Brainstorming Generative Adversarial Network (BGAN): Toward Multiagent Generative Models With Distributed Data Sets

Aidin Ferdowsi and Walid Saad, Fellow, IEEE

Abstract—To achieve a high-learning accuracy, generative adversarial networks (GANs) must be fed by large data sets that adequately represent the data space. However, in many scenarios, the available data sets may be limited and distributed across multiple agents, each of which is seeking to learn the distribution of the data on its own. In such scenarios, the agents often do not wish to share their local data as it can cause communication overhead for large data sets. In this article, to address this multiagent GAN problem, a novel brainstorming GAN (BGAN) architecture is proposed using which multiple agents can generate real-like data samples while operating in a fully distributed manner. BGAN allows the agents to gain information from other agents without sharing their real data sets but by “brainstorming” via the sharing of their generated data samples. In contrast to existing distributed GAN solutions, the proposed BGAN architecture is designed to be fully distributed, and it does not need any centralized controller. Moreover, BGANs are shown to be scalable and not dependent on the hyperparameters of the agents’ deep neural networks (DNNs) thus enabling the agents to have different DNN architectures. Theoretically, the interactions between BGAN agents are analyzed as a game whose unique Nash equilibrium is derived. Experimental results show that BGAN can generate real-like data samples with higher quality and lower Jensen–Shannon divergence (JSD) and Fréchet inception distance (FID) compared to other distributed GAN architectures.

Index Terms—Communication efficiency, distributed learning, generative adversarial networks (GANs).

I. INTRODUCTION

GENERATIVE adversarial networks (GANs) are deep neural network (DNN) architectures that can learn a data set distribution and generate realistic data points similar to this data set [1]. In GANs, a DNN called generator generates data samples while another DNN called discriminator tries to discriminate between the generator’s data and the actual data. The interaction between the generator and discriminator results in optimizing the DNN weights such that the generator’s generated samples look similar to the realistic data. Recently, GANs were adopted in several applications, such as image synthesis [2], anomaly detection [3], text to image translation [4], speech processing [5], and video generation [6].

Similar to many deep learning algorithms, GANs require large data sets to execute their associated tasks. Conventionally, such data sets are collected from the end users of an application and stored at a data center to be then used by a central workstation or cloud to learn a task. However, relying on a central workstation requires powerful computational capabilities and can cause large delays. On the other hand, such central data storage units are vulnerable to external attacks. Furthermore, in many scenarios, such as health and financial applications, the data sets are distributed across multiple agents (e.g., end users) who do not intend to share them. Such challenges motivate parallelism and the need for distributed, multiagent learning for GANs.

In a distributed learning architecture, multiple agents can potentially learn the GAN task in a decentralized fashion by sharing some sort of information with each other while minimizing the communication overhead. The goal of each agent in a distributed GAN would be to learn how to generate high-quality real-like data samples. Distributed GAN learning schemes also can also reduce the communication and computational limitations of centralized GAN models making them more practical for large-scale scenarios with many agents.

A. Related Works

For deep learning models, several distributed architectures have been proposed to facilitate parallelism using multiple computational units [7], [8], [9], [10], [11], [12]. Dean and Ghemawat [7] introduced the MapReduce architecture, in different agents aim at mapping the data into a new space reducing the data size. Moreover, GraphLab abstraction was proposed in [8] to facilitate graph computation across multiple workstations. Assran et al. [9] have proposed a stochastic gradient push for distributed deep learning that converges to a stationary point. In addition, Wen et al. [10] have proposed to use ternary gradients to accelerate distributed deep learning in data parallelism. Furthermore, model parallelism is introduced in [11], in which the different layers and weights of a DNN structure are distributed between several agents. Recently, federated learning (FL) was introduced in [12] as an
effective distributed learning mechanism that allows multiple agents to train a global model independently on their data set and communicate training updates to a central server that aggregates the agent-side updates to train the global model. However, the works in [7], [8], [9], [10], [11], and [12] as well as follow ups on FL focus on inference models and do not deal with generative models or GAN.

Recently, Hoang et al. [13], Durugkar et al. [14], Ghosh et al. [15], Hardy et al. [16], and Yonetani et al. [17] investigated the use of distributed architectures that take into account GAN’s unique structure which contains two separate DNNs (generator and discriminator). In [13] and [14], multiple generators or discriminators are used to stabilize the learning process but not to learn from multiple data sets. In [15], a single discriminator is connected to multiple generators in order to learn multiple modalities of a data set and to address the mode collapse problem. In [16], the notion of privacy preserving GAN agents was studied for the first time, using two architectures: 1) a multidiscriminator GAN (MDGAN) which contains multiple discriminators each located at every agent that owns private data and a central generator that generates the data and communicates its to each agent and 2) an adaptation of FL called FLGAN in which every agent trains a global GAN on its own data using a single per-agent discriminator and a per-agent generator and communicates the training updates to a central aggregator that learns a global GAN model. In [17], analogously to MDGAN, a forger-first update (F2U) GAN is proposed such that every agent owns a discriminator and a central node has a generator. However, unlike the MDGAN model, at each training step, the generator’s parameters are updated using the output of the most forgiving discriminator. Note that in the context of distributed learning privacy is defined as the right of controlling your own data and not sharing it with others. This is aligned with the common privacy definition of FL [12].

However, the works in [7], [8], [10], [11], [13], and [14] consider a centralized data set accessed by all of the agents. Moreover, the solutions in [12] and [16] can cause communication overhead since they require the agents to communicate, at every iteration, the DNN trainable parameters to a central node. Also, none of the GAN architectures in [13], [14], [15], [16], and [17] is fully distributed and they all require either a central generator or a central discriminator. In addition, the distributed GAN solutions in [13], [14], [15], [16], and [17] do not consider heterogeneous computation and storage capabilities for the agents.

B. Contributions

The main contribution of this article is the introduction of a novel brainstorming GAN (BGAN) architecture. This architecture allows agents to learn a data distribution in a fashion that is fully distributed, eliminating the need for a central controller. In the BGAN architecture, each agent, equipped with a single generator and a single discriminator, owns a unique data set. During training, agents share their ideas-generated data samples with their neighbors. This allows them to communicate information about their data set without sharing the actual data samples. As such, the proposed approach enables the GANs to collaboratively brainstorm in order to generate high-quality real-like data samples. This is analogous to how humans brainstorm ideas to come up with solutions. To the best of our knowledge, this is the first work that proposes a multiagent learning architecture for GANs that operates without a central controller, as mentioned earlier. In particular, the proposed BGAN has the following key features.

