Local Differential Privacy for Deep Learning

Pathum Chamikara Mahawaga Arachchige, Peter Bertok, Ibrahim Khalil, Dongxi Liu, Seyit Camtepe, and Mohammed Atiquzzaman, *Senior Member, IEEE*

***Abstract*—The Internet of Things (IoT) is transforming major industries, including but not limited to healthcare, agriculture, finance, energy, and transportation. IoT platforms are continually improving with innovations, such as the amalgamation of software-defined networks (SDNs) and network function virtualization (NFV) in the edge-cloud interplay. Deep learning (DL) is becoming popular due to its remarkable accuracy when trained with a massive amount of data such as generated by IoT. However, DL algorithms tend to leak privacy when trained on highly sensitive crowd-sourced data such as medical data. The existing privacy-preserving DL algorithms rely on the traditional server-centric approaches requiring high processing powers. We propose a new local differentially private (LDP) algorithm named LATENT that redesigns the training process. LATENT enables a data owner to add a randomization layer before data leave the data owners' devices and reach a potentially untrusted machine learning service. This feature is achieved by splitting the architecture of a convolutional neural network (CNN) into three layers: 1) convolutional module (CNM); 2) randomization module; and 3) fully connected module. Hence, the randomization module can operate as an NFV privacy preservation service in an SDN-controlled NFV, making LATENT more practical for IoT-driven cloud-based environments compared to existing approaches. The randomization module employs a newly proposed LDP protocol named utility enhancing randomization, which allows LATENT to maintain high utility compared to existing LDP protocols. Our experimental evaluation of LATENT on convolutional deep neural networks demonstrates excellent accuracy (e.g., 91%–96%) with high model quality even under low privacy budgets (e.g., $\varepsilon = 0.5$).**

***Index Terms*—Data privacy, deep learning (DL), differential privacy (DP), local DP (LDP).**

I. Introduction

THE Internet of Things (IoT) has become one of the essential assets that transform many industries, such as oil and gas, healthcare, agriculture, finance, and transportation. The production of a large amount of IoT data has led to the advancement of different technologies, such as big data analytics and machine learning (ML), and has opened up new opportunities. The underlying technologies of IoT are continuously improving to address complex heterogeneity and to improve programmable flexibility. Network infrastructural improvements, such as the amalgamation of software-defined networks (SDNs) and network function virtualization (NFV) in the edge-cloud interplay, are promising better quality-of-service (QoS) for complex IoT-driven applications. Such environments can be fully utilized using ML to generate efficient and advanced analytics. However, the server-centric architectures employed by many ML algorithms limit their integration into distributed environments such as SDN-controlled NFV.

Compared to traditional ML approaches, deep learning (DL) shows remarkable success in addressing complex problems, such as image classification, natural language processing, and speech recognition. DL models are often trained on sensitive crowd-sourced data, such as personal images, health records, and financial records. When DL models are trained on massive databases containing sensitive data, they tend to expose private information [1], [2]. With the advancement of distributed, cloud-based ML environments such as those offered by Google and Amazon [3], [4], more users may become vulnerable to such attacks. Trusting these environments, users may feed their data to train the models and obtain white-box or black-box access to these models without being concerned about the actual training process. However, an adversary can easily implement malicious algorithms and offer them as part of the training process. Malicious algorithms may memorize the sensitive user information as part of the trained models. Adversaries can later extract and approximate the memorized information, and thereby obtain information about the users and breach their privacy [5]. Privacy inference attacks, such as membership inference, show the vulnerability of DL models trained on sensitive data even when they are released as black-box models [2]. Another example that shows the weakness of trained ML models is model inversion attacks that recover images from a facial recognition system [6]. It is essential that ML as a service employs sufficient privacy-preserving mechanisms to limit privacy leaks of trained DL models. It is also essential that these privacy-preserving approaches for DL can be used for IoT-based applications, such as smart healthcare, IIoT, and Industry 4.0.

In this article, we examine the privacy issues of DL and develop a distributed privacy-preserving mechanism using differential privacy (DP) to control and limit privacy leaks in DL. DP constitutes a robust framework guaranteeing strong levels

Manuscript received August 31, 2019; revised October 26, 2019; accepted November 3, 2019. Date of publication November 7, 2019; date of current version July 10, 2020. *(Corresponding author: Pathum Chamikara Mahawaga Arachchige.)*

P. C. Mahawaga Arachchige is with the Department of Computer Science and Software Engineering, School of Science, RMIT University, Melbourne, VIC 3000, Australia, and also with CSIRO Data61, Melbourne, VIC 3008, Australia (e-mail: pathumchamikara.mahawagaarachchige@rmit.edu.au).

P. Bertok and I. Khalil are with the Department of Computer Science and Software Engineering, School of Science, RMIT University, Melbourne, VIC 3000, Australia.

D. Liu and S. Camtepe are with CSIRO Data61, Sydney, NSW 2122, Australia.

M. Atiquzzaman is with the School of Computer Science, University of Oklahoma, Norman, OK 73019 USA.

Digital Object Identifier 10.1109/JIOT.2019.2952146

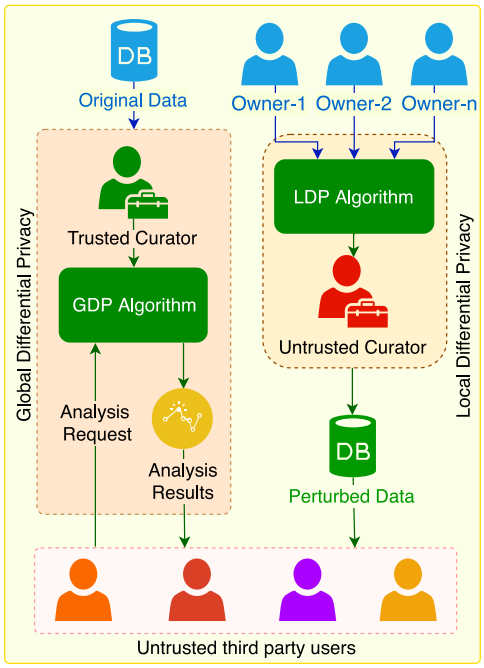

Fig. 1. Global versus local DP.

of privacy [7]. The existing benchmark privacy-preserving approaches for DL are based on global DP (GDP) [1], [8]. However, we chose local DP (LDP) over GDP. As shown in Fig. 1, GDP employs a trusted curator to apply calibrated noise to produce DP [9], [10]. The necessity of a trusted curator makes the existing GDP methods unsuitable for practical DL services such as those offered by Google. In such a scenario, the GDP algorithm should reside in the server, and the original data need to be uploaded to the server for training. This approach can pose a threat to privacy, as an adversary can perform server-centric attacks, such as membership inference and model memorizing attacks. Moreover, DL algorithms are inherently computationally complex, and privacy-preserving solutions on DL models also tend to be complex and need high computational processing power. As a result, GDP algorithms are preferred to run on high-performance computers, and resource-constrained data owners cannot use them in untrusted environments. Furthermore, noise calibration of GDP methods, such as Laplacian and Gaussian mechanisms for DL models, can be complex, indefinite, and produce less accurate results or entail a higher level of privacy leak [1], [8]. However, in LDP, data owners perturb their data before releasing them, which avoids the need for a trusted third party while guaranteeing better privacy, as depicted in Fig. 1 [10]. The local approach to data perturbation in LDP innovates the development of distributed privacy preservation algorithms for many modern distributed scenarios such as those based on IoT.

