Inductive learning and local differential privacy for privacy-preserving offloading in mobile edge intelligent systems

Jude Tchaye-Kondi¹, Yanlong Zhai¹ and Liehuang Zhu¹ *Member, IEEE*,

Abstract—We address privacy and latency issues in the edge/cloud computing environment while training a centralized AI model. In our particular case, the edge devices are the only data source for the model to train on the central server. Current privacy-preserving and reducing network latency solutions rely on a pre-trained feature extractor deployed on the devices to help extract only important features from the sensitive dataset. However, finding a pre-trained model or public dataset to build a feature extractor for certain tasks may turn out to be very challenging. With the large amount of data generated by edge devices, the edge environment does not really lack data, but its improper access may lead to privacy concerns. In this paper, we present DeepGuess, a new privacy-preserving, and latency aware deep-learning framework. DeepGuess uses a new learning mechanism enabled by the AutoEncoder(AE) architecture called Inductive Learning, which makes it possible to train a central neural network using the data produced by end-devices while preserving their privacy. With inductive learning, sensitive data remains on devices and is not explicitly involved in any backpropagation process. The AE's Encoder is deployed on devices to extract and transfers important features to the server. To enhance privacy, we propose a new local differentially private algorithm that allows the Edge devices to apply random noise to features extracted from their sensitive data before transferred to an untrusted server. The experimental evaluation of DeepGuess demonstrates its effectiveness and ability to converge on a series of experiments.

Index Terms—Edge intelligence, Local Differential privacy, Internet of Things, Cloud Computing, Deep Learning

I. INTRODUCTION AND MOTIVATIONS

Over the past decade, humanity has experienced accelerated development, which continues to intensify with technological advancements. With the breakthroughs in the Internet, mobile computing, the internet of things(IoT), and artificial intelligence(AI), more and more device processing power is getting increased while becoming smart and interconnected. IoT devices are conquering all sectors, be it healthcare, economy, agriculture, military, transport, etc. Most of the objects we use every day connect to the internet for data exchange. Phones with their many applications and sensors, smartwatches, and many of today's electronic devices produce data. This resulted in an explosion of data being generated at the network edge. To successfully enable AI power on edge devices and benefit from recent breakthroughs, the AI needs the data produced by edge devices to train intelligent models. However, training

deep neural networks on mobile devices is hardly feasible due to their cheap computation capability. Generally, IoT[1] devices have constrained resources, while deep neural network training requires considerable energy, memory, and processing capacity. Usually, edge devices upload their data to a cloud

Fig. 1: Framework Architecture

server and let cloud computing do the computations for them, nevertheless with an exponentially rising rate of generated data at the network edge, this strategy poses several challenges for the cloud server (latency, storage cost ...), and, is a source of privacy concern for the end-users. Besides the risks of private data being exposed to internal attacks from these cloud industries, there are also external threats if the industry is breached. Moreover, extra information on the private datasets could be obtained even if the data is anonymized[2][3] by masking sensitive values. Such private data may be stored as raw data features that might present a higher risk as the data is ready to be processed in any way possible. Cloud computing problems lead to a new paradigm that seeks to shift some of the computing from cloud to edge nodes or edge servers, i.e. edge computing and edge intelligence when AI is used in edge computing. Several studies take advantage of edge computing and the fact a Deep Neural Network(DNN) can be internally divided into two or more parts to solve these problems. The common concept is to train our AI model on a public dataset related to our task. The trained model is divided into two parts, such a way that the output of the first part has a reduced dimension compared to the input. The

J. Tchaye-Kondi, Y. Zhai and L. Zhu are with School of Computer Science, Beijing Institute of Technology, Beijing 100081, China.
E-mail: tchaye59@gmail.com, ylzhai@bit.edu.cn, liehuangz@bit.edu.cn
Manuscript received January 31, 2021; revised August 26, 2021.

first part is then sent to the edge devices, where it serves as a feature extractor on the user's sensitive data and only performs forward passes. Since the extracted features have a reduced dimension, the feature extractor also serves as a data compressor and helps tackle the network latency during data transfers. Further, a differentially private algorithm can be used on the edge device to ensure that the server can not recover sensitive data from the extracted features. On the server, the latent features are used to tune the second part of the model. This is the principle used by proposed frameworks such as ARDEN[4] and many others[5][4][6][7]. These frameworks focused mainly on image data types. Indeed, this type of resource is abundant on the internet. With a few clicks, it is possible to find different types of publicly available image datasets or even pre-trained models to build a feature extractor. But in top secrecy sectors, or when privacy comes into play, it gets a little more difficult. It is difficult to find some datasets in the military, medical, and many other fields. The most surprising example we can give is that of the IoT. Although the IoT field does not lack data, finding publicly available IoT-related data is not easy and often complicates research in this area. This can be explained by the fact that IoT objects such as smartphones, smartwatches, smart homes, etc, track several aspects of customer's daily life. Therefore, public exposure to this information will lead to many privacy concerns. This situation shows the requirement for a system that can preserve privacy, reduces network latency, with edge devices as its unique data source.

Deploying the full AI capacity on edge devices is not trivial due to their limited resources. To make matters worse, for copyright reasons, the service provider may not want to deploy its fine-tuned model on edge devices, or the model may be required by a more complex system deployed on a centralized server that produces the final result for end-users. For these reasons, it may not always be possible to train or deploy the AI model designed by a third-party cloud service provider on these edge-devices. It may therefore be necessary to centralize the training process on the cloud/edge server. But the client, the only data source for training the model in our case, does not want to release its sensitive data to the cloud service provider. To satisfy these conflicting needs, this paper proposes a new learning approach via a framework called DeepGuess illustrated in Fig.1. DeepGuess allows the analysis of the data produced at the network edge while reducing latency and preserving end-users' privacy by increasing the uncertainty of the features they send to the cloud via differential privacy. Our framework uses the AutoEncoder architecture(AE) to mitigate network latency and provide a first privacy layer. As shown in Fig.1, on the IoT devices, the Encoder is used as a features extractor and extracts from the sensitive data important features. To provide a second privacy layer, the differential privacy mechanism is used to introduce random noise into the extracted feature vector before we transfer it to the central server for further processing. The extracted features will be referred to as a latent vector in the rest of the paper. The contributions of our work can be summarized as follows:

- As most solutions in the literature, to build the feature extractor, we don't assume that we can find publicly available datasets or pre-trained models to help with our task. We believe this may constitute a bottleneck in some cases. Our system's unique data source is the edge devices. To access edge data in a privacy-preserving manner, we have introduced a new learning process that we call Inductive Learning. The proposed solution ensures that the raw private data is not centralized on the central server while providing the data utility.
- We reduce latency during data transmission from edge to cloud by only transferring latent vectors instead of the user's raw data. As most heavy computations migrate to the cloud, we then decrease the challenges of limited resources on edge devices.
- Additionally, we provide a new standardized approach for enabling ϵ -Local Differential Privacy in our system by allowing the devices to add noise to the latent vectors. And finally, we validate the performance of our framework with a series of experiments.