1) The BGAN architecture is fully distributed and does not require any centralized controller.
2) Compared to previous distributed GAN models, such as MDGAN, FLGAN, and F2U, our approach significantly reduces communication overhead.
3) It allows defining different DNN architectures for different agents depending on their computational and storage capabilities.

To characterize the performance of BGAN, we define a game between the BGAN agents and we analytically derive its Nash equilibrium (NE). We prove the uniqueness of the derived NE for the defined game. Moreover, we analyze each agent’s connection structure with neighboring agents and characterize the minimum connectivity requirements that enable each agent to gain information from all of the other agents. We compare the performance of our proposed BGAN with other state-of-the-art architectures, such as MDGAN, FLGAN, and F2U, and show that BGAN can outperform them in terms of Jensen–Shannon divergence (JSD), Fréchet inception distance (FID), and communication requirements besides the fact that, unlike the other models, BGAN is fully distributed and allows different DNN architectures for agents.

BGANs are particularly useful in practical, real-world scenarios where data owners wish to receive information from other data owners without sharing their own data. For instance, hospitals that own patients’ data, such as radiology images of patient lungs, may wish to generate synthetic images for future experiments. To improve the image generation process, they can use BGAN to receive information from other hospitals without sharing their own data sets. Similarly, financial firms, which often have scarce data sets that do not cover the entire data space, can use BGANs to learn better data representations without sharing their local data. Another key application is within the context of an Internet of Things. In particular, Internet of Things devices collect information from their users (e.g., location, voice, and heartbeats) that cannot be shared publicly, thus they can use BGANs for information flow and to learn better data space representations. Therefore, BGANs can be used in broad range of areas, such as medical applications, finance, personal activity, and the Internet of Things.

The remainder of this article is organized as follows. Section II describes the multiagent learning system model. In Section III, the BGAN architecture is proposed and the analytical results are derived. Experimental results are presented in Section IV and conclusions are drawn in Section V.

II. SYSTEM MODEL

Consider a set \( N \) of \( n \) agents, each represented as a node in Fig. 1. Every agent \( i \) owns a unique data set \( D_i \) which follows a distribution \( p_{\text{data}} \). We define a set \( D \), represented as the outer circle in Fig. 1, which is the union of all data sets \( D_1, D_2, \ldots \).
Fig. 1. BGAN architecture.

$D_n$. This total available data follows a distribution $p_{data}$, which is the target distribution that our model aims to mimic. It is worth noting that the connectivity pattern shown in Fig. 1 is a simplified representation for illustrative purposes. In practice, the connectivity can be any configuration as deemed suitable for the application. For each agent $i$, $p_{data}$ is a data distribution that does not span the entire data space as good as $p_{data}$. In this model, every agent $i$ tries to learn a generator distribution $p_{gi}$ over its available data set $D_i$ that can be close as possible to $p_{data}$. To learn $p_{gi}$ at every agent $i \in N$, we define a prior input noise $z$ with distribution $p_z(z)$ and a mapping $G_i(z, \theta_{gi})$ from this random variable $z$ to the data space, where $G_i$ is a DNN with a vector of parameters $\theta_{gi}$. For every agent $i \in N$, we also define another DNN called discriminator $D_i(x, \theta_{di})$ with vector of parameters $\theta_{di}$ that gets a data sample $x$ as an input and outputs a value between 0 and 1. When the output of the discriminator is closer to 1, then the received data sample is deemed to be real and when the output is closer to 0 it means the received data is fake. In our BGAN’s architecture, the goal is to find the distribution of the total data, $p_{data}$ under the constraint that no agent $i$ will share its available data set $D_i$ and DNN parameters $\theta_{di}$ and $\theta_{gi}$ with other agents. In contrast to MDGAN, FLGAN, and F2U, we will propose an architecture which is fully distributed and does not require any central controller. In BGANs, the agents only share their ideas about the data distribution with an idea being defined as the output of $G_i(z, \theta_{gi})$, at every epoch of the training phase with the other agents. Then, they will use the shared ideas to brainstorm and collaboratively learn the required data distribution.

While every agent’s generator DNN tries to generate data samples close to the real data, the discriminator at every agent aims at discriminating the fake data samples from the real data samples that it owns. Hence, we model these interactions between the generators and discriminators of the agents by a game-theoretic framework. For a standalone agent $i$ that does not communicate with other agents, one can define a zero-sum game between its generator and discriminator such that its local value function is [1]

$$\hat{V}_i(D_i, G_i) = \mathbb{E}_{x \sim p_{data}}[\log D_i(x)] + \mathbb{E}_{z \sim p_z}[\log(1 - D_i(G_i(z)))] \quad \text{(1)}$$

In (1), the first term forces the discriminator to produce values equal to 1 for the real data. On the other hand, the second term penalizes the data samples generated by the generators. Therefore, the agent’s generator aims at minimizing the value function while its discriminator tries to maximize this value. It has been proven in [1] that the NE for this game happens when $p_{gi} = p_{data}$ and $D_i = 0.5$. At the NE, the discriminator cannot distinguish between the generated samples and agent $i$’s real data. Although a standalone GAN can learn the representation of its own data set, if the owned data set at each agent is not representative of the entire data space, then the standalone agent will learn a distribution $p_{gi}$ that is not exactly the actual data representation. For instance, if an agent has a limited number of data samples that do not span the entire data space, then the learned distribution will be inaccurate [18]. In order to cope with this problem, we next introduce BGAN, in which every agent will only share their idea, i.e., their generated points with other neighboring agents without actually sharing their data set.