Our contribution is a distributed LDP mechanism with a new LDP protocol for limiting the privacy leaks of convolutional neural network (CNN) models that are released as black-box models. The proposed algorithm (named LATENT) employs the properties of randomized response [11], a popular survey technique that satisfies local DP. The LDP setting and the layered architecture of LATENT allow privacy-preserving communication between several parties, which is not possible with existing GDP methods for DL. We conducted an in-depth analysis of the existing LDP protocols and devised a new protocol named utility enhancing randomization (UER) that provides better utility than existing LDP protocols. We first improved the optimized unary encoding (OUE) protocol to propose a new LDP protocol named modified OUE (MOUE), which provides enhanced flexibility of binary string randomization. OUE is an LDP protocol that follows the intuition of randomizing 1's and 0's differently to improve utility. MOUE achieves improved flexibility by introducing an additional coefficient α (named as the privacy budget coefficient) that provides improved flexibility in choosing randomization probabilities. We then followed the motivation behind MOUE to propose UER that maintains the utility during the randomization of long binary strings with high sensitivity. LATENT can be easily integrated into modern environments such as SDN controlled NFV by moving the layer of randomization to run as an NFV service. As the LDP approach of LATENT enables the control of the privacy budget before the perturbation process, accuracy can be effectively tuned independently. In other words, LATENT reduces the impact of the privacy budget (ε) on the accuracy, and this leads to significantly higher privacy and accuracy than what is offered by existing solutions. Compared to current GDP methods for DL, LATENT provides excellent accuracy (above 90%) under extreme cases of privacy budgets (e.g., $\varepsilon = 0.5$) that ensure minimum leak. Our experiments clearly show that a general-purpose computer is sufficient to perform the required computations efficiently and reliably at the data owner's end. Accordingly, LATENT can be a more practical and robust tool to limit the privacy leak of DL models than existing methods.

The rest of this article is organized as follows. The underlying concepts used in LATENT are presented in Section II. Section III explains the steps of the differentially private mechanism for DL. The results of LATENT are discussed in Section IV. Section V provides a summary of existing related work. This article is concluded in Section VI.

II. Background

This section provides brief descriptions of the underlying concepts of LATENT. It includes brief summaries of basic principles related to "DP" and to "DL," which are used in LATENT.

A. Differential Privacy

DP is a privacy model that is known to render maximum privacy by minimizing the chance of individual record identification [12]. In principle, DP defines the bounds to how much information can be revealed to a third party/adversary about someone's data being present in a particular database. Conventionally ε (epsilon) and δ (delta) are used to denote these bounds, which describe the level of privacy rendered by a randomized privacy-preserving algorithm (M) over a particular database (D).

1) Privacy Budget/Privacy Loss (ε): ε is called the privacy budget that provides an insight into the privacy loss of a DP

algorithm. The higher the value of ε, the higher the privacy loss.

2) Probability to Fail/Probability of Error (δ): δ is the parameter that accounts for "bad events" that might result in high privacy loss; δ is the probability of the output revealing the identity of a particular individual, which can happen $\delta \times n$ times where n is the number of records. To minimize the risk of privacy loss, $\delta \times n$ has to be maintained at a low value. For example, the probability of a bad event is 1% when $\delta = 1/100 \times n$.

3) Definition of Differential Privacy: Let us take a data set D and two of its adjacent data sets, x and y, where y differs from x only by one person. Assume, data sets x and y as being collections of records from a universe $\mathcal{X}$ and $\mathbb{N}$ denotes the set of all non-negative integers including zero. Then M satisfies (ε, δ)-DP if (1) holds.

Definition 1: A randomized algorithm M with domain $\mathbb{N}^{|\mathcal{X}|}$ and range R: is (ε, δ)-differentially private for $\delta \geq 0$ if for every adjacent data sets x, $y \in \mathbb{N}^{|\mathcal{X}|}$ and for any subset $S \subseteq R$,

$$\Pr[(M(x) \in S)] \leq \exp(\varepsilon)\Pr\big[(M(y) \in S)\big] + \delta. \tag{1}$$

B. Global Versus Local Differential Privacy

GDP and LDP are two approaches that can be used by randomized algorithms to achieve DP. As depicted in Fig. 1, GDP employs a trusted curator who applies carefully calibrated random noise to the real values returned for a particular query. The most frequently used noise generation processes for GDP include the Laplace mechanism and Gaussian mechanism [7]. The GDP setting is also called the trusted curator model [13]. A randomized algorithm, M provides (ε, δ)-global DP if (1) holds. LDP needs no trusted third party, hence it is also called the untrusted curator model [10]. With LDP, data is randomized before the curator can access it. LDP can also be used by a trusted party to randomize all records in a database at once. The right-hand column of Fig. 1 represents the LDP setting. LDP algorithms may often produce too noisy data, as noise is applied commonly to achieve individual data privacy. LDP is considered to be a strong and rigorous notion of privacy that provides plausible deniability. Due to the above properties, LDP is deemed to be a state-of-the-art approach for privacy-preserving data collection and distribution. A randomized algorithm A provides ε-local DP if (2) holds [14].

Definition 2: A randomized algorithm A satisfies ε-local DP if for all pairs of client's values v_1 and v_2 and for all $Q \subseteq \text{Range}(A)$ and for ($\varepsilon \geq 0$), the following equation holds:

$$\Pr[A(v_1) \in Q] \leq \exp(\varepsilon)\Pr[A(v_2) \in Q]. \tag{2}$$

Range(A) is the set of all possible outputs of the randomized algorithm A.

C. Randomized Response

Randomized response is a survey technique to eliminate evasive answer bias by randomizing the responses to a survey question with the answer "yes" or "no" [15]. An answer is randomized by flipping two independent, unbiased coins. The answer is truthful if the first coin comes up "heads," else, the second coin is flipped, and the answer is "yes" if "heads," "no" if "tails." Assume that a biased coin is used and the probability of a user providing an answer truthfully is p [otherwise provides the opposite of the true answer, with $(1 - p)$ probability]. It has been shown that this approach provides ε- DP when $p = e^{\varepsilon}/(1 + e^{\varepsilon})$ [10].

D. Sensitivity, Privacy Budget (ε), and Determination of the Probability (p) of Randomization

To quantify the probability of randomization (p, the probability of preserving an original bit) of an LDP process that is based on transferring bit strings, we can use the method employed by randomized aggregatable privacy-preserving ordinal response (RAPPOR), which is an LDP algorithm proposed by Google [14]. RAPPOR is motivated by the problem of estimating a client-side distribution of string values drawn from a discrete data dictionary. One application of RAPPOR is to track the distribution of users' browser configuration strings in the chrome Web browser.

Sensitivity is defined as the maximum influence that a single individual can have on the result of a numeric query. Consider an arbitrary function f, the sensitivity Δf of f can be given as in the following equation where x and y are two neighboring data sets and $\|.\|_1$ represents the $L1$ norm of a vector [16]:

$$\Delta f = \max\{\|f(x) - f(y)\|_1\}. \tag{3}$$

Since RAPPOR is an LDP algorithm, it considers x and y to be a pair of adjacent inputs in RAPPOR's definition of global sensitivity. In RAPPOR any input v_i is encoded as a vector of d bits, and each d-bit vector contains $d - 1$ zeros and 1 one, so the maximum difference, Δf (the sensitivity) between two adjacent vectors, is 2 bits. In other words, the underlying data representation of RAPPOR, f has a sensitivity of 2. RAPPOR is an LDP algorithm when the probability p of preserving the true value of an original bit in randomization follows

$$p = \frac{e^{\frac{\varepsilon}{\Delta f}}}{1 + e^{\frac{\varepsilon}{\Delta f}}} = \frac{e^{\frac{\varepsilon}{2}}}{1 + e^{\frac{\varepsilon}{2}}} \tag{4}$$

where ε is the privacy budget offered by the LDP process [14], [17].

