of the advent of deep learning is the proliferation of data from diverse data sources ranging from Internet-of-Things (IoT) devices to self-driving automobiles. Tapping into this unlimited reservoir of information presents the problem of finding quality data out of a myriad of irrelevant ones, which to this day, has been a significant issue in data science with a direct ramification of this being the inability to generate quality ML models for useful predictive analysis. Edge computing has been deemed a solution to some of issues such as privacy, security, data silos and latency, as it ventures to bring cloud computing services closer to end-nodes. A new form of edge computing known as edge-AI attempts to bring ML, AI, and predictive analytics services closer to the data source (end devices). In this paper, we investigate an approach to bring edge-AI to end-nodes through a shared machine learning model powered by the blockchain technology and a federated learning framework called iFLBC edge. Our approach addresses the issue of the scarcity of relevant data by devising a mechanism known as the Proof of Common Interest (PoCI) to sieve out relevant data from irrelevant ones. The relevant data is trained on a model, which is then aggregated along with other models to generate a shared model that is stored on the blockchain. The aggregated model is downloaded by members of the network which they can utilize for the provision of edge intelligence to end-users. This way, AI can be more ubiquitous as members of the iFLBC network can provide intelligence services to end-users.**

## I. Introduction

At the helm of the advent of deep learning is the proliferation of big data generated from IoT devices to autonomous cars to smart critical infrastructure. The sudden furtherance in AI that has provided various advantages in the lives of people [1] is a direct consequence of this explosion of data in recent times. Consequently, this has made the vision to make AI more ubiquitous a bit attainable but still with more work to be done. The main obstacle that hinders the omnipresence of AI to every person and every organization can be attributed to the inability of cloud computing to accommodate the vast amounts of data that continues to be generated by devices [2]. At its very core, cloud computing aims to transfer jobs to a remote network of powerful servers. These jobs usually involve the transfer of data for storage, management, or computation. A significant factor responsible for the intricacies associated with cloud computing is the heterogeneity of the data sources, which ranges from massive data centers to a wide variety of end-nodes (mobile phones, sensors, laptops) coupled with the fact that they tend to be far away from the cloud. Consequentially, cloud computing faces issues such as network latency, efficiency, and the cost of transferring extensive data to a remote server. These issues have necessitated the rise of edge computing and its variants as a vital tool in the quest to provide better services to clients. Edge computing [3], [4] entails pushing the data for analysis closer to the source as it is a less expensive undertaking, especially when moving this across a wide area network to a remote server.

Furthermore, uploading the data over a long distance can expose the data to security issues (eavesdropping, Man in the Middle attack, etc.) [5]. Edge computing endeavors to eliminate some of the risks of these attacks as it involves shortening the distance the data travels. However, this still does not present a foolproof solution to data security. A more dependable solution is to get the data to stay on the device for processing. Federated learning [6] has been proposed by Google researchers in an attempt to ensure data privacy in a distributed machine learning over multiple devices setting. In federated learning, the data does not have to leave the data source. The code comes to the data instead, which makes it a reliable way of optimizing privacy issues. Wang et al [7] proposed a framework they term In-Edge AI that combines edge and federated learning by employing the collaboration between devices through federated learning.

Conventionally, ML modeling and analysis happens in the cloud, but the need for real-time information is a consequence of the advancements of smart cities, self-driving vehicles, and the IoT. These have rendered the back and forth transfer of data to the cloud for analysis inefficient and outdated. Consequently, this has compelled a new form of edge computing known as edge intelligence. In edge intelligence, the goal is to bring ML, AI, and predictive analytics services closer to the data generator in an attempt to distribute computing intelligence over the network. This is beneficial as it brings intelligence closer to the data source as opposed to the laborious task of sending data to the cloud for analysis. The Gartner Hype Cycle foresees edge intelligence to play a significant role in the next decade [8]. However, work in [9] highlights the various factors that limit the widespread adoption of edge intelligence. These limitations are the cost associated with the bandwidth of transmitting vast amounts of data, the latency associated with cloud/edge services, reliability of the edge server, and the privacy of the data. In this work, we present an approach that aims to address these issues.