### III. BRAINSTORMING GENERATIVE ADVERSARIAL NETWORKS ARCHITECTURE

Let $N_i$ be the set of neighboring agents from whom agent $i$ receives ideas, let $\mathcal{O}$ be the neighboring agents to whom agent $i$ sends ideas, and let $\mathcal{G}$ be the directed graph of connections between the agents as shown in Fig. 1. Here, a neighboring agent for agent $i$ is defined as an agent that is connected to agent $i$ in the connection graph $\mathcal{G}$ via a direct link. For our BGAN architecture, we propose to modify the classical GAN value function in (1) into a *brainstorming value function* which integrates the received generated data samples (ideas) from other agents, as follows:

$$V_i(D_i, G_i, \{G_j\}_{j \in N_i}) = \mathbb{E}_{x \sim p_{bi}}[\log D_i(x)] + \mathbb{E}_{z \sim p_z}[\log(1 - D_i(G_i(z)))] \quad \text{(2)}$$

where $p_{bi}$ is a mixture distribution of agent $i$’s owned data and the idea that agent $i$ received from all neighboring agents. Formally, $p_{bi} = \pi_i p_{data} + \sum_{j \in N_i} \pi_j p_{bj}$, where $\pi_i + \sum_{j \in N_i} \pi_j = 1$. $\pi_i$ and $\pi_j$ represent, respectively, the importance of agent $i$’s own data and neighbor $j$’s generated data in the process of brainstorming. Such values can be assigned proportionally to the number of real data samples each agent owns since an agent having more data samples has more information about the data space. From (2), we can see that the brainstorming value functions of all agents in $N_i$ will be interdependent. Therefore, in order to find the optimal values for $G_i$ and $D_i$, we define a multiagent game between the discriminators and generators of agents. In this game, the generators collaboratively aim at generating real-like data to fool all of the discriminators while the discriminators try to
distinguish between the generated and real data samples of the generators. To this end, we define the total utility function as follows:

\[ V(D_i)=\sum_{i=1}^{n} V_i(D_i, G_i, \{G_j\}_{j \in N_i}). \]  

In our BGAN, the generators aim at minimizing the total utility function defined in (3), while the discriminators try to maximize this value. Therefore, the optimal solutions for the discriminators and generators can be derived as follows:

\[ \{D_i^*\}_{i=1}^{n}, \{G_i^*\}_{i=1}^{n} = \arg \min_{G_1,\ldots,G_n} \max_{D_1,\ldots,D_n} V(D_i,G_i,\{G_j\}_{j \in N_i}). \]  

where for notational simplicity we omit the arguments of \( V(D_1, \ldots, D_n, G_1, \ldots, G_n) \). In what follows, we derive the NE for the defined game and characterize the optimal values for the generators and discriminators. At such NE, none of the agents can get a higher value from the game if it changes its generator and discriminator while other agents keeping their NE generator and discriminators.

**Proposition 1:** For any given set of generators, \( \{G_1, \ldots, G_n\} \), the optimal discriminator is

\[ D_i^* = \frac{p_{bi}}{p_{bi} + p_{gi}}. \]  

**Proof:** For any given set of generators, \( \{G_1, \ldots, G_n\} \), we can derive the probability distribution functions for the generators, \( \{p_{bi}, \ldots, p_{gi}\} \). Thus, we can write the total utility function as

\[
V = \sum_{i=1}^{n} \left[ \int_x p_{bi}(x) \log D_i(x) dx + \int_z p_{zi}(z) \log(1 - D_i(G_i(z))) dz \right] = \sum_{i=1}^{n} \int_x \left( p_{bi}(x) \log D_i(x) + p_{gi}(x) \log(1 - D_i(x)) \right) dx
\]

where \( p_{bi} \) is the probability distribution of the generated data for agent \( i \) when discriminators reach an optimal state. Next, to find the maximum of (6) with respect to all of the \( D_i \) values, we can separate every term of the summation in (6) because each term contains \( D_i \) for a single agent \( i \). Thus, the optimal value, \( D_i^* \), is the solution of the following problem:

\[ D_i^* = \arg \max_{D_i} \int_x \left( p_{bi}(x) \log D_i(x) + p_{gi}(x) \log(1 - D_i(x)) \right) dx. \]  

In order to find the \( D_i \) that maximizes (7), we can find the value of \( D_i \) that maximizes the integrand of (7), which is given in (5).

Having found the maximizing values for the discriminators, we can move to the minimization part of (4). To this end, using (5), we can rewrite (4) as follows:

\[ G_1^*, \ldots, G_n^* = \arg \min_{G_1,\ldots,G_n} V(D_1^*, \ldots, D_n^*, G_1, \ldots, G_n). \]  

Now, we can express \( W \) as follows:

\[
W = \sum_{i=1}^{n} \left[ \mathbb{E}_x \log D_i^*(x) + \mathbb{E}_z \log(1 - D_i^*(G_i(z))) \right]
\]

where \( B \) is a matrix with element \( \pi_{ij} \) at every row \( i \) and column \( j \), \( C \) is a diagonal matrix whose \( i \)th diagonal element is \( \pi_{ii} \), \( p_g \triangleq [p_{g1}, \ldots, p_{gn}]^T \), and \( p_{data} \triangleq [p_{data1}, \ldots, p_{dataN}]^T \).

**Proof:** We can rewrite \( W \) as follows:

\[
W = \sum_{i=1}^{n} \left[ \mathbb{E}_x \log \left( \frac{p_{bi}}{p_{bi} + p_{gi}} \right) \right] + \mathbb{E}_z \log \left( \frac{p_{bi}}{p_{bi} + p_{gi}} \right)
\]

\[
= \sum_{i=1}^{n} \mathbb{E}_x \left[ \log \left( \frac{1}{2} \frac{p_{bi}^2 + p_{gi}^2}{p_{bi} + p_{gi}} \right) \right] + \mathbb{E}_z \log \left( \frac{1}{2} \frac{p_{bi}^2 + p_{gi}^2}{p_{bi} + p_{gi}} \right)
\]

\[
= -n \log(4) \sum_{i=1}^{n} \text{JSD}(p_{bi} || p_{gi})
\]

where \( \text{JSD}(p_{bi} || p_{gi}) \) is the JSD between \( p_{bi} \) and \( p_{gi} \) with a minimum at 0. Therefore, we can easily show that \( \sum_{i=1}^{n} \text{JSD}(p_{bi} || p_{gi}) \geq 0 \). Now, given that the minimum of \( W \) occurs when \( p_{gi} = p_{bi} \) for \( i \in N \), we will have

\[
p_{gi} = \pi_i p_{data} + \sum_{j \in N_i} \pi_{ij} p_{bj} \forall i \in N
\]

which can be simplified to (10) if we move the term \( \sum_{j \in N_i} \pi_{ij} p_{bj} \) to the left side of the equation. In this case, we will have \( D_i^* = [p_{gi}/(2p_{gi})] = (1/2) \). Thus, \( W \) will be simplified to

\[
W = \sum_{i=1}^{n} \left[ \mathbb{E}_x \log \left( \frac{1}{2} \right) \right] + \mathbb{E}_z \log \left( \frac{1}{2} \right)
\]

\[
= -n \log(4).
\]

By comparing (11) with (13), we can see that the solution of (10) yields \( \sum_{i=1}^{n} \text{JSD}(p_{bi} || p_{gi}) = 0 \), thus, minimizes \( W \).