II. DATA ANONYMIZATION MECHANISMS

Anonymization of a dataset is a processing procedure that enables information to be deleted or modified in such a way as to make it anonymous. As a result, instances of the derived dataset can no longer be associated with a specific identity. In this section, we discuss current approaches applied to datasets for privacy:

A. Naive Data anonymization

Naive anonymization techniques are basic anonymization strategies that attempt to eliminate all sensitive attributes before the dataset is released. These methods include:

1) *Data masking*: hides information with altered values. Concretely, it consists of creating a mirror version of the dataset by applying tricks such as characters/words shuffling or replacement... Data masking is extremely efficient, but at the same time, it eliminates all the utility of this data and makes it difficult to perform several kinds of analysis on the masked version.

2) *Data generalization*: consists of removing or replacing certain information from data records to make them less identifiable. The goal is to eliminate identifiers that can uniquely identify a particular record while preserving essential patterns of the data. If we take the address example, you can remove part of its properties, such as the street number, the postcode, and any other details that might help anyone recognize the exact location or the entity related to the address. This process will make it more general. The most prevalent technique for data generalization is K-anonymity[8]. K-anonymity generalizes the dataset to a similar subgroup of K-instances. So if at least "k-1" records have the same properties in a dataset, we have achieved k-anonymity. For example, imagine a k-anonymized dataset for which "k" is 10, the property still the address. If we check each record in this set, we'll always find 9 other records that share the same address. It would therefore be hard to link the corresponding address to a single record.

3) *Data swapping*: It is also called permutation and consists of rearranging the dataset by swapping values with each other. For instance, it can be performed by swapping the values of the age column in the feature matrix with those of the gender column.

The biggest drawback of naive data anonymization methods is when one begins cross-referencing the anonymized dataset with a related one from a different source that may or may not be publicly accessible. It has been shown to be possible to identify records from the anonymized data using this cross-reference approach. The 2006 NetFix Prize[9] dataset is a typical example. In 2006, Netflix, the world's largest online DVD rental service, launched a competition to improve its video recommendation system. For the competition, Netflix released a dataset containing movie ratings of their subscribers between 1999 and 2005. Obviously, the dataset has been anonymized by removing all identifying customer details. Later, Narayanan et al. demonstrate that removing identifying details is not sufficient for confidentiality. They develop a de-anonymization algorithm based on a score system[10] that uses auxiliary information an adversary might have access to. Using this algorithm, they manage to identify some of the records. To illustrate the power of cross-referencing, it's important to note that the records retrieved are from users who have also rated movies under their name in the publicly available IMDb dataset[11]. The second concern with naive data anonymization methods is that they are often conducted on the server-side, so users have to rely on service providers' goodwill to anonymize their data.

B. Synthetic data

A synthetic dataset is a dataset generated from an existing real dataset using an algorithm. These strategies are becoming increasingly popular due to the advances in deep learning. Nowadays, a generative neural network[12](GAN) is capable of producing any data type: human faces, medical images, sound, etc. The dilemma with synthetic datasets is that there is a balance to be determined between privacy and data utility. Finding the right balance can be difficult because the more private the synthetic data, the less usable it is, and the more utility the synthetic data retains, the less privacy it offers.

C. Noise addition

Noise addition works by using a stochastic number to alter the value of sensitive attributes in the dataset. As shown by Table I, this alteration can be carried out via a simple mathematical operations[13], such as addition, multiplication, logarithmic...

Noise method	Operation
Additive Noise[14]	$y_i = x_i + r_i$
Multiplicative Noise[15]	$y_i = x_i r_i$
Logarithmic multiplicative noise[14]	$y_i = \ln(x_i) + r_i$

TABLE I: r_i is random variable, x_i represents our sensitive variable after applying the noise on x_i we get y_i .

Nevertheless, the latest state-of-the-art data perturbation approach is differential privacy and has been introduced by Dwork[2]. Differential privacy is the latest and most common method of data anonymization in computer science. Unlike previous approaches, it is the only one to have formal guarantees of data confidentiality (mathematical proofs)[16][17]. Formal guarantees are important and make it possible to quantify the re-identification risk of data records hence the enthusiasm for this method. Indeed, when analyzing an anonymized dataset via DP, a third party can never be sure whether or not the dataset has been altered. Businesses need data to improve their products, but as users, we want to preserve our privacy. Therefore, these contradicting needs can be met using differential privacy techniques and allow businesses to collect information about their users without compromising an individual's privacy. For service providers to train their ML models, devices at the network edge must release their data. However, they have to ensure that an adversary is not in a position to reverse engineer sensitive information from it. This is where differential privacy helps by offering strong privacy guarantees that facilitate the design of the PPDP algorithm used in our proposed framework. DP mechanisms rely on the incorporation of random noise in the data such that anything received by the adversary becomes noisy and imprecise, making it far more difficult to violate privacy.

D. Differential Privacy and Local Differential Privacy

In differential privacy, the amount of noise applied to the data is totally controlled by ϵ . $\epsilon = 0$ is the maximum noise and guarantees perfect privacy. $\epsilon = +\infty$ is the lowest noise and does not guarantee any privacy. ϵ is called the privacy budget. The concept of ϵ -Differential Privacy($\epsilon - DP$) was introduced in [2] and is formalized as follows:

Definition II.1 ($\epsilon - DP$). Let $\epsilon \geq 0$ and A be a randomized algorithm that takes as input a dataset, and A' the image of A . The algorithm A is said to provide $\epsilon - differential privacy$ if, for any adjacent datasets D_1 and D_2 that differ on a single element, and any subsets S of A' :

$$Pr[A(D_1) \in S] \leq \exp(\epsilon) \cdot Pr[A(D_2) \in S]$$

Where $Pr[A(D_1) \in S]$ indicates the probability that the outputs of algorithm A belong to S .

There are various ways for a randomized algorithm to achieve differential privacy: Laplace mechanism, Exponential mechanism, Posterior sampling. The problem with the $\epsilon - DP$ is that it remains centralized. Clients must trust the central server to keep their privacy and share their private information. Fortunately, a new DP variant called Local Differential Privacy (LDP) is proposed. LDP allows each client to add noise to the sensitive information locally. The concept was introduced in [18] and is formalized as follow:

Definition II.2 ($\epsilon - LDP$). Let $\epsilon \geq 0$ and A be a randomized algorithm that takes its input in X with X representing the user local data. A' is the image of A . The algorithm A is said to provide $\epsilon - local differential privacy$ if and only if, $\forall x_1, x_2 \in X$ and $\forall y \in A'$:

$$Pr[A(x_1) = y] \leq \exp(\varepsilon) \cdot Pr[A(x_2) = y]$$

To know how much noise or randomness we can introduce with ε , it is important to estimate our data sensitivity. In DP, the global data sensitivity given by Equation 1 is the maximum effect between two adjacent items or datasets on the output of an arbitrary f function, typically referred to as the query.

$$S_f = \max |f(d_1) - f(d_2)|_1 \quad (1)$$

E. Randomized Response: coin flips

LDP is a recent technique but the intuition behind is quite old and was inaugurated by Warner's *et al.* in [19]. It was introduced to collect statistical data from users' answers while ensuring confidentiality. In a conducted survey where a person has to answer YES or NO, the procedure is as follows: The user flips a coin in private if the head comes up, flips the coin again a second time, but ignores the results and answers truthfully. If the first flip wasn't a head, he flips the coin a second time and answers Yes if the head, No otherwise. The second flip of the first case is used to fool if a stranger is watching the flipping. Let suppose p the probability that the person will answer truthfully and $(1 - p)$ otherwise. This approach provides ϵ - differential privacy for $p = e^\epsilon / (1 + e^\epsilon)$ [20]. Google's LDP framework RAPPOR [21] uses this process to collect from chrome users' data (home pages, chrome configuration strings).