In our approach, we devise a solution that attempts to deliver fast and reliable edge intelligence to users. The provision of this edge intelligence service is achieved through the creation of a network of nodes that incorporates federated learning with blockchain, called iFLBC edge. The novelty in our approach lies in the employment of the PoCI mechanism that ensures that the data used to train models in the network is trustworthy and relevant. This addresses the ongoing problem of finding useful data currently plaguing data science [10]. We consider this undertaking to be a necessity because, for predictive analysis to be truly effective, the need for quality data is imperative. We achieve our goal of making AI more ubiquitous by incorporating federated learning, the blockchain, and Attribute-Based Encryption (ABE). The sections below describe the role these technologies play in our system. The result section reinforces the fact that our goal of getting AI services closer to the end-user becomes more achievable when the network scales.

## II. Background & Related Work

### A. Blockchain

Bitcoin instigated the widespread adoption of the blockchain when it highlighted its significance by solving the double-spending problem in digital currency back in 2009 [11]. The blockchain ensured data integrity and validity through a computational process known as mining. Through the mining process, a new block gets appended to the current blockchain. Mining involves the process of solving a computationally-intensive cryptographic puzzle known as the proof-of-work (PoW) for creating new blocks. The new block gets appended if the majority of nodes involved in the PoW process reach a consensus. Consensus is reached when the network approves the mined block as legitimate.

The blockchain technology has many challenges. Scalability issues is one such challenge that has garnered the attention of researchers in the blockchain domain [12]. An approach devised to solve this predicament is *sharding*. Sharding in simple terms can be explained as dividing a blockchain network into multiple teams [13]. Each team is referred to as a shard. Each shard has its own ledger and can validate transactions [14]. By splitting the network in this fashion, the network's efficiency is enhanced. These shards collaborate in parallel to maximize the performance of the network [15]. In [16], they propose a new sharding technique that divides the network into teams they call an Interest Group (IG). In our network, each IG has its own unique distributed ledger that stores an aggregated shared model.

### B. Federated Learning

Federated Learning [6] falls under a branch of computer science coined recently as Decentralized AI. In federated learning, machine learning models are trained on the device that generated the data. The training data stays at the source and never goes anywhere. This solves the privacy concerns associated with individuals not willing to share data. In the centralized manner of data training, the training data is collected from the device and transferred to a centralized data center. This is privacy-intrusive as it allows big companies such as the GAFA (Google, Apple, Facebook and Amazon) to collect data on users, which they derive insights from to build personalized machine learning models aimed at providing tailored services to consumers. Federated learning follows a decentralized approach where various data owners can collaborate to learn a model while keeping all the personal data private. This implies there is a shared intelligence among the nodes in the network.

Federated Learning plays a role in our system by gathering local updates of Potential Members (PM) and then generating an averaged global model update that is shared by members of a shard. The shared model is an aggregation of the intelligence derived from the local data of members that belong to that IG. The shared model is stored on a blockchain that is unique to each IG. Members of an IG can download the shared model which can be used to provide edge intelligence to clients that request it.

### C. Proof of Common Interest (PoCI)

The work in [16] proposed a network that aims to determine relevant data using the PoCI. In this section, we build on the work done in [16] as the service we intend to provide is maintained by such a network. The network is divided into teams known as IGs. Each IG has a shared model that is accessible to each member of that IG. The reputation of an IG depends on how competent the model that it owns is, which is a direct consequence of how relevant the data that a model was trained on is. As such, the data of a potential member (PM) must be vetted before its model is aggregated with the other members' model and appended to the blockchain. This vetting process is done by the PoCI. The PoCI process requires that nodes accepted to the IG must share similar interests with the other members of the IG. This is enforced by members of the IG that are randomly selected to perform the PoCI. The PoCI process involves the selected nodes solving a small computational work where they verify that a PM's data aligns with the interests of the IG. The PM is accepted into the IG and it's model is added to the blockchain ledger for that IG when the nodes performing the PoCI come to a consensus that the PM's data is of relevance to the IG.