**Theorem 1:** The defined game between the generators and discriminators of our BGANs has a unique NE where \( \forall i \in N, D_i = (1/2) \) and \( p_{gi} \) is the unique solution of (10).

**Proof:** Since we know \( \pi_i + \sum_{j \in N_i} \pi_{ij} = 1 \), then \( I - B \) is a diagonally dominant matrix. In other words, for every
row of \( I - B \), the magnitude of the diagonal element in a row is larger than or equal to the sum of the magnitudes of all the nondiagonal elements in that row. Therefore, using the Levy-Desplanques theorem [19] we can show that \( I - B \) is a nonsingular matrix and, thus, \( (10) \) always has a unique solution. Moreover, since the solution of \( (10) \) results in having \( \forall i \in N : \text{JSD}(p_{g_i}||p_{g_j}) = 0 \), that means it can minimize every term of the summation in \( (11) \), it is the unique minimum point of \( W \). Also, using (5) the solution of (10) yields \( D_i^* = (1/2) \) which completes the proof.

Theorem 1 plays a pivotal role in our approach, as it provides the conditions under which a unique NE can be found for the game defined in our model. This theorem is crucial in understanding the behavior and results of our approach, as it provides the mathematical basis for finding the optimal values for the generator’s distributions, which in turn leads to the unique NE.

In essence, Theorem 1 lays the foundation for our model by showing that the global minimum of \( W(G_1, \ldots, G_n) \) can be achieved at the solution of \( (10) \). This solution represents the optimal values for \( p_{g_i}^* \), which are the probability distributions for the generators.

By finding these optimal values, we can ensure that the generators produce data samples that are as close as possible to the real data samples. This, in turn, leads to a unique NE for the game, where the generators and discriminators are balanced, and neither can improve their outcome by unilaterally changing their strategy.

Therefore, Theorem 1 is instrumental in achieving our goal of creating a balanced and effective multiagent system where the agents collaboratively generate real-like data samples.

On the other hand, Corollary 1 shows that the defined game between the discriminators and generators has a unique NE. At this NE, the agents can find the optimal value for the total utility function defined in (4). However, one key goal of our proposed BGAN is to show that using the brainstorming approach each generator can integrate the data distribution of the other agents into its generator distribution. To this end, in the following, we prove that in order to derive a generator that is a function of all of the agents’ data sets, the graph of connections between the agents, \( G \), must be strongly connected.

Definition 1: An agent \( i \) can reach an agent \( j \) (agent \( j \) is reachable from agent \( i \)) if there exists a sequence of neighbor agents which starts with \( i \) and ends with \( j \).

Definition 2: The graph \( G \) is called strongly connected, if every agent is reachable from every other agent.

Theorem 2: BGAN agents can integrate the real-data distribution of all agents into their generator if their connection graph, \( G \), is strongly connected.

Proof: From Corollary 1, we know that \( (10) \) will always admit a unique solution, \( p_{g_i}^* \). In order to derive this solution, we can use the iterative Jacobi method. In this method, for an initial guess on the solution \( p_{g_i}^{(0)} \), the solution is obtained iteratively via

\[
p_{g_i}^{(k+1)} = I^{-1} \left( CP_{\text{data}} + Bp_{g_i}^{(k)} \right)
\]

where \( p_{g_i}^{(k)} \) is the \( k \)th approximation on the value of \( p_{g_i}^* \). Letting \( p_{g_i}^{(0)} = 0 \), we will have

\[
p_{g_i}^{(k+1)} = \left( \sum_{m=1}^{k} B^m + I \right) CP_{\text{data}}.
\]

Therefore, we have

\[
p_{g_i}^* = \lim_{k \to \infty} p_{g_i}^{(k+1)} = \left( \sum_{m=1}^{\infty} B^m + I \right) CP_{\text{data}}.
\]

From (18), we can see that in order to have \( p_{g_i}^* \) as a function of \( p_{\text{data}}, \) there should be a \( B^m \) whose entry in row \( i \) and column \( j \) is nonzero. This entry of \( B^m \) is nonzero if \( i \) is reachable from \( j \) via \( m \) steps in the graph \( G \). Therefore, in order to receive information from all of the agent data sets, every agent must be reachable from every other agent which completes the proof.

Theorem 2 shows that, in order to have a BGAN architecture that enables the sharing of data (ideas) between all of the agents, \( G \) must be strongly connected. In this case, the optimal solution for the generator is a linear mixture of agents real data distribution as follows:

\[
p_{g_i}^* = \sum_{j \in N} \lambda_{ij} p_{\text{data}j}
\]

where \( \lambda_{ij} \) are some positive-valued constants that can be derived by solving \( (10) \). However, for the case in which \( G \) is not strongly connected, the following corollary (whose proof is similar to Theorem 2) shows how the information is shared between the agents.

Corollary 2: A BGAN agent can receive information from every other agent, if it is reachable from that agent.

Therefore, in BGANs, the agents can share information about their data set with every agent which is reachable in \( G \). In practice, agent \( j \) is reachable from agent \( i \) if there is a communication path on the connection graph from agent \( i \) to agent \( j \). Therefore, a BGAN agent can receive information (ideas) from every agent to which it is connected via a communication path. In this case, the optimal generator distribution of each agent \( i \) will be a mixture of the data distributions of the agents which can reach agent \( i \) in graph \( G \), written as

\[
p_{g_i}^* = a_ip_{\text{data}i} + \sum_{j \in \mathcal{N}_i} a_jp_{\text{data}j}, \quad \mathcal{N}_i \text{ is the set of agents that can reach agent } i \text{ and } a_i \text{ and } a_j \text{ come from the solution of } (10).
\]

In order to implement the BGAN architecture, one of the important steps is to integrate the mixture model for each agent in the brainstorming step. For a batch size \( b \) at each training episode, agent \( i \) receives \( \pi_i b \) generated samples from agent \( j \in \mathcal{N}_i \). This approach guarantees that agent \( j \) has \( \pi_i \) contribution in the brainstorming phase compared to other neighbor agents in \( \mathcal{N}_i \). To ensure all agents are in the same training round and have fresh shared ideas, each agent waits until it has received input from all its neighbors before starting training. This synchronization mechanism ensures concurrent progression through the training rounds, addressing concerns about data freshness. Algorithm 1 summarizes the steps needed to implement the proposed BGAN architecture. Our BGAN architecture enables a fully distributed learning scheme in which every agent can gain information from all of the other
**Algorithm 1 BGAN Training**