III. RELATED WORKS

In this section, we discuss earlier work on privacy-preserving frameworks for AI-based systems training in the edge computing environment. Three main approaches stand out from the literature. *Centralized approach*: this is the conventional cloud computing approach that consists of dedicating all processing to a central server. *Decentralized approach*: All processing is performed at the edge of the network. It can also take the form of a collaboration between end-devices. *Hybrid approach*: it is neither centralized nor decentralized, the processing is shared between the devices and the central server.

The centralization and processing of device data could lead to privacy issues by exposing some confidential information. To avoid these problems its android apps, Google has developed Federated Learning [22], a distributed learning system. Federated Learning is an AI framework where the objective is to train a high-quality centralized model while training data remains decentralized via a large number of clients devices [22]. In Federated Learning (FL), the edge device (smartphone) downloads the central model, learns how to improve it on its local data, and then summarizes the improvements like a small focused update. Only the update is sent to the cloud using encrypted communication, where it gets immediately averaged with other devices updates to improve the shared model. All the training data remains on edge devices, and Google claims that no updates are stored in the cloud. To measure the stakes, let's take the example

of GBoard, Android's predictive keyboard. Centralizing its data would enable Google to have direct access to all of the keystrokes performed by each user. This would be an infringement of user privacy and could also allow Google to collect by mistake user passwords, secret codes, credit card numbers, and much other confidential text typed on the device. Using Federated learning Google addresses this issue by only collecting model updates. Unfortunately, the FL is still young and faces important challenges that remain unresolved:

1) *Network Cost* [23]: In FL, the tradeoff between privacy and other factors such as communication expenses is not well balanced. Federated networks are potentially made up of large number of smart devices (phones, watches, cars, TVs, etc.) which may lead to a heavy network activity during each training round.

2) *Devices Diversity* [23]: Due to the hardware diversity of each device, federated networks could contain different types of devices with unbalanced resource capabilities. For this reason, only eligible devices may participate in the training. Moreover, in order to reduce communication and power fees, the selected devices must be plugged into a power source and connected to a wifi network.

3) *Concerns on privacy* [23]/[24]: The FL has not yet fully achieved its main privacy objective. Training the model at the data source, then sending model updates (gradient information) to the server, rather than raw user data, does not guarantee total privacy, as studies show that an interceptor could disclose sensitive information with these updates.

Fig. 2: AutoEncoder and Sparse AutoEncoder

Faced with the difficulties encountered by decentralized training strategies, the Hybrid option preserves the conventional centralized approach, except that this time the data is pre-processed and anonymized at the device level before being transferred to the central server. The strategy is adopted by the hybrid framework proposed by Osia et al [7]. Instead of running the entire process on the server, their system breaks down the DNN into a feature extraction module, which should be deployed on the client's device, and a classification module that operates in the cloud. The idea is to let the IoT

device run the initial layers of the neural network and then send the output to the cloud to feed the remaining layers and generate the final output. The service provider should pre-train the feature extractor on a public dataset before releasing it to the devices. For better privacy, various techniques are applied to the feature extractor's output. Firstly a Dimensionality Reduction technique through PCA[25] is used to reduce the feature dimensionality. Secondly, a technique referred to as Siamese Fine-tuning has helped to refine the feature extractor so that features of the same class fall within a small neighborhood of each other. Finally, noise addition is used to increase the inference uncertainty of unauthorized tasks. A similar process is adopted in most of proposed hybrid frameworks[4],[5][4][6][7]. Privacy-preserving in edge-computing is a hot topic and several challenges are still present. Our framework mainly addresses the issue of having to rely on a public dataset or pre-train model to build the features extractor that has never been addressed before.

IV. METHODOLOGY

In this section, we start by presenting the proposed framework and illustrate how the Inductive learning mechanism works. We also discuss the use of randomized unit response and the Laplace-DP mechanism to make our framework privacy-preserving.

Fig. 3: Different framework configurations: The Main model input may be the noisy latent vector or the reconstructed data.

A. FRAMEWORK OVERVIEW

Fig.1 provides the general structure for the DeepGuess framework. The framework is divided into two parts running respectively on the edge-device and the server. The edge device runs the Encoder and the Noise module while the Cloud server has the full AE and the Main model we aspire to train. During the training process, the AE and the Main model weights are randomly initialized and then fine-tuned with edge data.

EAs are special types of neural networks that learn to output their input. They are commonly used to learn a latent

representation of a dataset or as a dimensionality reduction technique. As shown in Fig.2a, the AE takes advantage of DNN splitting[26] property and splits the network architecture into two parts: Encoder and Decoder. The Encoder is used to convert the input into a reduced latent representation. Alongside, the Decoder tries to restore from the reduced encoding a representation as close as possible to its original input. There is also another AE variant called Sparse AE[27](SAE) which instead of reducing the input dimensionality will rather increase it(Fig.2b).

In our system, the AE acts as a device-to-server data bridge. As its name implies, the Main model is used for the primary task. It infers a value of interest used by a service API to generate a final result for the client. Deployed at the network edge, the Encoder makes it possible to reduce the data to be transmitted to a latent representation to which is added noise. Once this noisy latent representation has been transferred to the server, Fig.3 displays two possible configurations for training the Main model:

- The first is to use the latent vectors as input to train the main model. This configuration is the best because given that the Encoder already did the feature extraction. The Main model could therefore focus on predicting the output. The configuration also reduces the complexity of the Main model's neural network and leads to faster convergence.
- In the second configuration, the Decoder takes the noisy latent vectors and approximates the original data. This approximation is used for training the Main model.