## III. System Model

### A. PoCI Computation

The PoCI process demands the calculation of a unique hash function known as the MinHash of the PM's data. The first step in the PoCI process is a PM sending a request to be a member of the IG. The IG then sends a Latent Dirichlet Allocation (LDA) [17] model to the PM which it downloads. This topic modeling model is used to discover topics from the PM's data. Topic models require that the words that are used as input for the model are relevant. The dataset the model utilizes demands that it is split into suitably sized batches as this plays a vital role in determining the context in which the words are connected. Based on experiments, we discovered 300 samples (words in a dataset) perform better. As such, we require the PMs to have data that has been set to a limit of 300 words. Datasets that go over the limit are split into batches (300 samples per batch). The LDA model tends to get better as more data is added.

The first step is a data pre-processing phase which involves filtering words based on stop lists and lemmatization. We get a list of words from the dataset using TF-IDF term frequency. The list of words derived is used as the model's input. The TF-IDF algorithm is used to estimate the relevance of a keyword in a document. A score is then assigned to that keyword based on the number of times it appears in the document. This algorithm generates a list of words that can be used as input. Equation 1 below is the TF-IDF formula.

$$w_{i,j} = tf_{i,j} \times log\frac{N}{df_i} \tag{1}$$

In equation 1, $tf_{i,j}$ is the number of occurrences of the word $i$ in the document $j$, $df_i$ is the number of documents containing $i$, and $N$ is the total number of documents.

In the LDA model, the observable features the model sees are the words that appear in the documents. Other parameters are hidden/latent (inferred). One of those parameters is a topic that is assigned to each word, thus making every document a mixture of such topics. The goal of the model is to figure out how such document collection could have been generated in the first place. LDA produces a file that contains all the topics consisting of words with the probabilities of them belonging to that topic. The number of topics we get is dependent on the number of IGs present in the network. As such, topics are generated based on the names of the IGs. The result of the LDA model is a file that tells us the appropriate IG (label) the words in the dataset belong to. This process confirms whether the PM's data actually belongs to the IG it has requested to be a member of. If the LDA model does not produce the IG the PM requested, the PM's request is canceled.

LDA functions under the premise that a document constitutes various topics, i.e., $P(z|d)$. Each topic is a distribution over terms in a vocabulary, i.e., $P(t|z)$. LDA suggests documents are probability distributions over latent topics, and topics are probability distributions over words. LDA can be formally written as $P(t_i|d) = \sum_{j=1}^{z} P(t_i|z_i = j)P(z_i = j|d)$, where $P(t_i|d)$ can be expressed as the probability of the $i^{th}$ term for a given document $d$ and $z_i$ is the latent topic. $P(t_i|z_i = j)$ is also expressed as the probability of $t_i$ within topic $j$. Furthermore, $P(z_i = j|d)$ is the likelihood of selecting a term from the topic $j$ in the document. The number of latent topics $Z$, has to be defined in advance; this allows for the adaption of the level of specialization of the latent topics. LDA approximates the topic–term distribution $P(t|z)$ and the document–topic distribution $P(z|d)$ from an unlabeled Corpus of documents using Dirichlet priors for the distributions and a fixed number of topics. Gibbs sampling is one viable method to achieve this: It iterates multiple times over each term $t_i$ in document $d_i$, and samples a new topic $j$ for the term based on the probability $P(z_i = j|t_i, d_i, z_{-i})$ based on (6), until the LDA model parameters converge.