1. Initialize $D_i$ and $G_i$ for $i \in N$.
2. Repeat:
3. \hspace{1em} Parallel for $i \in N$:
4. \hspace{2em} Generate $b$ samples $\{y_i^{(1)}, \ldots, y_i^{(b)}\}$ using $G_i$ and $p_{z_i}$.
5. \hspace{2em} For $j \in O_i$:
6. \hspace{3em} Send $\pi_{ij}b$ points from the generated samples (ideas), $\{y_{ij}^{(1)}, \ldots, y_{ij}^{(\pi_{ij}b)}\}$, to agent $j$.
7. \hspace{2em} Wait until ideas from all neighbors are received.
8. \hspace{2em} Sample $\pi_{ib}$ data samples, $\{x_i^{(1)}, \ldots, x_i^{(\pi_{ib})}\}$, from $D_i$.
9. \hspace{2em} Update $\theta_{d_i}$ by ascending the following gradient:
   $$\nabla_{\theta_{d_i}} \frac{1}{b} \left( \sum_{j \in X_i} \pi_{ij} \log D_j(y_{ij}^{(k)}) + \sum_{k=1}^{\pi_{ib}} \log D_i(x_i^{(k)}) + \sum_{k=1}^{b} \log \left( 1 - D_i(y_i^{(k)}) \right) \right)$$
   \hspace{1em} (14)
10. \hspace{2em} Generate $b$ samples $\{y_i^{(1)}, \ldots, y_i^{(b)}\}$ using $G_i$ and $p_{z_i}$.
11. \hspace{2em} Update $\theta_{g_i}$ by descending the following gradient:
   $$\nabla_{\theta_{g_i}} \frac{1}{b} \sum_{k=1}^{b} \log \left( 1 - D_i(y_i^{(k)}) \right)$$
   \hspace{1em} (15)
12. Until convergence to the NE

agents without sharing their real data. Next, we showcase the key properties of BGAN by conducting extensive experiments.

**IV. EXPERIMENTS**

We empirically evaluate the proposed BGAN architecture on data samples drawn from multidimensional distributions as well as image data sets. Our goal here is to show how our BGAN architecture can improve the performance of agents by integrating a brainstorming mechanism compared to standalone GAN scenarios. Moreover, we perform extensive experiments to show the impact of architecture hyperparameters, such as the number of agents, number of connections, and DNN architecture. In addition we compare our proposed BGAN architecture with MDGAN, FLGAN, and F2U, in terms of the quality of the generated data as well as the communication resources.

**A. Data Sets**

For our experiments, we use two types of data samples. The first type which we call the ring data set contains 2-D samples $(r \cos \theta, r \sin \theta)$ where $r \sim \Gamma(\alpha, \beta)$, $\theta \sim U(0, 2\pi)$, and $\alpha$ and $\beta$ are chosen differently for multiple experiments. This data set constitutes a ring shape in 2-D space as shown in Fig. 2 and was used since 1) the generated points by GAN agents can be visually evaluated; 2) the dimensions of each sample has a nonlinear relationship between each other; and 3) the stochastic behavior of the data samples is known since they are drawn from a gamma distribution and a uniform distribution and, thus, the JSD between the actual data and the generated data samples can be calculated. We use JSD as the quality measure for the BGAN architecture since, as shown in [1] and Theorem 1, the JSD between the generated data samples and the actual data must be minimized at the NE. The second type of data sets contain the well-known MNIST, fashion MNIST and CIFAR-10 data sets [20], [21], [22]. These data sets are used to compare BGAN’s generated samples with the other distributed GAN architectures such MDGAN, FLGAN, and F2U.

**B. Implementation Details**

For every data set, we use a different DNN architecture depending on the complexity of the data set. For instance, for the ring data set we use two simple multilayer perceptrons with only two dense layers. However, for image data sets that are multi dimensional and are more complex than the ring data set we use multiple convolutional layers and a dense layer for the discriminator and multiple transposed convolutional layers for the generator. Note that our main goal is to show that our proposed BGAN architecture is fully distributed and communication efficient. Thus, we do not use sophisticated DNNs as in [23], [24], and [25]. However, since the proposed BGAN does not have any restrictions on the GAN architecture, architectures, such as those in [23], [24], and [25], can naturally be used to achieve higher accuracy in the generated data. In order to train our BGAN, we have used Tensorflow and 8 Tesla P100 GPUs which helped expediting the extensive experiments. We have used multiple batch sizes and the reported values are the ones that had the best performance. Moreover, we have distributed an equal number of samples among the agents and we assign equal values for $\pi_{ij}$ and $\pi_i$, unless otherwise stated.

**C. Effect of the Number of Agents and Data Samples**

GANs can provide better results if they are fed large data sets. Therefore, using the ring data set, we try to find the minimum number of training samples that is enough for a
standalone GAN to learn the distribution of the data samples and achieve a minimum JSD from the data set. From Fig. 3, we can see that the JSD remains constant after 1000 data samples and, hence, in order to showcase the benefits of brainstorming, we will assign less than 1000 samples to each agent to see if brainstorming can reduce the JSD.

For implementing BGAN, we consider a connection graph in which each agent receives data (idea) only from one neighbor and the graph is strongly connected as shown in Fig. 4. We implement BGAN with two to ten agents with 10 to 10000 data samples for each agent. Fig. 5 shows the JSD of generated points from the actual data set for BGAN agents and for a conventional standalone (single) GAN agent for a different number of data samples. Fig. 5 shows that, by increasing the number of data samples, the JSD decreases and reaches its minimum value. More importantly, BGAN agents can compensate the lack of data samples that occurs in a standalone GAN by brainstorming with other agents. For instance a standalone agent with ten data samples has a JSD of 24, however, when the same agent participates in the brainstorming process with nine other agents (ten agents in total), then, it can achieve a JSD of 13.