E. Properties of Differential Privacy

Postprocessing invariance/robustness, quantifiability, and composition are three of the essential characteristics of DP [18]. Although additional computations are carried out on the outcome of a differentially private algorithm, they do not weaken the privacy guarantee. So, the results of additional computations on ε-DP outcome will still be ε-DP. This property of DP is called the postprocessing invariance/robustness. Quantifiability is the ability of DP scenarios to provide transparency in calculating the precise amount of perturbation applied by a particular randomization process. Thus, the user of a particular DP algorithm knows the level of privacy provided by the data/results released after perturbation. Composition is the degradation of privacy when multiple differentially private algorithms are performed on the same or

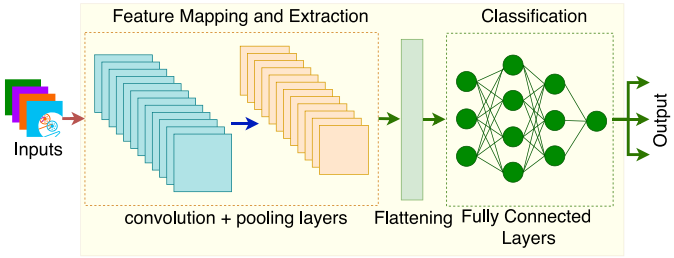

Fig. 2. Generic architecture of a CNN.

overlapping data sets [18]. According to DP definitions, when two DP algorithms ε_1-DP and ε_2-DP are applied to the same or overlapping data sets, the union of the results is equal to $(\varepsilon_1 + \varepsilon_2)$-DP [18]. The more DP algorithms are applied to the same data, the more privacy loss is accumulated. Depending on the process of synthesis, DP algorithms can be categorized into the two types; basic algorithms or derived algorithms [19]. DP is self-contained in basic algorithms while the derived algorithms are derived from existing methods by applying the theories of composition and postprocessing invariance.

F. Deep Learning Using Convolutional Neural Networks

A CNN is commonly trained to recognize essential features of images. As shown in Fig. 2, a CNN uses a collection of layers named convolution layers with large receptive fields. A sequence of steps through this stack of convolution layers is followed by an intermediate functionality called pooling to reduce the dimensions from the previous layer to the next layer. The final pooled output which is produced from the last convolution layer is flattened to produce a sizeable 1-D vector [20]. Then a fully connected artificial neural network (ANN) is trained using these input vectors to generate predictions (classifications) on the inputs (images). An ANN is more or less a connected network of processing modules called neurons, each producing a sequence of real-valued activations.

Overfitting happens when the training accuracy is significantly higher than the testing accuracy [20], [21]. Good quality models avoid overfitting. Regularization, image augmentation, and hyperparameter tuning are three of the commonly used concepts to avoid overfitting and improve the performance and robustness of neural networks [20], [21]. Regularization is the process of applying any modification to a learning algorithm to reduce the generalization error. Regularization can be achieved using dropouts, where a certain percentage of neurons are randomly dropped in each epoch (training cycle) to avoid overfitting. Image augmentation is a data preparation technique, which uses the existing input images in the training data set and manipulates them to create many altered versions of the same input using different transformation methods, such as reflection, sheer, and rotation. This technique allows an ANN to learn a wider variety of inputs to make the trained model more generalizable with high robustness [22]. In hyperparameter tuning the inputs to hyperparameters, such as percentage dropouts, batch size, activation functions, number of neurons, number of epochs, and optimizer are changed under different training phases to identify the best case study that returns the best results [20], [21]. Batch size is the number of training examples that are going to be propagated in one forward/backward pass [21]. Activation functions define the output of a particular neuron given a set of inputs that, with corresponding weights, introduce nonlinear properties to the network [20]. A neuron (also called a node) is the primary component of an ANN. A single pass in which the entire data set is introduced forward and backward through the neural network is called an epoch [20]. An optimizer (or an optimization algorithm) is used to update the model parameters, such as weights and bias values [20].

G. Amalgamation of SDN and NFV in Edge-Cloud Interplay

SDN and NFV are two types of programmable infrastructures that improve the versatility of networking. Both technologies work based on the concept of creating virtual instances for network functionalities where SDN virtualizes controlling aspects, and NFV virtualizes essential network functions such as encrypting channels. The amalgamation between SDN and NFV can bring forward many advanced capabilities in terms of flexibility, efficiency, and QoS with increased reconfigurability. SDN-controlled NFV can introduce a series of virtualizations that can benefit the edge-cloud interplay, which can improve the security and QoS of communication between local devices and cloud servers [23], [24].

III. Our Approach: LATENT

This section discusses the differentially private mechanism employed in LATENT for DL. LATENT can be classified as a derived differentially private algorithm which is based on the randomized response technique. LATENT uses two properties of DP: 1) postprocessing invariance and 2) composition when applying DP to a CNN. LATENT uses regularization, image augmentation, and hyperparameter tuning to optimize its performance under noisy input conditions in the randomization process. We implemented and tested LATENT on CNNs using the Python Keras neural networks API, which runs on top of the TensorFlow dataflow engine developed by Google [3], [25]. Keras provides a high-level neural-network API designed primarily for fast experimentation.

A. Introduction of the Intermediate Layer (LATENT) to Inject Differential Privacy Into the CNN Architecture

As shown in Fig. 3, we divide the structure of a CNN into two main modules and introduce an intermediate module of randomization into the CNN structure. Recall (described in Section II-F) that in CNN, the input features are initially subjected to dimensionality reduction, using a collection of convolutional layers and pooling layers. The output of the final pooling layers is flattened into a single 1-D array before feeding it to a fully connected ANN. We call this part of a CNN the CNM, and we name the fully connected network component of the CNN as the FC module. We insert the randomization layer named LATENT between the CNM and the FC module, as shown in Fig. 3. In the proposed architecture, we use the CNM only to generate the 1-D flattened output that corresponds to a particular image input. This flattened output

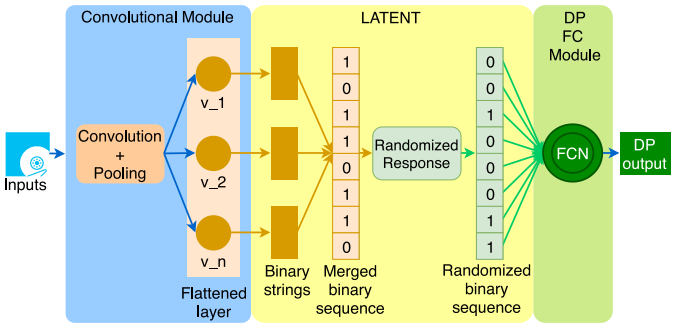

Fig. 3. CNN architecture with the LATENT randomization layer (FCN: fully connected network).