$$P(z_i = j|t_i, d_i, z_{-i}) \propto \frac{C^{TZ}_{t_{ij}} + \beta}{\sum_t C^{TZ}_{t_{ij}} + T\beta} \frac{C^{DZ}_{d_{ij}} + \alpha}{\sum_z C^{DZ}_{d_{iz}} + Z\alpha} \tag{2}$$

where $C^{TZ}$ keeps a tally of all topic–term assignments, $C^{DZ}$ counts the document–topic assignments, $z_{-i}$ represents all topic–term and document–topic assignments except the current assignment $z_i$ for term $t_i$, and $\alpha$ and $\beta$ are the hyperparameters for the Dirichlet priors, working as smoothing parameters for the counts. Based on the counts the posterior probabilities in (6) can be estimated as

$$P(t_i|z_i = j) = \frac{C^{TZ}_{t_{ij}} + \beta}{\sum_t C^{TZ}_{t_{ij}} + T\beta} \tag{3}$$

$$P(t_i|z_i = j|d_i) = \frac{C^{DZ}_{d_{ij}} + \alpha}{\sum_z C^{DZ}_{d_{iz}} + Z\alpha} \tag{4}$$

After the PM obtains the resulting file generated by the LDA model, the next step is to select the words associated with the IG (topic) it was assigned to. This set of words is used to calculate the MinHash of the dataset. It is through this MinHash that the PoCI is computed. To calculate the MinHash, we employ data mining methods such as shingling and Jaccard Similarity. The shingling of documents involve viewing a document as a set of short strings. In this manner, documents that share common sub-strings are perceived as similar. Shingling solves this by transforming a document into multiple sub-strings of length $k$ that is present within the document. As such, documents are represented as a set of k-shingles. The length $k$ needs to be picked according to the size of the document. We pick a shingle size of $k = 4$. After picking the shingle size, we introduce MinHashing. Like other hashing techniques, MinHashing works by converting a document of any size into a specific size. However, MinHashing can specifically return a fixed-size numeric signature for documents. We can use this numeric signature to calculate the similarities between the two documents. This is done through the help of Jaccard Similarity. We can find the similarity between two documents A and B by performing the Jaccard Similarity by discovering the relative size of their intersection. When documents are presented as a set of shingles, we can use the Jaccard Index to measure the similarity. The Jaccard Similarity can be applied to MinHashes as well. The Jaccard Similarity between the MinHashes of two documents A ($MH_A$) and B ($MH_B$) and is defined as

$$J(MH_A, MH_B) = \left.\frac{|MH_A \cap MH_B|}{|MH_A \cup MH_B|}\right| \tag{5}$$

MinHashing uses randomized algorithms to estimate the Jaccard Similarity between documents. The steps below show the MinHashing process:

Step 1: Break down the ledger into a set of shingles.
Step 2: Calculate the hash value for every shingle.

Step 3: Store the minimum hash value found in step 2.

Step 4: Repeat steps 2 and 3 with different hash algorithms 199 more times to get a total of 200 min hash values (MinHash signature).

To compute the PoCI, we need to find the similarities between the MinHash of the PMs and the members of the IG randomly selected to participate in the PoCI process (Approvers). Each PM is assigned an Approver. An Approver computes the PoCI by comparing its MinHash with the PM's MinHash. This is achieved by counting the number of signature components in which they match. That gives the similarity score for the comparison of any two documents. The formula for calculating the MinHash is expressed as:

$$h_\pi(C) = \min_\pi \pi(C) \tag{6}$$

where $C$ represents a document. If the Approvers confirm that the PM's data passed the PoCI, this indicates the PM has proved that it owns relevant data.

## B. Model Aggregation

For the federated learning portion of the algorithm, we follow the approach implemented in [18]. Let's assume a PM $A$ wants to be a part of the network. $A$'s local model is locally trained via stochastic variance reduced gradient algorithm utilizing the method illustrated in [19]. $A$'s datasize is split into 300 samples. $A$ computes a local model update using equation 1

$$w_i^{(t,\mathfrak{l})} = w_i^{(t-1,\mathfrak{l})} - \frac{\beta}{N_i}\left(\left[\nabla f_k(w_i^{t-1,\mathfrak{l}}) - \nabla f_k(w^{(\mathfrak{l})})\right]\right) + \nabla f_k(w^{\mathfrak{l}}) \tag{7}$$

f(w) is the loss function whose goal is to mininimize $w_i^{(t,l)}$. $f(W)$ is calculated as a mean squared error defined as $f(w) = \frac{1}{N_S}\sum_{i=1}^{ND}\sum_{s_k\in s_i} f_k(w) where f_k(w) = (x_k^T w - y_k)^2/2$. $N_i$ is the number of iterations (number of batches needed to complete an epoch), $\mathfrak{l}$ is the current epoch (an epoch represents the number of times the algorithm works through the dataset). $\beta$ is the step size.