Next, in Fig. 6, we show the generated points by standalone and BGAN agents for different numbers of available data samples. When an agent has access to a small data set, the GAN parameters will be underfitted, however, brainstorming can still, in some sense, increase the size of data set by adding the generated samples of neighboring agents into the training set. Therefore, as seen from Fig. 6, by participating in the brainstorming process, a BGAN agent with a limited data set can generate data samples that are closer to the actual data distribution in Fig. 2. This demonstrates that our BGAN architecture can significantly improve the learning and generation performance of agents.
Note that, no distributed GAN agent can perform better than a standalone GAN that has access to all data samples. The reason that we compare BGAN agents with a standalone GAN agent with the same number of data samples is to show how BGAN can help the flow of information between BGAN agents and confirm our theorems. For example, a standalone GAN with 1000 data samples will have a better performance than ten BGAN agents with 100 data samples each since, in this case, BGAN agents have access to 90% less real data samples than a standalone GAN, but they try to compensate that lack of data samples by communicating their generated ideas with neighboring agents. Meanwhile, ten BGAN agents each having 100 data samples have a higher performance on average than ten standalone GAN agents (who cannot communicate) having 100 samples each. Therefore, a BGAN agent’s average performance is lower bounded by a standalone GAN having the same number of data samples as a single BGAN agent and upper bounded by a standalone GAN having the total number of samples for all BGAN agents. This is a result that has not been shown for other state-of-the-art distributed GANs which is important since it quantifies performance bounds of distributed GAN agents.

**D. Effect of the Connection Graph**

In order to study the effect of the connection graph $G$ on the performance of BGAN agents we consider two scenarios: 1) a strongly connected graph in which every agent has an equal number of neighbors as shown in Fig. 7 and 2) a string such that all of the agents have only one neighbor except one agent, agent $n$, that does not have any neighbors but sends data to one other agent as shown in Fig. 8.

In the first scenario, we consider ten agents and we implement the BGAN architecture while varying the number of neighbors for each agent between one to nine. We also consider 10, 20, 50, 100, and 1000 data samples for each agent. Fig. 9 shows that for cases in which the number of samples is too small (10, 20, and 50) having more neighbors reduces the average JSD for the BGAN agents. However, when the number of samples is large enough adding more neighbors does not affect the JSD. We explain this phenomenon from (10) and (19), where the optimal generator distribution is shown to be related to the graph structure. For instance, Table I shows the dependence of each agent’s generator distribution on all of the agents’ data sets for the cases with 1 and 9 neighbors. From Table I, we can see that, for the 1-neighbor scenario, as an agent gets farther away from another agent, it will less affect the other agent’s generator distribution since $\lambda_{ij}$ decreases when $\text{mod}(i - j, 10)$ increases. However, for a 9-neighbor scenario, $\lambda_{ij}$ stays constant for $\text{mod}(i - j, 10) > 0$. Therefore, when the agents have a small data set and have a higher number of neighbors, they can gain information almost equally from other agents and can span all the data space. As such, as shown in Fig. 9, the JSD of the agent which owns only ten data samples gets better when the number of neighbors increases. However, when the number of data samples is large then $p_{\text{data}} \simeq p_{\text{data}}$ for all $i \in \mathcal{N}$. In this case, $p_{\text{data}} \simeq p_{\text{data}}$ for $i, j \in \mathcal{N}$ and, thus, from (19) we will have $p_{g_i} \simeq p_{\text{data}}$ irrespective of the graph structure.

In the second scenario, we again consider ten agents. Now, agent 10 sends data to agent 9, agent 9 sends data to agent 8, and so on until agent 2 sends data to agent 1. However, unlike the first scenario, agent 1 does not close the loop and does not
send any information to agent 10. Fig. 10 shows the generated samples and JSD of every agent. From Fig. 10, we can see that the samples generated by agents 1 to 5 are close to the ring data set and they have a small JSD. However, as we move closer to the end of the string (agent 10), the generated samples diverge from the actual data distribution and the JSD increases. Therefore, in order for each agent to get information from the data sets of all of the agents, the graph should be strongly connected.

E. Effect of Nonoverlapping Data Samples

Next, we prove that the BGAN agents can learn nonoverlapping portions of the other agents’ data distributions. To this end, we consider two BGAN agents whereby agent 1 has access to data points only between $0^\circ$ to $\pi^\circ$ while agent 2 owns the data between $\pi^\circ$ to $2\pi^\circ$. Fig. 11 shows the available portions of the ring data set for each agent as well their generated points after brainstorming. As can be seen from Fig. 11, the agents can generate points from the ring data set that they do not own. This showcases the fact that brainstorming helps agents to exchange information between them without sharing their data sets. However, a standalone agent can at best learn to mimic the portion of data that it owns and cannot generate points from the data space that it does not have access to.

F. Effect of Different Architectures

We now show how the proposed BGAN can allow the different agents to have a different DNN architecture considering a BGAN with five agents whose DNNs differ only in the size of their dense layer. In other words, each agent has a different number of neurons. Moreover, the agents’ connection graph is similar to the one in Fig. 7 with one neighbor. In Fig. 12, we compare the output of the agents resulting from both the standalone and BGAN scenarios. Fig. 12 demonstrates that, in the standalone case, agents having denser DNNs will have a lower JSD compared to agents having a smaller number of
trainable parameters at the dense layer. However, by participating in brainstorming, all of the GAN agents reduce their JSD and improve the quality of their generated points. This allows agents with lower computational capability to brainstorm with other agents and improve their learned data distribution. Note that this capability is not possible with the other baselines, such as FLGAN.

G. Comparison With FLGAN, MDGAN, and F2U

In order to compare the generated points by BGAN agents with other state-of-the-art distributed GAN architectures, we use the ring and MNIST data set. For the ring data sets, we run experiments with two to ten agents where each agent owns only 100 data samples and their connection graph is similar to the one in Fig. 7 with one neighbor. We compare the average JSD of BGAN agents with the ones resulting from FLGAN, MDGAN, F2U, and standalone GAN agents. We consider $n$ samples for each standalone agent as the upper bound performance of distributed agents and 100 samples for each standalone agent as the lower bound performance indicator. In other words, no distributed GAN agent can perform better than a standalone GAN that has access to all data samples. On the other hand, distributed architectures should always have a lower JSD compared to a standalone agent with the same number of available data samples.

Fig. 13 shows the average JSD resulting from the various distributed GAN architectures. We can see from Fig. 13 that BGAN agents will always have a lower JSD compared to the other distributed GAN agents whereby for the two-agent case, BGAN can achieve a JSD as low as a standalone with 200 data samples. Furthermore, all distributed agents yield a better performance than a standalone agent with 100 samples, however, they cannot achieve a JSD lower than a standalone agent with $n \times 100$ samples. In addition, adding more agents reduces the JSD for MDGAN and F2U agents while the JSD of FLGAN and BGAN agents stay constant. For BGAN, we have already seen this fact in Fig. 5 where the JSD achieves a minimum value for a particular number of data samples such that adding more agents will not improve the performance of BGAN agents.