Consequently, the encoder acts as a data compressor and the decoder as a data decompressor, helping to considerably reduce the volume of data during network transmission and therefore the cost of communication. The inferences and the whole training process to tune AE and Main model weights take place in the cloud. The client's device performs minimal processing: forward pass with Encoder and noise addition. We can also observe that the server never directly interacts with sensitive user data. As the Encoder reduces sensitive input to a lower dimension, there is a loss of information that pushes the EA to prioritize and learns important aspects of the input during training. This results in an imperfect reconstruction on the server during inferences. Additionally, adding noise to the latent vectors on the client-side will makes it difficult to restore sensitive information on the server-side. The scenario described in the introduction section, which consists of splitting the main model into two parts and using the first as a feature extractor, has some limitations in edge computing. This approach works with one model architecture at a time and requires a public dataset to pre-train the feature extractor. Neural network design consists of a lot of fine-tuning and often involves testing a number of architectures before selecting the correct one. Separating the feature extraction Module as in our solution provides more flexibility as several network models can be evaluated simultaneously using the same Encoder and Decoder. For the whole system to work properly, we need to train the AE before deploying the encoder to the devices. Related research efforts have adopted two solutions: the first

consists of using a public dataset to train the AE. In the second option, we collect the edge-data and centralize it on the server to train the AE. As mentioned earlier, we do not want to rely on publicly available data for our framework, and the second option is not desirable due to privacy concerns. Deprived of both options, we introduced a new learning mechanism called Inductive Learning discussed in the following section. Using Inductive Learning makes it possible for the server to tune the AE with the latent vectors it receives from the edge-devices.

B. Inductive Learning

Wireless charging or Inductive charging allows charging a device's battery without plugging it into a power source. Similarly, Inductive learning is a process that allows a neural network to learn a dataset structure without been fed with this one. Learning a dataset structure is the main purpose of AE. In the direct training mode illustrated in Fig.4, the dataset(X) representing the input and label is involved in the backpropagation process.

Fig. 4: Direct mode configuration

The indirect training configuration adopted by Inductive Learning is displayed in Fig.5. In this configuration, the dataset is not included in the backpropagation. It could be seen as a reverse engineering attack in which, by just extracting certain key features V from X , we try to learn the structure of the entire X . During training, the AE is sparse with the Decoder

Fig. 5: Inductive Learning configuration

as the first component, Encoder as the second component, and the latent vector V as input and label. The training is iterative, Fig.5 illustrate a single training round constituted of 2 steps:

- (1) The Encoder firstly extracts the latent vector V from X .
- (2) We use V to update the EA for one or more epochs.

So V is not constant at each training round, the Encoder generates a new V from X . The key to successfully train the AE is to always use the latest vector V output by the latest Encoder from X . This indirect training mode is used in our framework to tune the EA while maintaining the confidentiality of end devices, as it does not require centralizing sensitive data X on the server.

C. CLearn: Edge-cloud fine-tuning

Noisy latent vectors are the unique information the central server collects from the devices. These vectors are sufficient to tune the Encoder, Decoder, Main model's weights using the Inductive Learning mechanism presented in Section IV-B, and after for making inferences. Tuning Encoder's and Decoder's weights is like training a normal Sparse AutoEncoder(SAE). But this time, the latent vector dimensionality is increased by the Decoder before rebuilt by the Encoder. The SAE training is achieved through the back-propagation of computed loss, just like with a standard feed-forward neural network using mini-batch Stochastic Gradient Descent (SGD)[28] algorithm. The choice of the loss function may vary according to the dataset, but generally, the simple mean square error (MSE) adopted in our experiments usually works very well. The decoder will try to somehow produce a distribution as close as possible to the end-devices data. Accordingly, it is necessary to prevent the SAE from simply copying its input to output during training and perform poorly at inference(overfitting). Regularization and sparsity penalty are used for this purpose. A sparsity penalty is applied to hidden layers to stimulate their activations. The final SAE loss ($MSE + regularizer + sparsity\ penalty$) looks like Equation 2. In the equation, V represents the latent vector of size n and V' it output by EA: $V' = Encoder(Decoder(V))$.

$$L(V, V') = \left(\frac{1}{n}\right) \sum_{i=1}^n (V - V')^2 + R + \beta \sum_{j=1}^{s_2} KL(p || \hat{p}_j) \quad (2)$$

The regularization term R should result from different regularization techniques(l_0, l_1, l_2 [29],etc). For the sparsity penalty $\beta \sum_{j=1}^{s_2} KL(p || \hat{p}_j)$, we implemented the sparse activity regularization using Kullback-Leibler (KL) penalty according to Andrew Ng[27]. Where $KL(p || \hat{p}_j) = p \log \frac{p}{\hat{p}_j} + (1-p) \log \frac{1-p}{1-\hat{p}_j}$ is the KL divergence between a random Bernoulli variable with mean p and a Bernoulli random variable with mean $\hat{p}_j$ [27]. p controls the sparsity level of each layer and β is just a weight factor. Algorithm 1 summarized the entire training steps. It illustrates the way a differentially private output is generated from the client-device and how tuning is conducted on the central-server.

The training process consisting of several rounds is completely transparent to the end-users. Whenever a client accesses the service, he only makes inference by transferring noisy latent encoding to the server and receiving the result regardless of whether the model is trained or not. In order to continuously improve the service quality and client experience, the central server needs data produced by the end-devices to train the models. To exchange data in a privacy-conserving manner, the end-devices pre-process their sensitive data and only send a noisy latent coding and the corresponding labels to the server. The central server then collects latent coding from multiple devices over a given period of time, uses it to tune models, and deletes it after tuning. Both SAE and Main model weights are randomly initialized, and the training round presented in Fig.3 and Algorithm 1 is as follows:

1) *Encoder deployment*: The server deploys the latest Encoder to the end-devices.

2) *Features Extraction*: Each device uses the Encoder, perform a forward pass with its local sensitive data, and extract important features within a latent vector. The latent vector is further passed to the Noise module that incorporates random noise. After that, the noisy latent vector is transmitted to the central server.

3) *SAE Tuning*: On the server, the SAE is firstly tuned for one or more epochs, while the Main model remains constant. The Decoder takes the noisy latent features and tries to approximate the device's sensitive data. With the generated approximation, the Encoder attempts the inverse operation. The error measured between Decoder input and Encoder output (Equation 2) is used in a backpropagation process by the optimizer to adjust their weights.

4) *Main model Tuning*: Similarly, we keep the SAE constant when training the Main model. The Main model is also tuned for one or more epochs. As shown by Fig.3, the Main model can be designed to take the latent vectors or the Decoder's output as its input. The error between the Main model prediction and the expected result is then used to update its weights. If the SAE is not locked, gradients will flow through the Decoder and make the entire system unstable.

After several rounds, as the SAE improves with training, the main model performance will also improve until the whole system converges. When the system totally converges, the final Encoder is deployed on the edge-devices where it is only used for inferences until the next training session.