The candidates then upload the local update: $(w^l, \{\Delta f_k(w^l)\} s_k \in s_i)$ and the MinHash of their data to their assigned Approvers where $w^l$ denotes the local weight. The Approvers broadcast the local model updates of their assigned PMs to the other members of the Approver set after they have performed PoCI by verifying the MinHash. The other Approvers must verify the MinHashes of the models that have been broadcast to them as well. If a PM's MinHash passes the PoCI, it's local model is included in the Approver's block until the Approver reaches the alloted blocksize. After this is done, an Approver executes the PoW. The first Approver to solve the PoW and generate a block broadcasts it to the other Approvers and all the members of the IG. The PMs then download the blockchain from their assigned Approvers. The PMs then go ahead to locally compute the global model update by utilizing the aggregated local model updates in the newly appended blockchain. After this is done, the PMs become members of the IG. They can themselves become future Approvers in the future. Every member of the IG gets an updated shared model as well by computing the global model update.

## IV. Proposed Approach

The goal of this work is to provide secure and reliable intelligence to client nodes through the aid of a distributed shared model. The initial step requires that a client node sends a request to the network. The sent request is an encrypted message describing the service needed. This request is encrypted using an Attribute-Based Encryption (ABE) scheme, which ensures the encrypted data gets decrypted by nodes in the network that belong to the IG the request was sent to. Each node in the network automatically gets assigned a set of attributes (belonging to an IG) when they become a member of an IG. Once the client node chooses the IG, the message/service request it sends to the network gets encrypted with the attributes of the IG the client chose. Each node in the network has a blockchain that contains an updated shared model, as explained earlier. The nodes that decrypt the data become the candidate nodes that will provide AI services to the client node. The criteria for determining the actual node that provides the intelligence depends on the distance of that node from the client node. The closest node to the source of the client becomes the designated intelligence provider. A connection is established, and the selected node then sends a secret key to the client node, which is used to decrypt the homomorphically encrypted predictions of the shared model.

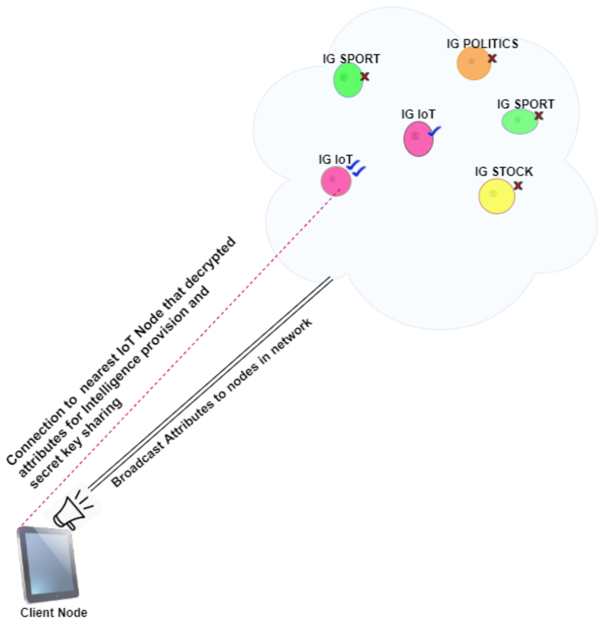
IG SPORT
IG POLITICS
IG IoT
IG SPORT
IG IoT
IG STOCK
Connection to nearest IoT Node that decrypted attributes for Intelligence provision and secret key sharing
Broadcast Attributes to nodes in network
Client Node

Figure 1. iFLBC Edge Intelligence Provisioning Process.

Figure 2 depicts the process of determining the intelligence provider. The blue marks represent the nodes that decrypted the request. The double blue marks depict the node closest to the client node, and thus the node that gets chosen to provide intelligence.