Furthermore, we compare the performance of BGAN agents with other distributed GANs using the MNIST, fashion MNIST, and CIFAR-10 data sets. In order to show the information flow between the agents, we consider ten agents each of which owns only images of a single class (a digit in MNIST data, a cloth in fashion MNIST, or an animal type in CIFAR-10 data sets). Fig. 14 shows the images generated by BGAN for the MNIST data set. From Fig. 14, we can see that all BGAN agents are able to not only generate digits similar to the data set that they own but they can also generate digits similar to their neighbors’ data sets. This is a valuable result since, similar to the experiments on the nonoverlapping data sets, we can see that BGAN can enable agents to transfer information among them while not sharing their data set.

In addition to MNIST data set, Figs. 15 and 16 show the generated samples by BGAN agents who have been fed by only fashion MNIST and CIFAR-10 images. In Figs. 15 and 16 each agent owns the images of only one class. From Figs. 15 and 16 we can see that using the BGAN architecture, each agent can generate the images that look like its own class as well as its neighbors’ classes. These results indicate that the BGAN architecture enables the agents to flow the information regarding their data set without sharing their real data samples.

Moreover, we calculate the FID between the generated samples of BGAN, MDGAN, FLGAN, and F2U. Table II shows that BGAN outperforms the other architectures in terms of the FID value (normalized with respect to the maximum achieved FID), especially for CIFAR-10 data set which has more complex images with multiple color channels, the difference between the FID of the BGAN architecture and other architectures is higher. In particular, BGAN has 13%, 31%, and 38% lower FID compared to the best of the other distributed architectures for MNIST, fashion MNIST, and CIFAR-10 data sets, respectively.

One of the key advantages of BGAN is its low-bandwidth requirements, particularly for cases with very deep architectures. This advantage becomes especially significant when considering the bandwidth requirements for transmitting data between agents. As shown in Table III, the communication requirements of different distributed GAN architectures vary significantly.

In the case of BGAN, each agent receives $b$ samples from its neighbors at every training epoch, where $b$ is the batch size. This means that at every time step, $O(nb|x|)$ bandwidth

![Graph showing JSD comparison between BGAN, FLGAN, MDGAN, and F2U agents.](image-url)
is needed to transmit data between the agents, where $|\mathbf{x}|$ is the size of each data sample.

In contrast, the bandwidth required for MDGAN, FLGAN, and F2U are $O(n(|\mathbf{x}| + |\theta_d|))$, $O(n(|\theta_c| + |\theta_d|))$, and $O(n(|\mathbf{x}| + 1))$, respectively. Clearly, BGAN can significantly reduce the bandwidth overhead of distributed GANs, making it a more efficient choice for distributed learning.

Furthermore, BGANs do not require a central unit that aggregates information from multiple agents, which enhances their robustness in scenarios where an agent fails to communicate with its neighbors. This is a significant advantage over FLGAN, MDGAN, and F2U, which rely on a central unit. In these architectures, any failure in the central unit will disrupt the GAN output at all of the agents.

It is also important to note that our BGAN model does not require the network to be fully connected. Instead, as stated in Theorem 2, the network needs to be strongly connected. A graph is said to be strongly connected if every agent is reachable from every other agent, either directly or indirectly. This ensures the effective propagation of ideas or data samples across the network and allows for more flexibility in the network topology. This requirement of strong connectivity is common to other distributed GAN architectures, such as FLGAN, MDGAN, and F2U, and it is a crucial factor in managing the bandwidth requirements of the network.
V. CONCLUSION

In this article, we have proposed a novel BGAN architecture that enables agents to learn a data distribution in a distributed fashion in order to generate real-like data samples. We have formulated a game between BGAN agents and derived a unique NE at which agents can integrate data information from other agents without sharing their real data. Our architecture is fully distributed and does not require a central controller, significantly reducing communication overhead compared to other state-of-the-art distributed GAN architectures. Furthermore, our BGAN allows agents with different DNN designs to participate, enabling even computationally limited agents to contribute to brainstorming and gain information from others. Experimental results have shown superior performance in terms of lower JSD and FID across multiple data distributions. Looking forward, potential research directions include exploring different network architectures within the BGAN framework, applying our approach to other types of data, such as text or time-series, and investigating more complex multiagent games in the context of GANs. This work paves the way for future research in distributed learning and data generation, with the potential to inspire new methodologies and applications.

REFERENCES

[1] I. Goodfellow et al., “Generative adversarial nets,” in Proc. Adv. Neural Inf. Process. Syst. 27, 2014, pp. 2672–2680.
[2] C. Li and M. Wand, “Precomputed real-time texture synthesis with Markovian generative adversarial networks,” in Proc. Eur. Conf. Comput. Vision, 2016, pp. 702–716.
[3] A. Ferdowsi and W. Saad, “Generative adversarial networks for distributed intrusion detection in the Internet of Things,” in Proc. IEEE Global Commun. Conf., Waikoloa, HI, USA, Dec. 2019, pp. 1–6.
[4] S. Reed, Z. Akata, X. Yan, L. Logeswaran, B. Schiele, and H. Lee, “Generative adversarial text to image synthesis,” 2016, arXiv:1605.05396.
[5] S. Pascual, A. Bonafonte, and J. Serra, “Segan: Speech enhancement generative adversarial network,” 2017, arXiv:1703.09452.
[6] C. Vondrick, H. Pirsiavash, and A. Torralba, “Generating videos with scene dynamics,” in Proc. Adv. Neural Inf. Process. Syst., 2016, pp. 613–621.
[7] J. Dean and S. Ghemawat, “MapReduce: Simplified data processing on large clusters,” Commun. ACM, vol. 51, no. 1, pp. 107–113, Jan. 2008.
[8] Y. Low, D. Bickson, J. Gonzalez, C. Guestrin, A. Kyrola, and J. M. Hellerstein, “Distributed graphlab: A framework for machine learning and data mining in the cloud,” Proc. VLDB Endowment, vol. 5, no. 8, pp. 716–727, 2012.
[9] M. Assran, N. Loizou, N. Ballas, and M. Rabbat, “Stochastic gradient push for distributed deep learning,” in Proc. Mach. Learn. Res., Jan. 2019, pp. 344–353.
[10] W. Wen et al., “TernGrad: Ternary gradients to reduce communication in distributed deep learning,” in Proc. Adv. Neural Inf. Process. Syst. 30, Dec. 2017, pp. 1509–1519.
[11] J. Dean et al., “Large scale distributed deep networks,” in Proc. Adv. Neural Inf. Process. Syst., 2012, pp. 1223–1231.
[12] J. Konečný, H. B. McMahan, F. X. Yu, P. Richtárik, A. T. Suresh, and D. Bacon, “Federated learning: Strategies for improving communication efficiency,” 2016, arXiv:1610.05492.
[13] Q. Hoang, T. D. Nguyen, T. Le, and D. Phung, “Multi-generator generative adversarial nets,” 2017, arXiv:1708.02556.
[14] I. Durugkar, I. Gemp, and S. Mahadevan, “Generative multi-adversarial networks,” 2016, arXiv:1611.01673.
[15] A. Ghosh, V. Kulharia, V. Namboodiri, P. H. Torr, and P. K. Dokania, “Multi-agent diverse generative adversarial networks,” 2017, arXiv:1704.02906.
[16] C. Hardy, E. Le Merre, and B. Sericola, “MD-GAN: Multi-discriminator generative adversarial networks for distributed datasets,” in Proc. IEEE Int. Parallel Distrib. Process. Symp. (IPDPS), 2019, pp. 866–877.
[17] R. Yonetani, T. Takahashi, A. Hashimoto, and Y. Ushiku, “Decentralized learning of generative adversarial networks from multi-client non-IID data,” 2019, arXiv:1905.09684.
[18] I. Goodfellow, Y. Bengio, and A. Courville, Deep Learning. Cambridge, MA, USA: MIT Press, 2016.
[19] R. A. Horn and C. R. Johnson, *Matrix Analysis*. Cambridge, U.K.: Cambridge Univ. Press, 2012.