D. ENHANCING PRIVACY WITH NOISE

Extracting and transferring important features to the server doesn't guarantee total privacy since it has been proven[30][31] to be possible to leak sensitive information with these features. The trained model is capable of memorizing some sensitive information, as shown by Fig.8, and may later be used by a malicious adversary to leak the memorized knowledge, thus violating privacy. In addition, some sensitive information of the input can remain clearly perceptible in the latent feature vector. To reinforce privacy by making it difficult to disclose sensitive features, we add random noise to the latent vector. We apply two levels of randomization on the latent features before it is transferred to the central server. The first level that we called Randomized Units Response(RUR) is inspired by the Randomized Response mechanism presented in Section II-E. We used the Laplace differential privacy additive noise mechanism[32] for the second level of randomization. Client data may be images, text, audio, etc. Adding noise directly to the data would involve designing a specific DP mechanism for each data type. It is important to note that DP is not a catch-all solution for any task and dataset but a methodology for enforcing privacy in a framework. For example, only categorical data and strings are supported by Google's RAPPOR[21] DP framework. Nevertheless, our solution does not make any assumptions about the dataset and apply noise on the latent vector rather than the input data. During the client's side features extraction, from the lowest

Algorithm 1: Training Algorithm

Data:
 $S_G \leftarrow$ Encoder's global sensitivity;
 $\epsilon \leftarrow$ Privacy budget;
 $p \leftarrow$ RUR probability;
1 for *round* in range $[0, 1, \dots, n_rounds]$ **do**
 /* End node */
 Download the *Encoder* from Server;
 Get (x, y) user's sensitive data and the target;
 $v \leftarrow \text{Encoder}(x)$; /* Extract features
 for $v_i \in v$ **do**
 /* Add noise according to Equation 5
 $v_i \leftarrow \text{RUR}(v_i, p) + \mathcal{R}, \mathcal{R} \in \text{Lap}(\frac{S_G}{\epsilon})$
 end
 Upload v and target y to the Server

 /* Edge-Server */
 /* collect latent vectors from multiple
 nodes
 for *node* in $[node1, node2, \dots]$ **do**
 Download (v_i, y_i) from *node*;
 Add (v_i, y_i) to *buffer*;
 end
 /* Tuning
 $(V, Y) \leftarrow$ Get all latent vectors and targets from
 buffer;
 /* Tune the EA
 for *epochs* in $[1, 2, \dots, n_epochs]$ **do**
 $X' \leftarrow \text{Decoder}(V)$; /* Noisy estimation of
 data distribution
 $V' \leftarrow \text{Encoder}(X')$;
 Compute the error between V, V' (Equation 2);
 Optimize *Encoder* and *Decoder* ;
 end
 /* Tune the Model
 $X' \leftarrow \text{Decoder}(V)$;
 for *epochs* in $[1, 2, \dots, n_epochs]$ **do**
 $Y' \leftarrow \text{Model}(X')$;
 Compute the error between Y, Y' ;
 Optimize *Model*;
 end
 Clear the *buffer*;
27 end

to highest Encoder's layers, the input dimensionality will be gradually reduced until the units on the last layer produce the final output, which can be a 1D latent vector. Here the Encoder is our query function f and maps the user's input to a latent vector that contains activations of the neural units on its last layer. In a neural network, the output of each layer $h_{W,b}(X) = \phi(XW + b)$ is used as input for the next layer. X is what the layer receives in input, W is a weights matrix, b is a bias vector, ϕ is called the activation function. As we can observe $XW + b$ is just a linear operation, the role of ϕ is to incorporate nonlinearity and help the network learn much more complex structure from the data. For reasons

we'll explain later, all Encoder's layers could use any available activation function except the last layer, which activation must be bounded within a certain range. Then we have $\alpha \leq f_u \leq \beta$ with α and β the minimum and maximum activation for a unit u on the last. Possible bounded activation function[33] may include the Hyperbolic Tangent ($-1 < f(x) < 1$), the Binary Step Function ($f(x) \in \{0, 1\}$), Sigmoid function ($0 < f(x) < 1$), etc. The bounded variant of the rectified linear unit (ReLU) proposed in [34] can also be used.

1) *Randomized Units Response (RUR)*: Enabling DP using Randomized Response only works when the response is a binary attribute (YES or NO)[32] or a categorical value[21] however, in our situation, the latent vector contains real-value numbers. We could binarize the latent vector as suggested by Pathum et al.[35] with an algorithm called LATENT. But the problem with LATENT is that its algorithm is quite complex and would require much computation from the client device.

$$RUR(f_u, p) = \begin{cases} \mathcal{R}, & \mathcal{R} \in [\alpha, \beta]; \\ f_u & ; \end{cases} \quad \begin{matrix} \text{prob} : p \\ \text{prob} : (1 - p) \end{matrix} \quad (3)$$

With RUR, we've taken a simpler approach. Equation 3 illustrates the RUR mechanism performed by randomizing each unit activation value present in the latent vector. The value is replaced by the random number $\mathcal{R}$ drawn from the range $[\alpha, \beta]$ with a probability of p and is preserved with a probability of $1 - p$. This is the first reason why we need a bounded activation for the last layer. It ensures that an adversary could not easily find patterns to recognize True activations values for random values within the latent vector. For $p = 0$ (utility preserved and low privacy), all True activations are preserved. For $p = 1$ (No utility and perfect privacy), all activations are replaced by random values. Since p is defined by the client, RUR provides him with a kind of refutability to the data he transfers to the cloud. As we will see in the experiments section, RUR also acts as a regularizer by enforcing the network to learn the overall data distribution instead of a particular user distribution. It also helps fight overfitting during training.

2) *Additive noise*: Some True activations can pass through the RUR. To further enforce uncertainty, noise is added to each variable using the Laplace-DP mechanism. We will start by defining the data sensitivity according to a single unit response base on the global sensitivity given in Equation 1. Since the last layer's activation is bounded by α and β , the sensitivity of f for each unit can easily be defined as :

$$S_{f_u} = \beta - \alpha \quad (4)$$

If we consider that the last layer has k neural units, the overall sensitivity will be $S_f = k(\beta - \alpha)$.

$$f'_u = RUR(f_u) + \mathcal{R}, \mathcal{R} \in \text{Lap}\left(\frac{S_f}{\epsilon}\right) \quad (5)$$

At the end, each value f_u in the latent vector is replaced by f'_u define in Equation 5. This time the random number $\mathcal{R}$ is drawn from the Laplace distribution $\text{Lap}(\frac{S_f}{\epsilon})$ and the final result f' is sent to the central server for further processing. The client could control his total privacy budget with p and ϵ . For strong privacy, he could increase the random response

likelihood by setting a high p value and/or use a high noise by using a smaller ϵ value. As a side effect, setting high privacy budgets could deteriorate the quality of the service he receives. Then he has to find a good balance between privacy and the service quality he needs. Combining RUR with the Laplace-DP mechanism helps us enable ϵ -LDP using the randomized response mechanism on real-value responses instead of binary responses. Our approach is much more resource-efficient and quicker to compute than LATENT[35]. This is crucial if we want to reduce the amount of processing and latency at the client-side.