**Attribute-Based Encryption for Candidate Selection**: In this section, we describe in detail the process for the candidate selection. ABE is a public key encryption mechanism that allows for the encryption and decryption of messages based on the attributes of the receiver. ABE was first introduced by Sahai and Waters and is also known as Fuzzy-Identity based encryption (a variant of Identity Based Encryption [20]). The first application of ABE is in the area of using fingerprints in Identity Based Encryption. Fingerprints have peculiar characteristics that make it unique. For example, a fingerprint may have a set of 40 characteristics, and not all of these characteristics may match. Consequentially, this may fail to decrypt a message. The approach to address this was the Fuzzy -Identity Based encryption or ABE. In this approach, there was a threshold set in a manner that ensured a message got decrypted if $d$ many characteristics were matching in the fingerprint. ABE is based on bilinear maps which have the following properties:

**Symmetric bilinear map**: Let $p$ be a large prime number. $G_1$,$G_2$ are two groups of order $p$, and $g$ is a generator of $G_1$. $e : G_1 \times G_1 \rightarrow G_2$ is a symmetric bilinear map, that satisfies the following properties [**?**]:

1) Bilinear: For all $u,v \in G_1$ and $a_0,b_0, \in Z_p$, $e(u^{a_0},v^{b_0}) = e(u,v)^{a_0,b_0}$
2) Non-degenerate: $e(g,g) \neq 1$
3) Computable: There is an efficient algorithm to compute e(P,Q) for any $P,Q \in G_1$

We denote this bilinear map by $(p,\mathbb{G},g,\mathbb{G}_T,e)$ where g is a generator of $\mathbb{G}$.

Let U be the set of attributes assigned to a node based on its membership to an IG. Let $(p,\mathbb{G},g,\mathbb{G}_T,e)$ be the public parameters where $T_i$ are randomly chosen from the cyclic group. $\alpha$ is also randomly chosen from $\mathbb{Z}_p$, which later becomes the master key. Nodes in the network send their credentials $L$ to the Key Generation Center (KGC). The KGC then sends the secret key $(SK_L)$ back to the user.

- $(p,\mathbb{G},g,\mathbb{G}_T,e)$
- $U = \{att_1,att_2,...,att_n\} = \{1,2,3,...,n\}$
- $params = [p,g,e,g_2,hY = e(g,g_2)^{\alpha},T_1,T_2,...,T_n]$, where $\alpha \overset{R}{\leftarrow} \mathbb{Z}_p, g_2, h, T_i \overset{R}{\leftarrow} \mathbb{G}$
- $MK \equiv \alpha$
- $SK_L = [L, d_1 = g^r, d_2 = g_2^{\alpha}h^r, d_i = T_i^r, \forall_i \in L]$, where r $\overset{R}{\leftarrow}$ $\mathbb{Z}_P$

The equations below depict the encryption and decryption process of the message. The $msg$ is the ciphertext, $CT$. $SK_L$ is the secret key received from KGC upon sending the credentials of the nodes. It can be seen that If $\{i_1,i_2,...,i_k\}$ is not a subset of $L$, then the decryption fails. However, if it matches a set threshold, then the message gets decrypted. $I_K$ is the set threshold. If the attributes of the nodes meet the threshold set, then the decryption should take place. This process is shown below:

- Encrypt($params[p,g,e,g_2,h,Y$= $e(g,g_2)^{\alpha},T_1,T_2,...,T_n],W = i_1 \wedge i_2 \wedge ... \wedge i_k, msg \in \mathbb{G}_T$)
- $CT = [W,C_1,C_2,C_3]$ where $C_1 = msg.Y^s, C_2 = g^s, C_3 = (h\prod_{j=1}^{k} T_{i_j})^s, s \overset{R}{\leftarrow} \mathbb{Z}_p$
- Decrypt($params, SK_L = [L,d_1,d_2,d_i,\forall_i \in L], CT = [W,C_1,C_2,C_3]$)
  - If $\{i_1,i_2,...,i_k\} \not\subset$, decryption fails
  - If $\{i_1,i_2,...,i_k\} \subset L$, then $d = d_2\prod_{j=1}^{k} d_{i_j}$and $msg = \frac{C_1 \cdot e(d_1,C_3)}{e(d,C_2)}$

The nodes that can decrypt messages become the first set of delegate nodes that get selected to participate in the edge intelligence process. The primary service provider is selected using the Dijkstra's shortest path algorithm to find the closest node to the client. The other candidate nodes serve as a contingency plan in situations where the primary node goes offline. The second nearest node is expected to take over in such a scenario.