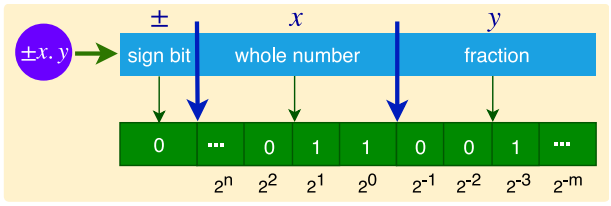

Fig. 4. Direct mapping of a float/integer to binary.

is simply a 1-D column vector of float values (real-valued numbers).

1) Apply z-Score Normalization to LATENT's Input Values: LATENT converts the input values to binary values before randomization. The inputs can have different ranges. Conversion of large values or small fractions into binary can involve a large number of bits. This can introduce an inconsistent level of complexity to the algorithm. To avoid this complexity, we apply z-score normalization to the values of the 1-D vector coming from the flattening layer.

2) Define the Bounds (Lengths of the Segments) for the Binary Conversion: The length of the bit pattern establishes the range of a particular z-score normalized input. The upper bound (UB) and the lower bound of a specific input need to be initially estimated. Fig. 4 shows the arrangement of bits of the binary conversion of a z-score normalized input. As shown in the figure, there are three primary segments of the binary string. The first bit represents the sign of the input (1 for negative and 0 for positive). The other two parts are for the whole number and the fraction part of an input number, respectively. The selection of the number of bits for the whole number depends on the maximum value of the whole number that needs to be represented. Due to z-score normalization, the number of bits necessary to represent the whole number is small. The selection of the number of bits for the fraction depends on the precision (how close is the binary fraction's decimal value to the input's fraction value). For more precision, a higher number of bits needs to be used for the fraction.

3) Convert Each Value of the Flattened Layer to Binary Using the Bounds: After determining the length of the components of binary strings of the inputs, the inputs can be mapped as shown in Fig. 4. The figure shows the direct mapping of an integer/float value to its binary representation. The binary representation can be generated according to

$$g(i) = \left(\left\lfloor 2^{-k} |x| \right\rfloor \bmod 2 \right)_{k=-m}^{n}, \quad \text{where, } i = k + m \quad (5)$$

where n and m are the numbers of binary digits of the whole number and the fraction, respectively, x represents the original input value where $x \in \mathbb{R}$, and $g(i)$ represents the ith bit of the binary string where the least significant bit is represented when $k = -m$. The sign bit is 1 for negative values and 0 for positive values. The sign bit is assigned to the most significant bit of the binary string.

4) Merge the Binary Strings to Reduce the Privacy Loss: We merge all the binary strings into one long binary string to avoid privacy loss due to the composition property of DP. If we conducted the randomization on each binary string corresponding to the flattened 1-D vector separately, it would add up the privacy budgets of all the randomization steps. If r binary strings were randomized, the resulting privacy loss of the final randomization would be $r \times \varepsilon$. As LATENT conducts the randomization on a particular merged binary string at once, we can maintain the privacy loss at the input value of ε.

5) Define the Probability of Randomization (p) in Terms of ε: The probability of randomization p (i.e., the probability of preserving the true value of an original bit) is calculated in terms of the privacy budget (ε) before the randomization of the merged binary string. Recall that in the randomized response technique (as described in Section II-D) used in RAPPOR, when the difference of the number of bits of two neighboring inputs is d, the sensitivity becomes d. In the case of LATENT, the length of a binary string is $l = (n + m + 1)$ which makes the length of the merged binary string equal to $l \times r$ where r is the number of outputs of the flattening layer of the CNM. According to our method of binary conversion, two consecutive inputs can differ by at most of $l \times r$ bits. Consequently, LATENT has a sensitivity of $l \times r$. Now, we can represent the probability of randomization according to

$$p = \frac{e^{\varepsilon/rl}}{1 + e^{\varepsilon/rl}}. \quad (6)$$

We note that merging the binary strings together increases sensitivity, hence, makes increasing the amount of randomization necessary.

6) Modifying Optimized Unary Encoding to Improve Utility: As discussed in Section III-A5, the probability of randomization in reporting opposite of the true bits is $(1 - p) = [1/(1 + e^{\varepsilon/rl})]$. However, this can introduce an unreliable level of randomization to the output of the LATENT layer, as the sensitivity = rl is extensive, as discussed in Section III-A5. To improve the utility, we follow the intuition behind OUE [26]. OUE perturbs 0 and 1 differently to reduce the probability of perturbing 0 to 1 ($p_{0\to1}$) as there are considerably more 0's than 1's when the input binary string is long. We propose a new approach to further optimize the selection of probabilities of randomization, which can provide enhanced utility when the bit strings are long.

Let v_i represent an instance in the database and B is a d bit binary encoded version of v_i. $B[i]$ represents the ith bit and $B'[i]$ is the perturbed ith bit. Assume that, only the jth position

of B is set to 1, whereas the other bits are set to zero. Unary encoding (UE) [14] perturbs the bits of B according to

$$\Pr\big[B'[i]=1\big]=\begin{cases}p, & \text{if } B[i]=1\\ q, & \text{if } B[i]=0.\end{cases} \tag{7}$$

UE satisfies ε-LDP [14], [26] for

$$\varepsilon=\ln\left(\frac{p(1-q)}{(1-p)q}\right). \tag{8}$$

This can be proven (refer to Appendix A) as done in [14] and [26] for any inputs v_1, v_2, and B with sensitivity = 2. Utilizing the concept of UE, OUE in its optimal setting sets $p=(1/2)$ and $q=[1/(1+e^{\varepsilon})]$, so that the randomization improves the budget allocation for transmitting the 0 bits in their original state as much as possible. According to (8), we can show that OUE provides ε-LDP when $p=(1/2)$, $q=[1/(1+e^{\varepsilon})]$, and sensitivity = 2 (refer to Appendix B for proof).

When the sensitivity is high, the bit strings will have many 1s. In such a scenario (such as in LATENT), more flexibility for controlling the randomization of 1s is also essential to generate better utility. Following this intuition, we propose the UB Theorem 1. In UB theorem, we consider a coefficient (α, the privacy budget coefficient) during the probability selection. This model provides ε_{ub}-LDP where $\varepsilon_{ub}=\ln(\alpha^2 e^{\varepsilon})$.

Theorem 1 (UB Theorem): Let UB(ε) be the UB of the privacy budget when the sensitivity = 2. Let the perturbation probabilities, $p=[(\alpha e^{[\varepsilon/2]})/(1+\alpha e^{[\varepsilon/2]})]$ and $q=[1/(1+\alpha e^{[\varepsilon/2]})]$. Then UB($\varepsilon$)) = $\ln(\alpha^2 e^{\varepsilon})$, where α is the privacy budget coefficient (refer to Appendix C for proof).

We can extend the idea in the UB theorem and use the privacy budget coefficient (α) to modify OUE as defined in Theorem 2; α provides more flexibility to MOUE in choosing the randomization probabilities. By increasing α, we can increase the probability of transmitting the 0 bits in their original state.

Theorem 2 (MOUE): For any inputs v_1, v_2 in MOUE, $\Pr[\boldsymbol{B}[v_1]=1|v_1]=(1/1+\alpha)$, $\Pr[\boldsymbol{B}[v_1]=1|v_2]=(\alpha/1+\alpha)$, $\Pr[\boldsymbol{B}[v_2]=0|v_1]=[(\alpha e^{\varepsilon})/(1+\alpha e^{\varepsilon})]$, $\Pr[\boldsymbol{B}[v_2]=0|v_2]=[1/(1+\alpha e^{\varepsilon})]$. Then, MOUE provides ε-LDP (refer to Appendix D for proof).