V. FRAMEWORK EVALUATION

We evaluated our privacy-preserving system on the popular MNIST[36] datasets used as benchmarks for testing differential privacy in deep-learning. The MNIST[36] is a handwritten digits image dataset. Each image contains a single digit in the grayscale format of size 28x28. There are in total 10 digits from 0 to 9 (10 classes). The dataset is provided with 60,000 examples to be used for training and 10,000 examples for testing. MNIST is considered to be the hello world dataset to go with new learning techniques and pattern recognition methods on image datasets while spending minimal efforts on preprocessing. We have implemented and evaluated the Inductive learning mechanism using the Python Keras API[37], which is a high-level API of Google's TensorFlow engine[38]. All the experiments are conducted on the Colab[39] platform with a configuration of 13G of RAM, a single CPU core of 2.20GHz, and a Tesla T4 GPU.

A. Experimental Setup

The MNIST learning objective is to predict the digit given its image. First, we built a Baseline Convolutional Neural Network (CNN) model to perform this task by directly taking the images as input. The architecture of that baseline CNN model is presented Fig.6a and does not include any differential privacy mechanism. To simplify the experiment from all the possible configurations in our framework, we consider the configuration of Fig.3 with Main model 2 as our task model. The main model is then fed the latent vector as input instead of a reconstruction (X'). Moreover, the AE and Main model are tuned for a single epoch in each training round. The structure of our system's Encoder, Decoder, and Main Model is also given in Fig.[6b,6c,6d]. All convolutional layers use the default $\text{stride} = 1$, $\text{pad} = 1$. The MaxPoolings size is $2 * 2$ in both Baseline and Encoder. The Baseline retakes Encoder's structure, flattens the last output, and adds some Dense layers to make predictions. So in the Baseline model, apart from its last Dense output layer with the *softmax* activation, other layers' activations are *relu*. In the Encoder, Decoder, and Main model, layers activations are also *relu* except the Encoder and Main model outputs layers. The Main model output layer uses *softmax* to output probabilities for predicting a digit. The Encoder's last layer activation is *tanh*. The *tanh* is chosen because the Encoder represents the query to the user's sensitive data. We then need its output to be bounded in order to correctly estimate the global data sensitivity when

Fig. 6: The baseline model and architecture of our framework components

applying differential privacy. The whole experiment will use a mini-batch size of 64 and a learning rate fixed to 0.01 with 0.1 exponentially decaying factor. Before the training, the dataset is normalized by dividing the sample by the maximum pixel value(255). To prove the efficiency of the inductive learning mechanism, we first of all, compared the Baseline model accuracy with our system without considering the noise module. Secondly, we add the noise module to our framework and analyze the impact on the client's privacy and model performance with different privacy budgets.

B. Results without differential privacy

To validate the mechanism behind the Inductive learning training compared with the conventional stochastic gradient descent, we remove the noise module from our framework and test it against the Baseline model. Fig.7a and Fig.7b plot accuracies of both approaches on the training and testing sets. The results confirm the intuition behind Inductive Learning and prove that it is possible to learn from a dataset without involving it in any back-propagation process. The baseline achieves a maximum accuracy is 0.9947 on the test data. With a maximum accuracy of 0.9932, our system achieves almost the same performance. The only remarkable difference is that the Baseline model converges much faster than our approach. This slow convergence part is observed during the firsts 15 training rounds and was expected. In our system, the main model convergence is highly dependent on AE performance. In the first place, the Encoder should learn how to effectively

extract important features from the dataset before the main model can learn anything from those features. For this reason, we can observe from Fig.7a and Fig.7b that as the AE loss decreases, the main model accuracy increases.

Fig.8 shows the Decoder output from some digits. Although the system is not trained on the sensitive dataset, the Decoder is able to estimate with great precision the dataset distribution. Because of that, Encoder was able to properly learn to extract features from the Decoder's guess X' of the sensitive data X and after perform well on X during features extraction on the client-side. We then train a centralized AI model without moving the end-devices raw data to the cloud. Even if the reconstructed digits seem to be a little bit bigger and blurry, we can clearly discern them. Therefore Inductive Learning without the noise module does not bring strong confidentiality to end-users. Particularly with image inputs. It does, however, assure that their raw private data is not stored in a central server while at the same time providing data utilities. At the client-side, the image of size 28×28 is transformed to a latent vector of size $4 \times 4 \times 16$, thereby reducing by 3 the size of the data that the client transfers to the central server. The weakness of our framework is observed during the training process. At each round, the client downloads the Encoder, extract and transfer a latent vector to the server. Consequently, in our experiment that trained the system for 60 rounds, this process is repeated 60 times. Indeed the training looks a bit expensive but, this cost is largely compensated during inferences. At inference time, the client holds the latest Encoder and only sends the latent vector to the server.

Framework	Privacy Budget	Max accuracy	
		train	test
Baseline Model	N/A	0.9999	0.9947
DeepGuess	N/A	0.9999	0.9932
DeepGuess	$(p = 0.1, \epsilon = 15)$	0.9900	0.9860
DeepGuess	$(p = 0.1, \epsilon = 10)$	0.9100	0.8500
DeepGuess	$(p = 0.1, \epsilon = 5)$	0.7800	0.5900
DeepGuess	$(p = 0.2, \epsilon = 15)$	1.0000	1.0000
DeepGuess	$(p = 0.3, \epsilon = 15)$	0.9800	0.9710
DeepGuess	$(p = 0.5, \epsilon = 15)$	0.7300	0.5100

TABLE II: Maximum accuracy for different privacy budgets

C. Results with differential privacy

The DeepGuess framework can be tuned in a privacy-preserving manner. The noise module allows the client to add $\epsilon - LDP$ noise in order to degrade the latent vector. This will prevent the cloud server from fully disclosing sensitive information. In addition, if intercepted, the latent vector will look noisy to the attacker. Recall that the noise is completely controlled by two parameters: p and ϵ . A high value of p will include more randomization within the latent vector by substituting certain units response by random values. While a small value of ϵ will increase the amount of Laplace noise added to them. So High noise means a high value of p or a small value of ϵ . We plot the Decoder's output activations when different ϵ and p values are used by the client in Fig.8. The output is normalized back within the range $[0 - 255]$. We observe that there is no privacy without noise since the

Fig. 7: Training and testing performance of the Baseline model compared to DeepGuess framework without differential privacy and DeepGuess performances by considering different privacy budgets.

Fig. 8: Decoder's output activations with and without DP noise

user's data is clearly recoverable. With $(p = 0.1, \epsilon = 0.10)$, the recovered digits are somehow damaged. However, some utilities are well preserved. Whereas when we set a very high noise $(p = 0.1, \epsilon = 0.5)$, we obtain better privacy, but the image loses all its utilities.