## V. Performance Evaluation & Security Analysis

In this section, we investigate the latency of our proposed iFLBC edge network. One of the foremost goals of this work is to aid in the quest to make AI more ubiquitous by making it more available to clients. We test this by conducting an extensive simulation. Our experiments were implemented in python and conducted on a workstation with Intel 3.40 GHz CPU and 32 GB RAM running windows operating system. We simulate a network of 5 IGs (IG IoT, IG Stock, IG Sports, IG Food, IG Politics) where various nodes are accepted to the network based on the data they own.

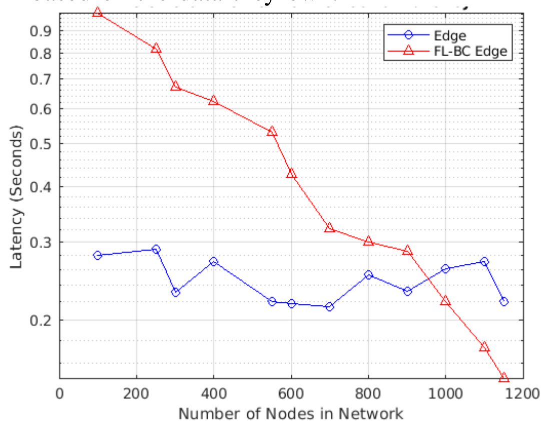

Figure 2. Impact of the number of nodes in the network on latency for proposed iFLBC and Vanilla edge.

In this simulation, we send client requests to the network with the aim of finding the appropriate node in the network to provide intelligence. We then record how long it takes to get back the result of a simple ML classification problem (intelligence). As explained earlier, nodes that decrypt the request become the candidate nodes that provide predictive services to the client node. The closest node to the client is then selected to provide this service. We compared our work to a regular edge network that has an edge server that provides similar predictions at a fixed location. From our simulations, we discover that the iFLBC edge performs faster than the vanilla edge network when the number of nodes in the network increases. This is because when the nodes in the network are considerably less, the distance between the client nodes and the selected intelligence provider tends to be higher. On the other hand, as more nodes get accepted to the network, the probability of a client node being closer to an intelligence provider gets higher. This result shows that our approach performs better as the network scales and thereby supports our goal of making AI more ubiquitous.

For security analysis, the network is potentially secure against Distributed Denial of Service attack because of the decentralized manner of data storage using federated learning and the blockchain. Federated Learning also aids in the prevention of data leakage. The federated learning approach of storing the local model updates on the blockchain makes the network less susceptible to the single point of failure attack. Moreover, since the global model can be computed by any node that has the current blockchain, it makes the network more reliable as nodes do not rely on a central source for global model updates. This is because the shortcomings of other nodes in the network do not in any way affect the computation of the global model updates. Furthermore, the network is regarded as a restrictive permissionless network as membership into the network demands that nodes only get accepted when the data they own are deemed to be relevant, thereby ensuring there is a vetting process that guarantees some level of security.

## VI. Conclusion

In this paper, we proposed an approach that aims to make AI more ubiquitous by providing nodes in a network with the necessary means to provide an intelligence service to client nodes. We show that our approach makes considerable effort to ensure the security and privacy of members in the network using the blockchain and federated learning. Our approach addresses the issue of the scarcity of relevant data by devising a mechanism known as the Proof of Common Interest (PoCI) that sieves out relevant data from irrelevant ones. We also incorporate Attribute-Based Encryption (ABE) to achieve our goal of making AI more ubiquitous. The result section further reinforces the fact that our goal of getting AI services closer to the end-user gets more achievable when the network scales.