MOUE is suitable for randomizing bit strings, which has a considerable number of 1s along with a large number of 0s such as in LATENT. Theorem 3 defines the probabilities for cases where the sensitivity is considerably large. Preserving as much 0s as possible will allow the utility to be preserved while tolerating an anticipated loss in the probability of preserving the original 1.

Theorem 3 (MOUE for High Sensitivities): MOUE for LATENT provides ε-LDP, when $\Pr[\boldsymbol{B}[v_1]=1|v_1]=(1/1+\alpha)$, $\Pr[\boldsymbol{B}[v_1]=1|v_2]=(\alpha/1+\alpha)$, $\Pr[\boldsymbol{B}[v_1]=1|v_2]=[(\alpha e^{[\varepsilon/rl]})/(1+\alpha e^{[\varepsilon/rl]})]$, $\Pr[\boldsymbol{B}[v_2]=0|v_2]=[1/(1+\alpha e^{[\varepsilon/rl]})]$ (refer to Appendix E for proof).

7) Improving the Utility of Randomized Binary Strings: Although MOUE improves the utility of randomized output, it can also massively increase the randomization of 1's when α is large. MOUE improves the probabilities of randomization by randomizing 1 and 0 differently following the intuition behind OUE. By further extending this idea, we apply two randomization models over the bits of a binary string to enhance the utility of randomized binary strings. In this way, we try to randomize half of the bits in the bit string differently compared to the other half as defined in Theorem 4. When α is increased, UER will increase the probability of preserving 0s in their original state. However, the probability of preserving 1s in their original state increases for half of the string while the corresponding probability decreases for the other half of the string. In this way, UER maintains the privacy budget at ϵ.

Theorem 4 (UER): Let $p(B[i]v)$ be the probability of randomizing the ith bit of the binary encoded string of v. For any inputs v_1, v_2 with a sensitivity = rl, define the probability, $p(B[i]v)$ as in

$$p(B[i]v)=\begin{cases}\Pr[\boldsymbol{B}[v_1]=1|v_1]=\frac{\alpha}{1+\alpha}, & \text{if } i\in 2n;\ n\in\mathbb{N}\\ \Pr[\boldsymbol{B}[v_2]=0|v_1]=\frac{\alpha e^{\frac{\varepsilon}{rl}}}{1+\alpha e^{\frac{\varepsilon}{rl}}} & ,,\\ \Pr[\boldsymbol{B}[v_1]=1|v_1]=\frac{1}{1+\alpha^3}, & \text{if } i\in 2n+1.\\ \Pr[\boldsymbol{B}[v_2]=0|v_1]=\frac{\alpha e^{\frac{\varepsilon}{rl}}}{1+\alpha e^{\frac{\varepsilon}{rl}}} & ,,\end{cases} \tag{9}$$

Then the randomization provides ϵ-LDP (refer to Appendix F for proof).

8) Conduct UER on the Bits of the Merged Binary Strings: Each bit in the merged binary string is subjected to MOUE, given in (6). The higher the p is, the lower the randomization of the binary string will be. According to (6), higher ε values and lower $l\times r$ values will result in higher p values. In LATENT, $l\times r$ is a considerably larger value than ε, p is often a smaller value. MOUE allows LATENT to fine-tune the probabilities of randomization by calibrating α. By increasing α to be greater than 5, MOUE decreases the probability of perturbing 0 to 1. This feature of MOUE helps LATENT to maintain the utility as there are a large number of 0's compared to 1's in binary encoded inputs of LATENT. However, this reduces the probability of releasing the real bits when the input bit is 1 for half of the bits in the input bit strings. Compared to RAPPOR, a binary string in LATENT has many 1's. As a result of that, LATENT can tolerate this loss of probability. Consequently, UER allows changing the amount of randomization of the input bits by changing α while maintaining the same privacy budget.

9) Generate a Differentially Private Classification Model Using the FC Module: After randomizing the merged binary strings, LATENT feeds the randomized binary strings into the FC module of the convolutional network. The FC module is then trained on the randomized binary strings to generate a differentially private ANN model (DPFC module). We improve the performance of the differentially private model using regularization, image augmentation, and hyperparameter tuning.

B. Algorithm for Generating Differentially Private CNN

Algorithm 1 shows the steps of LATENT in producing a differentially private output. It provides the sequence of steps of applying DP to the CNN architecture, as explained in Section III-A.

Algorithm 1: Differentially Private Model Generation

Input:
- $\{x_1, \ldots, x_j\}$ ← examples
- ε ← privacy budget
- n ← number of bits for the whole number of the binary representation
- m ← number of bits for the fraction of the binary representation
- α ← privacy budget coefficient

Output:
- *DPCNN* ← differentially private CNN model

1 Define the CNM as explained in Section III-A;
2 Declare, $l = (m + n + 1)$;
3 Feed $\{x_1, \ldots, x_j\}$ to the CNM and generate the sequence of 1-D feature arrays $\{d_1, \ldots, d_j\}$;
4 Convert each field (x) of d_q (where, $q = 1, \ldots, j$) to binary using, $g(i) = (\lfloor 2^{-k} |x| \rfloor \mod 2)_{k=-m}^{n}$ where, $i = k + m$;
5 Generate array $\{b_1, \ldots, b_j\}$ of the merged binary arrays for the elements in $\{d_1, \ldots, d_j\}$;
6 Determine the length (r) of a single element of $\{d_1, \ldots, d_j\}$;
7 Calculate randomization probability according to Eq. (9);
8 Randomize each element of $\{b_1, \ldots, b_j\}$ using UER (refer Theorem 4) with probability p to generate $\{pb_1, \ldots, pb_j\}$;
9 Train the FC module of the CNN using $\{pb_1, \ldots, pb_j\}$;
10 Optimize the FC module using regularization, image augmentation and/or hyperparameter tuning;
11 Return the DPFC module;

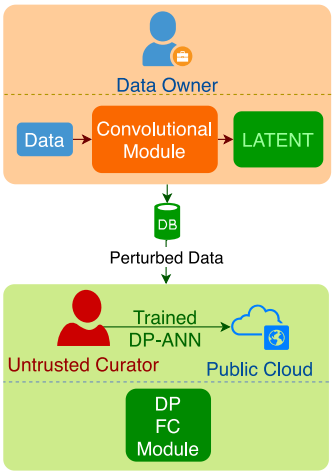

Fig. 5. LDP configurations of LATENT.

C. LDP Settings for LATENT

Since the randomization takes place after the CNM, we push the CNM and the LATENT module to the data owner's end as shown in Fig. 5. The DPFC module is trained at the untrusted curator's end and can be executed on a cloud computer or on any high-performance computing server. The model release will involve the release of only the trained DPFC module that can be used for testing by any third party. In the proposed setting, the CNM is not trained for features, leaving a minimum computational burden on a particular data owner.