Our framework training results using different (ϵ, p) are shown in Fig.[7c-7h]. The noise effect is easily perceptible because it slows down learning speed. First we fixed $p = 0.1$ and tried different ϵ . When ϵ is within $[10 - 15]$, we can still get high accuracy with relatively strong privacy by looking at Fig.8. For $\epsilon \leq 5$, the noise is high, and more training rounds are needed for the networks to minimize their losses. Second, we fixed $\epsilon = 15$ and tried different p . For $p \leq 3$, the convergence is faster and much more stable than the previous experiment, demonstrating that the RUR is typically acting as a regularizer. With $p = 0.2$, our framework was able to achieve the maximum accuracy of 1 on both training and testing datasets, thereby surpassing the Baseline model. But after a certain peak of randomization frequency, $p \geq 5$, the training becomes chaotic. Table II displays maximum accuracies on used privacy budgets. Very low noise use to act as a regularizer during training and doesn't provide strong privacy. Very high noise provides perfect privacy but tends to destroy all the data usefulness. The end-client should then find a good balance between the privacy and the utilities he is ready to disclose to the service provider the reason why we introduce the notion of Dynamic privacy budget.

D. Preventing leakage with Dynamic privacy budget

Different instances in the user's sensitive data could require different quantities of noise. In existing solutions proposed in the literature, the end-user blindly sets a constant privacy budget. Or the server determines a fixed budget for all clients. With the Dynamic privacy budget, the Decoder is also sent to the client's device. It helps him to evaluate the reconstruction risk by attempting various privacy budgets. The client could evaluate the decoder's output and will be aware of how potential reconstructions from the latent vector will look like on the cloud. He then set the noise according to the utilities he wants to reveal or hide. Without this option, most clients will tend to set a very high noise, which is not always appropriate.

VI. CONCLUSION

We introduced inductive learning, a new mechanism for training a feature extractor and a centralized neural network, while the sensitive data remain at the edge of the network. We also suggested a new local differentially private algorithm by combining Randomized Response and Laplace-DP mechanisms for better privacy. Both combined allow training in a privacy-preserving manner a centralized deep neural network with the end-devices as the only data source. Previous work typically uses public datasets or publicly available models to construct a feature extractor. Our work demonstrates that in the case no public dataset or no pre-trained model is found for the task, it is possible to build a very efficient feature extractor in the edge-computing environment without violating the edge devices' privacy. Our solution offers a guarantee that no confidential data is centralized on the cloud, takes into account data transmission costs, and the fact that edge-devices have constrained resources by moving heavy processing to the server. Our proposal opens several future directions for research. We intend to explore ways to minimize the training overheads and further applications of the Inductive learning mechanism.

ACKNOWLEDGMENT

The authors thank the anonymous reviewers for their insightful suggestions.