## Acknowledgments

This work is partly supported by the U.S. NSF under grants CNS 1650831, IIS 1924092 and HRD 1828811, and by the U.S. Department of Homeland Security under grant DHS 2017-ST-062-000003 and DoE's National Nuclear Security Administration (NNSA) Award # DE-NA0003946. However, any opinion, finding, and conclusions or recommendations expressed in this document are those of the authors and should not be interpreted as necessarily representing the official policies, either expressed or implied, of the funding agencies.

## References

[1] J. Schmidhuber, “Deep learning in neural networks: An overview,” *Neural networks*, vol. 61, pp. 85–117, 2015.

[2] M. V. Barbera *et al.*, “To offload or not to offload? the bandwidth and energy costs of mobile cloud computing,” in *2013 Proceedings Ieee Infocom*, pp. 1285–1293, 2013.

[3] M. Patel *et al.*, “Mobile-edge computing introductory technical white paper,” *White paper, mobile-edge computing (MEC) industry initiative*, pp. 1089–7801, 2014.

[4] W. Shi, J. Cao, Q. Zhang, Y. Li, and L. Xu, “Edge computing: Vision and challenges,” *IEEE Internet of Things Journal*, vol. 3, no. 5, pp. 637–646, 2016.

[5] Q. Pu *et al.*, “Low latency geo-distributed data analytics,” *ACM SIGCOMM Computer Comm. Review*, vol. 45, no. 4, pp. 421–434, 2015.

[6] J. Konečnỳ *et al.*, “Federated learning: Strategies for improving communication efficiency,” *arXiv preprint arXiv:1610.05492*, 2016.

[7] X. Wang *et al.*, “In-edge ai: Intelligentizing mobile edge computing, caching and communication by federated learning,” *IEEE Network*, 2019.

[8] K. Panetta, “trends emerge in the gartner hype cycle for emerging technologies, 2018,” *Retrieved November*, vol. 4, p. 2018, 5.

[9] Y. Han, X. Wang, V. Leung, D. Niyato, X. Yan, and X. Chen, “Convergence of edge computing and deep learning: A comprehensive survey,” *arXiv preprint arXiv:1907.08349*, 2019.

[10] M. A. Waller and S. E. Fawcett, “Data science, predictive analytics, and big data: a revolution that will transform supply chain design and management,” *Journal of Business Logistics*, vol. 34, no. 2, pp. 77–84, 2013.

[11] D. B. Rawat, V. Chaudhary, and R. Doku, “Blockchain: Emerging applications and use cases,” *arXiv preprint arXiv:1904.12247*, 2019.

[12] Y. Sompolinsky and A. Zohar, “Accelerating bitcoin's transaction processing,” *Fast Money Grows on Trees, Not Chains*, 2013.

[13] J. C. Corbett *et al.*, “Spanner: Google's globally distributed database,” *ACM Transactions on Computer Systems*, vol. 31, no. 3, p. 8, 2013.

[14] L. Luu *et al.*, “A secure sharding protocol for open blockchains,” in *2016 ACM SIGSAC Conf on Computer & Comm Security*, pp. 17–30, 2016.

[15] M. Zamani, M. Movahedi, and M. Raykova, “Rapidchain: A fast blockchain protocol via full sharding,”

[16] R. Doku, D. B. Rawat, and C. Liu, “Towards federated learning approach to determine data relevance in big data,” in *2019 IEEE 20th International Conference on Information Reuse and Integration for Data Science (IRI)*, pp. 184–192, 2019.

[17] D. M. Blei, A. Y. Ng, and M. I. Jordan, “Latent dirichlet allocation,” *Journal of machine Learning research*, vol. 3, no. Jan, pp. 993–1022, 2003.

[18] H. Kim *et al.*, “Blockchained on-device federated learning,” *IEEE Communications Letters*, 2019.

[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] A. Sahai and B. Waters, “Fuzzy identity-based encryption,” in *Annual International Conference on the Theory and Applications of Cryptographic Techniques*, pp. 457–473, 2005.