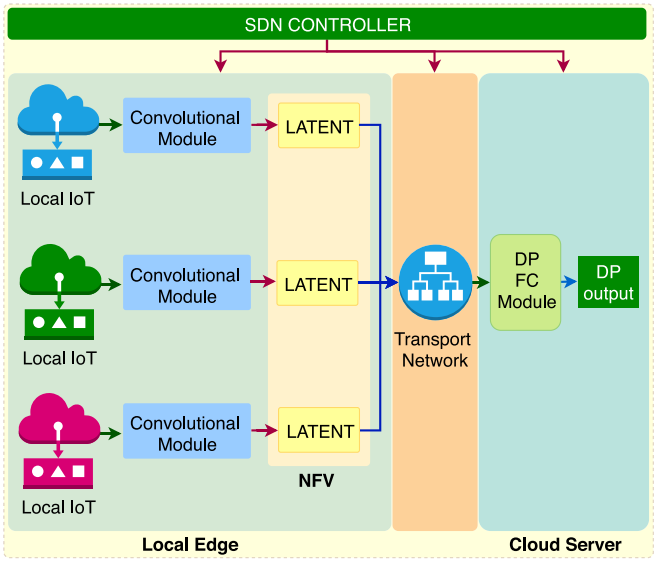

Fig. 6. Integration of LATENT with SDN and NFV in edge-cloud interplay.

The proposed component distribution of the CNN architecture, which moves the CNM to the data owner, produces additional privacy even before the randomization, as the output of the CNN module is a dimension-reduced 1-D vector. Additionally, in the big data context where millions of data owners communicate with the server, our CNN model distribution can provide additional flexibility and efficiency in data processing, leaving the FC module to train on the already dimension-reduced data.

However, we can also push the whole DP CNN architecture to a single machine where we keep a central repository that is maintained by a trusted curator. Then, we can apply the CNN with LATENT on the data set at the trusted curator's end where the model will be released with the whole architecture of the CNN [CNM (untrained)+LATENT+DPFC module].

D. Integrating LATENT in the Amalgamation of SDN and NFV in Edge-Cloud Interplay

We can distribute the layers in LATENT (refer Fig. 3) in an SDN+NFV setting, as shown in Fig. 6. In this configuration, the first two layers of Fig. 6 will reside at the local edge. First, the output from local IoT will go through the CNM. The randomization layer will run as an NFV service which applies the randomization to the outputs from the CNMs. Consequently, the public transport layer will receive a randomized version of the input data and pass them on to the DPFC module in the cloud server, which produces a differentially private output. One or more SDN controllers will control the whole communication setup, as shown in Fig. 6.

IV. Results and Discussion

In this section, we discuss the experiments, experimental configurations, and their results. We tested our method using the MNIST data set [27] and the CIFAR-10 data set [1] which are considered to be the benchmark data

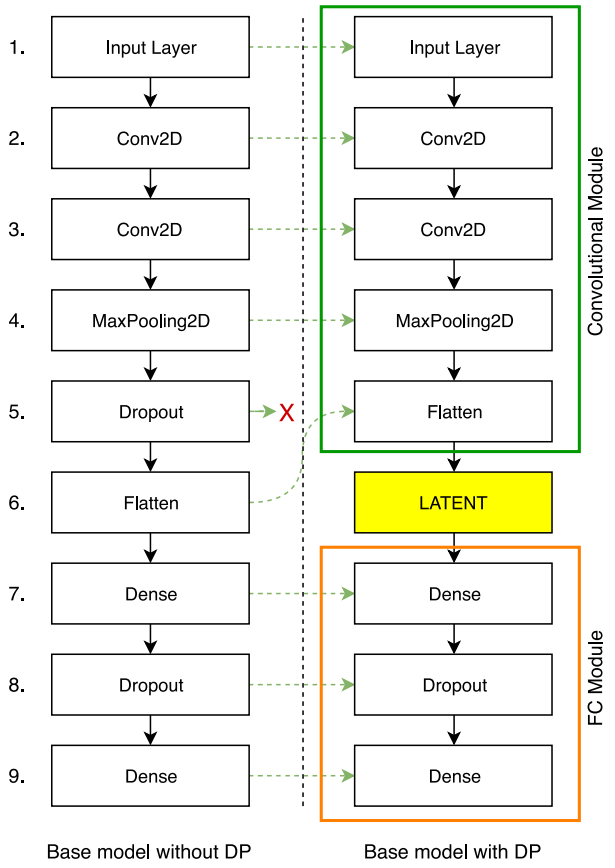

Fig. 7. Architectural differences between the nonprivate (NPCNN) and differentially private (DPCNN) baseline models for the handwriting recognition data set (MNIST data set).

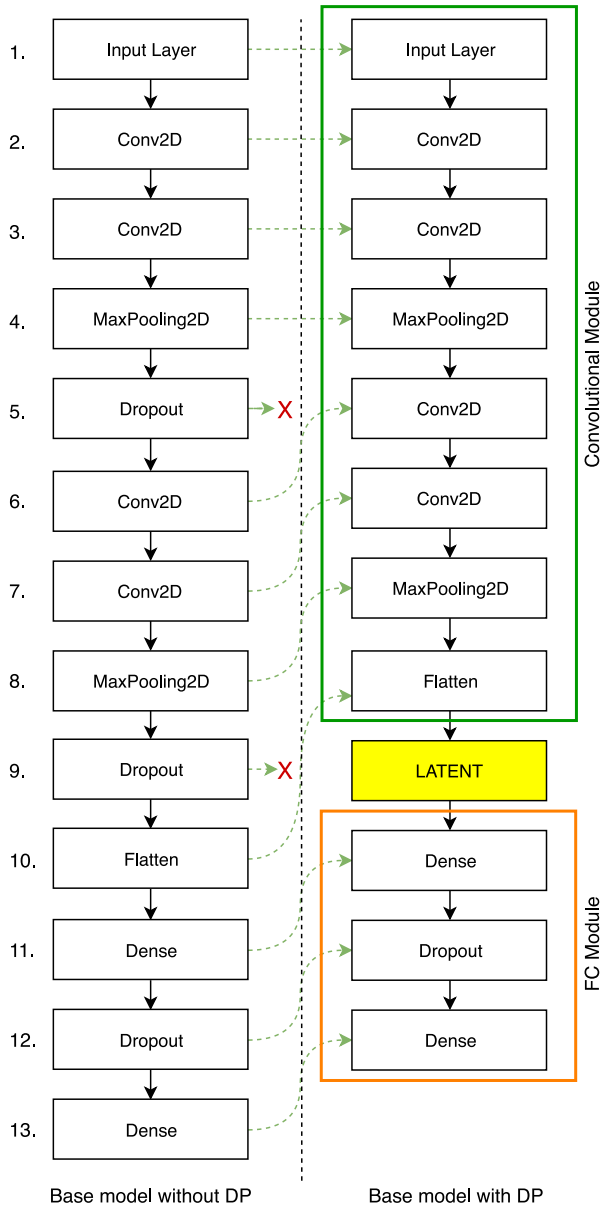

Fig. 8. Architectural differences between the baseline models (NPCNN and DPCNN) for image recognition of the CIFAR-10 data set.

sets to train and test DL (CNN) algorithms. We specifically selected MNIST and CIFAR-10 for the experiments as they have been used in recent works on DL with DP [1], [8]. MNIST is famous for generating good accuracy in DL, whereas CIFAR-10 is a complex data set and is difficult for training. These complementary properties of MNIST and CIFAR-10 provide a balanced experimental setup to test the performance of a specific DL scenario. We conducted all experiments on an HPC cluster (SUSE Linux Enterprise Server 12 SP3) with 112 Dual Xeon 14-core E5-2690 v4 Compute Nodes each with 256 GB of RAM, FDR10 InfiniBand interconnect, and 4 NVidia Tesla P100 (SXM2). The computational complexity and the computational burden of LATENT on resource-constrained data owners were evaluated using a general purpose Intel Core i5 computer. A comprehensive specification of the corresponding computer is provided in Section IV-C.