REFERENCES

- [1] A. Zanella, N. Bui, A. Castellani, L. Vangelista, and M. Zorzi, "Internet of things for smart cities," *IEEE Internet of Things journal*, vol. 1, no. 1, pp. 22–32, 2014.
- [2] C. Dwork, "Differential privacy: A survey of results," in *International conference on theory and applications of models of computation*. Springer, 2008, pp. 1–19.
- [3] A. Narayanan and V. Shmatikov, "Robust de-anonymization of large sparse datasets: a decade later," *May*, vol. 21, p. 2019, 2019.
- [4] J. Wang, J. Zhang, W. Bao, X. Zhu, B. Cao, and P. S. Yu, "Not just privacy: Improving performance of private deep learning in mobile cloud," in *Proceedings of the 24th ACM SIGKDD International Conference on Knowledge Discovery & Data Mining*, 2018, pp. 2407–2416.
- [5] Y. Mao, S. Yi, Q. Li, J. Feng, F. Xu, and S. Zhong, "A privacy-preserving deep learning approach for face recognition with edge computing," in *Proc. USENIX Workshop Hot Topics Edge Comput.(HotEdge)*, 2018, pp. 1–6.
- [6] A. M. Ghosh and K. Grolinger, "Deep learning: Edge-cloud data analytics for iot," in *2019 IEEE Canadian Conference of Electrical and Computer Engineering (CCECE)*. IEEE, 2019, pp. 1–7.
- [7] S. A. Osia, A. S. Shamsabadi, S. Sajadmanesh, A. Taheri, K. Katevas, H. R. Rabiee, N. D. Lane, and H. Haddadi, "A hybrid deep learning architecture for privacy-preserving mobile analytics," *IEEE Internet of Things Journal*, vol. 7, no. 5, pp. 4505–4518, 2020.
- [8] K. LeFevre, D. J. DeWitt, and R. Ramakrishnan, "Mondrian multi-dimensional k-anonymity," in *22nd International conference on data engineering (ICDE'06)*. IEEE, 2006, pp. 25–25.
- [9] K. Hafner, "if you liked the movie, a netflix contest may reward you handsomely," *New York Times*, vol. 2, 2006.
- [10] A. Narayanan and V. Shmatikov, "Robust de-anonymization of large sparse datasets," in *2008 IEEE Symposium on Security and Privacy (sp 2008)*. IEEE, 2008, pp. 111–125.
- [11] A. L. Maas, R. E. Daly, P. T. Pham, D. Huang, A. Y. Ng, and C. Potts, "Learning word vectors for sentiment analysis," in *Proceedings of the 49th Annual Meeting of the Association for Computational Linguistics: Human Language Technologies*. Portland, Oregon, USA: Association for Computational Linguistics, June 2011, pp. 142–150. [Online]. Available: <http://www.aclweb.org/anthology/P11-1015>
- [12] I. Goodfellow, J. Pouget-Abadie, M. Mirza, B. Xu, D. Warde-Farley, S. Ozair, A. Courville, and Y. Bengio, "Generative adversarial nets," in *Advances in neural information processing systems*, 2014, pp. 2672–2680.
- [13] K. Mivule, "Utilizing noise addition for data privacy, an overview," *arXiv preprint arXiv:1309.3958*, 2013.
- [14] J. Kim, "A method for limiting disclosure in microdata based on random noise and transformation," 1986.
- [15] J. Kim and W. Winkler, "Multiplicative noise for masking continuous data," *Statistics*, vol. 1, p. 9, 2003.
- [16] C. Dwork, A. Roth et al., "The algorithmic foundations of differential privacy," *Foundations and Trends in Theoretical Computer Science*, vol. 9, no. 3–4, pp. 211–407, 2014.
- [17] K. Nissim, T. Steinke, A. Wood, M. Altman, A. Bembenek, M. Bun, M. Gaboardi, D. R. O'Brien, and S. Vadhan, "Differential privacy: A primer for a non-technical audience," in *Privacy Law Scholars Conf*, vol. 3, 2017.
- [18] S. P. Kasiviswanathan, H. K. Lee, K. Nissim, S. Raskhodnikova, and A. Smith, "What can we learn privately?" *SIAM Journal on Computing*, vol. 40, no. 3, pp. 793–826, 2011.
- [19] S. L. Warner, "Randomized response: A survey technique for eliminating evasive answer bias," *Journal of the American Statistical Association*, vol. 60, no. 309, pp. 63–69, 1965.
- [20] P. Kairouz, S. Oh, and P. Viswanath, "Extremal mechanisms for local differential privacy," in *Advances in neural information processing systems*, 2014, pp. 2879–2887.
- [21] Ú. Erlingsson, V. Pihur, and A. Korolova, "Rappor: Randomized aggregatable privacy-preserving ordinal response," in *Proceedings of the 2014 ACM SIGSAC conference on computer and communications security*, 2014, pp. 1054–1067.
- [22] J. Konečný, H. B. McMahan, F. X. Yu, P. Richtárik, A. T. Suresh, and D. Bacon, "Federated learning: Strategies for improving communication efficiency," *arXiv preprint arXiv:1610.05492*, 2016.
- [23] T. Li, A. K. Sahu, A. Talwalkar, and V. Smith, "Federated learning: Challenges, methods, and future directions," *IEEE Signal Processing Magazine*, vol. 37, no. 3, pp. 50–60, 2020.
- [24] P. Kairouz, H. B. McMahan, B. Avent, A. Bellet, M. Bennis, A. N. Bhagoji, K. A. Bonawitz, Z. Charles, G. Cormode, R. Cummings, R. G. D'Oliveira, S. E. Rouayheb, D. Evans, J. Gardner, Z. Garrett, A. Gascón, B. Ghazi, P. B. Gibbons, M. Gruteser, Z. Harchaoui, C. He, L. He, Z. Huo, B. Hutchinson, J. Hsu, M. Jaggi, T. Javidi, G. Joshi, M. Khodak, J. Konečný, A. Korolova, F. Koushanfar, S. Koyejo, T. Lepoint, Y. Liu, P. Mittal, M. Mohri, R. Nock, A. Özgür, R. Pagh, M. Raykova, H. Qi, D. Ramage, R. Raskar, D. Song, W. Song, S. U. Stich, Z. Sun, A. T. Suresh, F. Tramèr, P. Vepakomma, J. Wang, L. Xiong, Z. Xu, Q. Yang, F. X. Yu, H. Yu, and S. Zhao, "Advances and open problems in federated learning," 2019. [Online]. Available: <https://arxiv.org/abs/1912.04977>
- [25] S. Wold, K. Esbensen, and P. Geladi, "Principal component analysis," *Chemometrics and intelligent laboratory systems*, vol. 2, no. 1–3, pp. 37–52, 1987.
- [26] Z. Zhou, X. Chen, E. Li, L. Zeng, K. Luo, and J. Zhang, "Edge intelligence: Paving the last mile of artificial intelligence with edge computing," *Proceedings of the IEEE*, vol. 107, no. 8, pp. 1738–1762, 2019.
- [27] A. Ng et al., "Sparse autoencoder," *CS294A Lecture notes*, vol. 72, no. 2011, pp. 1–19, 2011.
- [28] L. Bottou, "Stochastic gradient descent tricks," in *Neural networks: Tricks of the trade*. Springer, 2012, pp. 421–436.
- [29] R. Tibshirani and L. Wasserman, "A closer look at sparse regression," 2016.
- [30] C. Song, T. Ristenpart, and V. Shmatikov, "Machine learning models that remember too much," in *Proceedings of the 2017 ACM SIGSAC Conference on Computer and Communications Security*, 2017, pp. 587–601.
- [31] M. Fredrikson, S. Jha, and T. Ristenpart, "Model inversion attacks that exploit confidence information and basic countermeasures," in *Proceedings of the 22nd ACM SIGSAC Conference on Computer and Communications Security*, 2015, pp. 1322–1333.
- [32] Y. Wang, X. Wu, and D. Hu, "Using randomized response for differential privacy preserving data collection," in *EDBT/ICDT Workshops*, vol. 1558, 2016.
- [33] C. Nwankpa, W. Ijomah, A. Gachagan, and S. Marshall, "Activation functions: Comparison of trends in practice and research for deep learning," *arXiv preprint arXiv:1811.03378*, 2018.
- [34] S. S. Liew, M. Khalil-Hani, and R. Bakhteri, "Bounded activation functions for enhanced training stability of deep neural networks on visual pattern recognition problems," *Neurocomputing*, vol. 216, pp. 718–734, 2016.
- [35] P. C. M. Arachchige, P. Bertok, I. Khalil, D. Liu, S. Camtepe, and M. Atiquzzaman, "Local differential privacy for deep learning," *IEEE Internet of Things Journal*, 2019.
- [36] Y. LeCun, C. Cortes, and C. Burges, "Mnist handwritten digit database," *ATT Labs [Online]*. Available: <http://yann.lecun.com/exdb/mnist>, vol. 2, 2010.
- [37] F. Chollet et al., "Keras," <https://github.com/fchollet/keras>, 2015.
- [38] M. Abadi, A. Agarwal, P. Barham, E. Brevdo, Z. Chen, C. Citro, G. S. Corrado, A. Davis, J. Dean, M. Devin, S. Ghemawat, I. Goodfellow, A. Harp, G. Irving, M. Isard, Y. Jia, R. Jozefowicz, L. Kaiser, M. Kudlur, J. Levenberg, D. Mané, R. Monga, S. Moore, D. Murray, C. Olah, M. Schuster, J. Shlens, B. Steiner, I. Sutskever, K. Talwar, P. Tucker, V. Vanhoucke, V. Vasudevan, F. Viégas, O. Vinyals, P. Warden, M. Wattenberg, M. Wicke, Y. Yu, and X. Zheng, "TensorFlow: Large-scale machine learning on heterogeneous systems," 2015, software available from tensorflow.org. [Online]. Available: <https://www.tensorflow.org/>
- [39] E. Bisong, "Google colabouratory," in *Building Machine Learning and Deep Learning Models on Google Cloud Platform*. Springer, 2019, pp. 59–64.

Jude Tchaye-Kondi Received the BS degree from the Department of Computer Science, Catholic University of West Africa. He joins in 2017 Beijing Institute of Technology, China as a graduate student and is now pursuing a Ph.D. program. His research interest includes parallel and distributed computing, edge computing, machine learning.

Currently, his research projects focus on applying artificial intelligence algorithms in the edge computing environment.

Yanlong Zhai received the B.Eng. degree and Ph.D. degree in computer science from Beijing Institute of Technology, Beijing, China, in 2004 and 2010. He is an Assistant Professor in the School of Computer Science, Beijing Institute of Technology. He was a Visiting Scholar in the Department of Electrical Engineering and Computer Science, University of California, Irvine. His research interests include cloud computing and big data.

Liehuang Zhu received the B.Eng. and Master Degrees in computer application from Wuhan University, Wuhan, Hubei, China, in 1998 and 2001 respectively. He received the Ph.D. degree in computer application from Beijing Institute of Technology, Beijing, China, in 2004. He is currently a Professor in the Department of Computer Science, Beijing Institute of Technology, Beijing, China. He is selected into the Program for New Century Excellent Talents in University from Ministry of Education, China. His research interests include internet of things, cloud

computing security, internet, and mobile security