[20] Y. LeCun and C. Cortes. “MNIST handwritten digit database.” 2010. [Online]. Available: http://yann.lecun.com/exdb/mnist/

[21] H. Xiao, K. Rasul, and R. Vollgraf, “Fashion-MNIST: A novel image dataset for benchmarking machine learning algorithms,” 2017, arXiv:1708.07747.

[22] A. Krizhevsky, “Learning multiple layers of features from tiny images,” Dept. Comput. Sci., Univ. Toronto, Rep. 2009-TR, 2009.

[23] A. Radford, L. Metz, and S. Chintala, “Unsupervised representation learning with deep convolutional generative adversarial networks,” 2015, arXiv:1511.06434.

[24] K. He, X. Zhang, S. Ren, and J. Sun, “Deep residual learning for image recognition,” in Proc. IEEE Conf. Comput. Vision Pattern Recognition, 2016, pp. 770–778.

[25] I. Gulrajani, F. Ahmed, M. Arjovsky, V. Dumoulin, and A. C. Courville, “Improved training of Wasserstein GANs,” in Proc. Adv. Neural Inf. Process. Syst., 2017, pp. 5767–5777.

Aidin Ferdowsi is currently residing in McLean, VA, USA. He received the Ph.D. degree from Virginia Tech, Blacksburg, VA, USA.

He is a Distinguished Principal Machine Learning Engineer with Capital One, McLean, and a Celebrated Alumnus of Virginia Tech. He has held significant roles with Hughes Network Systems and Carvana, applying his expertise in Python, SQL, TensorFlow, Keras, and AWS to optimize autonomous connected vehicles, IoT networks, and satellite communications. With specialization in machine learning, deep learning, conversational AI, and cyber-physical security, he holds a U.S. patent on deep learning and has authored 23 published papers in esteemed venues like IEEE TRANSACTIONS ON COMMUNICATIONS.

Dr. Ferdowsi earned the 2021 Outstanding Ph.D. Dissertation Award in STEM at Virginia Tech. He is a Certified AWS Solutions Architect and a recipient of numerous awards, including the Bill and LaRue Blackwell Graduate Research Award and the 2017 Wireless@VT Fellowship, underscoring his commitment to innovation and excellence in his field.

Walid Saad (Fellow, IEEE) received the Ph.D. degree from the University of Oslo, Oslo, Norway, in 2010.

He is currently a Professor with the Department of Electrical and Computer Engineering, Virginia Tech, Blacksburg, VA, USA, where he leads the Network Science, Wireless, and Security Laboratory. He is also the Next-G Wireless Faculty Lead with Virginia Tech’s Innovation Campus. His research interests include wireless networks (5G/6G/beyond), machine learning, game theory, security, UAVs, semantic communications, cyber–physical systems, and network science.

Dr. Saad is also the recipient of the NSF CAREER Award in 2013, the AFOSR Summer Faculty Fellowship in 2014, and the Young Investigator Award from the Office of Naval Research in 2015. He is the recipient of the 2015 and 2022 Fred W. Ellersick Prize from the IEEE Communications Society, the IEEE Communications Society Marconi Prize Award in 2023, and the IEEE Communications Society Award for Advances in Communication in 2023. He received the Dean’s Award for Research Excellence from Virginia Tech in 2019. He was the (co)author of 11 conference best paper awards at IEEE WiOpt in 2009, ICIMP in 2010, IEEE WCNC in 2012, IEEE PIMRC in 2015, IEEE SmartGridComm in 2015, EuCNC in 2017, IEEE GLOBECOM (2018 and 2020), IFIP NTMS in 2019, and IEEE ICC (2020 and 2022). He was also the coauthor of the papers that received the IEEE Communications Society Young Author Best Paper Award in 2019, 2021, and 2023. Other recognitions include the 2017 IEEE ComSoc Best Young Professional in Academia Award, the 2018 IEEE ComSoc Radio Communications Committee Early Achievement Award, and the 2019 IEEE ComSoc Communication Theory Technical Committee Early Achievement Award. From 2015–2017, he was named the Stephen O. Lane Junior Faculty Fellow at Virginia Tech, and in 2017, he was named the College of Engineering Faculty Fellow. He has been annually listed in the Clarivate Web of Science Highly Cited Researcher List since 2019. He currently serves as an Area Editor for the IEEE TRANSACTIONS ON NETWORK SCIENCE AND ENGINEERING and the IEEE TRANSACTIONS ON COMMUNICATIONS. He is the Editor-in-Chief of the IEEE TRANSACTIONS ON MACHINE LEARNING IN COMMUNICATIONS AND NETWORKING. He was also an IEEE Distinguished Lecturer in 2019 and 2020.