A. Experimental Setup

First, we created suitable baseline CNN models for each data set. The baseline models include CNN without DP (NPCNN) and a differentially private version of the same configuration (DPCNN). Figs. 7 and 8 show the baseline CNN architectures defined for the MNIST data set and the CIFAR-10 data sets, respectively. In the figures, the left-hand side models represent the NPCNN. The right-hand side models are the differentially private versions (DPCNN) of the corresponding left-hand side models. First we tested the accuracy of the NPCNN models, then we tested the DPCNN models' performance relative to NPCNN models, and conducted hyperparameter tuning and image augmentation on the DPCNN models to improve performance. Finally, the results of the best DPCNN models were chosen to compare the results with other existing differentially private methods for DL.

1) Data Sets and CNN Model Information: This section provides information about the data sets and the architectures of the corresponding CNN models used in the experiments. The architecture of a CNN needs to be custom configured, as the performance depends on the characteristics of the input data set. The model quality of the trained ANN depends on

the correct configuration of its network architecture [20]. As explained below, we declared suitable CNN architectures for the two data sets separately because CIFAR-10 is a more complex data set than MNIST.

a) MNIST: The MNIST data set is composed of 70 000 grayscale handwritten digits, where 60 000 examples are used for training, and 10 000 are used for testing. Each image has a resolution of 28×28. The digits have been size-normalized and centered in a fixed-size image [27]. Fig. 7 depicts the CNN network architecture used in the baseline models for the MNIST data set. The figure shows the sequence of the layers of the network architectures of the baseline models. The network accepts 28×28 input images. The convolutional layer (layer 2) uses $32, 3 \times 3$ filters with stride 1 followed by a second convolutional layer (layer 3) which uses $64, 3 \times 3$ filters with stride 1. Both layers 2 and 3 use ReLU as the activation function. The output of layer 3 is subjected to a max pooling layer with 2×2 max pools. Thus, the max pooling layer outputs a $12 \times 12 \times 64$ tensor for each image. Next, the output of the max pooling layer is subjected to a dropout of 25% (layer 5) and flattened (layer 6) to a 1-D vector of size 9216. The output of the flattening layer is fed into a fully connected layer (layer 7) with 128 neurons with ReLU activation function, followed by a dropout of 50%. The output of the dropout layer is finally fed into a fully connected layer with 10 neurons which produces the final output of the CNN network. This model (NPCNN) achieves 99.25% training and 98.16% testing accuracies after 12 epochs of training with a batch size of 128 using the Adadelta optimization algorithm.

b) DPCNN for MNIST: The DPCNN has an additional layer: LATENT (layer number 6, colored in yellow) of randomization in between the CNM and the FC module. The green square represents the CNM, and the orange square represents the DPFC module. If the LATENT layer uses 10 bits to represent one element (one output of the flattening layer) coming from the fattening layer, the length of the randomized bit string generated by the LATENT layer is equal to 10 times the number of outputs of the flattening layer. In this case, the length of the randomized bit string will be equal to $9216 \times 10 = 92\,160$. The green arrows are used to indicate that the same configuration of the corresponding layer is available in the DPCNN. The red cross is used to indicate that the corresponding layer was omitted from the DPCNN. As shown in the figure, we do not use dropouts in the CNM of the DPCNN, as the CNM is not trained for the input features, which is explained in Section III-A. In the DPCNN experiments with the MNIST data set, we maintained a fixed size of 10 bits to represent each output of the flattening layer. The 10 bits are composed of 4 bits for the whole number, 5 bits for the fraction and 1 bit for the sign.

c) CIFAR-10: The CIFAR-10 data set consists of 60 000 color images and 10 classes (airplane, automobile, bird, cat, deer, dog, frog, horse, ship, and truck), with 6000 images per class. There are 50 000 training images and 10 000 testing images. Each image has a resolution of 32×32 [1]. Fig. 8 depicts the CNN architecture used in the baseline models for the CIFAR-10 data set. As CIFAR-10 is a complex data set compared to MNIST, for CIFAR-10 we considered a more complex CNN architecture that involves more layers and more neurons. The network accepts 32×32, 3 channel input images. The convolutional layer (layer 2) uses $32, 3 \times 3$ filters with stride 1 followed by a second convolutional layer (layer 3) which uses $32, 3 \times 3$ filters with stride 1. Both layers 2 and 3 use ReLU as the activation function. The output of layer 3 is subjected to a max pooling layer with 2×2 max pools. The output of layer 4 is fed to a dropout layer (layer 5) with 25% dropout. The output of layer 5 was subjected to two other convolutional layers (layers 6 and 7) which use $64, 3 \times 3$ filters. The output of layer 6 was next introduced with a max pooling layer (layer 8) with 2×2 max pools. Thus, the max pooling layer outputs a $6 \times 6 \times 64$ tensor for each image. Next, the output of the max pooling layer is subjected to a dropout of 25% (layer 9) and flattened (layer 10) to a 1-D vector of size 2304. The output of the flattening layer is fed into a fully connected layer (layer 11) with 512 neurons with ReLU activation function, followed by a dropout of 50%. The output of the dropout layer is finally fed into a fully connected layer with 10 neurons which produces the final output of the CNN network. This model (NPCNN) achieves 73.32% training and 78.75% testing accuracies after 100 epochs of training for a batch size of 32 using the Adadelta optimization algorithm.

d) DPCNN for CIFAR-10: The right-hand side figure of Fig. 8 represents the DPCNN for the CIFAR-10 data set. The DPCNN model creation follows the same approach explained under the model creation process for the MNIST data set. However, we increased the resolution of the images to 56×56 to enhance the features. As a result of that, the input layer (layer 1) was customized to accept $56 \times 56, 3$ channel input images. We do not use dropouts in the CNM for reasons explained in Section III-A. In the experiments with the DPCNN model for the CIFAR-10 data set, we maintained a fixed size of 10 bits to represent each output of the flattening layer. The 10 bits are composed of 2 bits for the whole number, 7 bits for the fraction and 1 bit for the sign.

2) Hyperparameter Tuning and Regularization: As explained in Section III-A, we conducted the training only on the FC module. Therefore, we apply hyperparameter tuning and regularization only on the FC module. Given the number of possible values for each hyperparameter, the number of test cases can become large, and it may entail a substantial computational cost with exponential time. Therefore, it can be imperative to use insights such as used by [1] to minimize the number of hyperparameter settings that need to be tested. Since LATENT does not change the internal parameters of the FC module, the architectural modifications necessary can be thought of as an independent procedure that can be common in any ANN training process. Consequently, we tested different combinations for percentage dropouts, batch sizes, activation functions, number of neurons, optimizers, number of epochs, for the hyperparameter tuning process as given in Table I.

The values used for the hyperparameters during the HPT processes with each data set (MNIST and CIFAR-10) are given in Table I. We preferred a higher number of neurons and epochs for CIFAR-10 due to its complexity compared to

TABLE I
LIST OF VALUES APPLIED FOR EACH HYPERPARAMETER IN THE TEST CASE GENERATION PROCESS OF HYPERPARAMETER TUNING

Hyperparameter	Hyperparameter settings for MNIST	Hyperparameter settings for CIFAR-10
percentage dropout	layer 8 => {20%, 40%, 50%}	layer 11 => {20%, 40%, 50%}
batch size	{ 128, 256, 512}	{400, 500, 600}
activation function	layer 7=> {relu, tanh, sigmoid}	layer 10=> {relu, tanh, sigmoid}
the number of neurons	layer 7=> {64, 128, 256, 512}	layer 10=> {256, 512, 1024}
number of epochs	{50, 100, 150}	{100, 200, 300}
optimizer	{SGD, Adadelta, Adam}	{SGD, Adadelta, Adam}

MNIST. During the hyperparameter tuning process, the probability of preserving the true value of an original bit (p) was set to 1, which corresponds to the nonprivate state of LATENT. When $p = 1$, the binary feature vectors will not be randomized, and the model will result in greater accuracy than when $p < 1$. Due to the enlarged feature space compared to the conventional 1-D output of the flattening layer, the input features will have more representative properties. These properties will allow the proposed architecture of CNN to generate better accuracy than that of the original CNN architecture without LATENT. We generated the test cases using the combinations of parameter values. We applied k-fold cross-validation ($k = 10$) on each test case to derive a fair set of accuracy results. Since the search space is large, we divided the list of test cases into nine groups based on the combinations of the optimizers and the activation function. The best parameter values returned for each data set are colored in red in Table I. Next, we used the best parameters returned by the tuning process to produce differentially private models (for MNIST and CIFAR-10) with the best performance. We used the final DPCNN models of the hyperparameter tuning process to carry out further experiments and analyzes.

3) Image Augmentation to Improve Robustness of the DPCNN Trained Using CIFAR-10: Although we could improve the accuracy of the DPCNN for CIFAR-10 using hyperparameter tuning, the model was still not performing well and tended to overfit. To improve the model robustness, we applied image augmentation and generated 150 000 additional augmented images using the 50 000 training input images. Each augmented image was generated by applying a random horizontal shift of a 0.1 fraction of the total width, a random vertical shift of a 0.1 fraction of the total height, a random rotation of 10 degrees, and a random horizontal flip, on the original input images. After introducing the new augmented images, the DPFC module stopped overfitting and started generating a training accuracy of around 98% and a testing accuracy of about 95% consistently for repeated attempts (under a randomization probability of 1).

4) Selection of ε: As we discussed in Section III-A, the probability of randomization can be given by (6). When $l = 10$ for the DPCNN architecture defined for the MNIST data set (depicted in Fig. 7), $p = [(e^{\varepsilon/92160})/(1 + e^{\varepsilon/92160})]$ as the sensitivity of the DP mechanism is 92 160. For the DPCNN defined for the CIFAR-10 data set (depicted in Fig. 8) also, $p = [(e^{\varepsilon/92160})/(1 + e^{\varepsilon/92160})]$, since the sensitivity is 92 160 due to the increased resolution of the input images from 32×32 to 56×56. Because of the large sensitivity, the effect of ε in generating the randomization probabilities is low. When α is maintained at 1, the probability of randomization (p) lies around 0.5 for the acceptable values of ε (less than 10) as the sensitivity of the processes for both models are large [refer (9)]. Hence, we used $\varepsilon = 0.5$ to generate the results in the experiments. We maintained α at 7 (unless mentioned otherwise), which was the lowest α value that generated reliable convergence of the models.

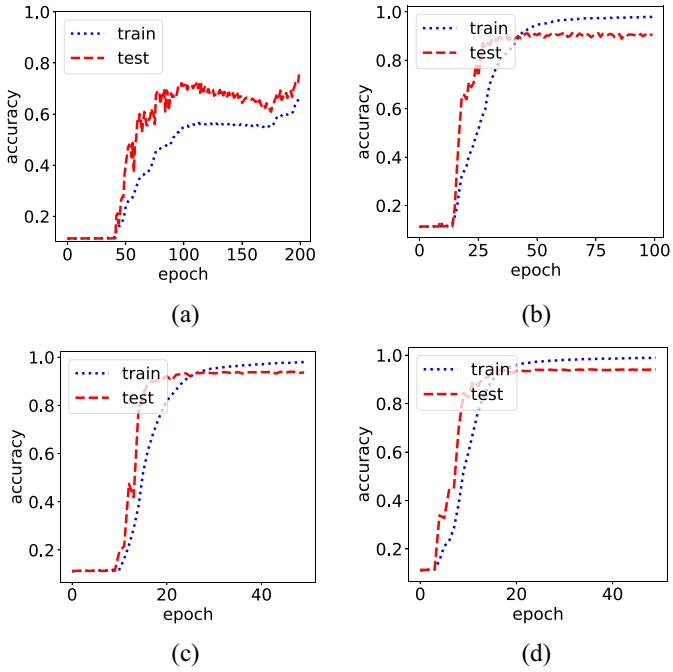

Fig. 9. Model convergence of the FC module for the MNIST data set (under $\varepsilon = 0.5$ and the chosen hyper-parameters which are red-colored in Table I for MNIST) under different α values (randomization levels). FC module training with (a) $a = 5$, (b) $a = 6$, (c) $a = 7$, and $a = 8$.

Fig. 9 shows the model convergence against different α values against the number of epochs during the training process of the FC module on the MNIST data set when $\varepsilon = 0.5$. The FC module converges at different number of epochs against the different levels of randomizations introduced by the various values of α. The converged models provide excellent training and testing accuracies as shown in the plots of Fig. 9. Clarity of the MNIST data set and the availability of a large feature space generated by LATENT allow the model to produce excellent accuracies: around 98%–99% for training and around 95%–96% for testing even at very low ε values, such as 0.5 and $\alpha > 5$ under the chosen hyper-parameters which are red-colored in Table I.

Fig. 10 shows the change of accuracy against the number of epochs during the training process of the FC module of the CIFAR-10 data set when $\varepsilon = 0.5$. After increasing the input image resolution and applying image augmentation to generate 150 000 new images under the best-chosen hyper parameters (red-colored in Table I), the trained model returned around

TABLE II
ACCURACY COMPARISON OF THE RESULTS OF LATENT AGAINST THE EXISTING METHODS

Dataset		NPCNN	[SS15] [8]	[ACG+16] [1]		LATENT	
		accuracy of the model without privacy	ε is large as it is reported per parameter	$\varepsilon = 2$ $\delta = 10^{-5}$	$\varepsilon = 0.5$ $\delta = 10^{-5}$	$\varepsilon = 2$	$\varepsilon = 0.5$
MNIST	Training	99.25%	N/A	~95%	~89%	98.46%	98.97%
	Testing	98.16%	98%	95%	90%	95.67%	96.37%
CIFAR-10	Training	73.32%	N/A	~68%	N/A	95.62%	95.01%
	Testing	78.75%	N/A	67%	N/A	91.34%	91.47%

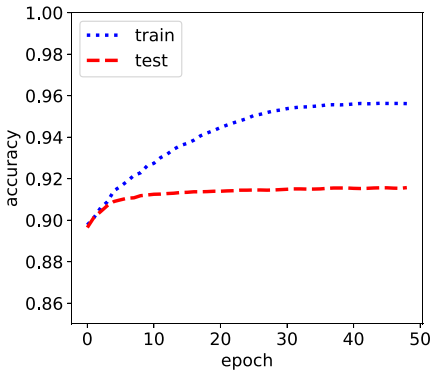

Fig. 10. Change of accuracy versus the number of epochs during the training of the DPFC module for the CIFAR-10 data set (under $\varepsilon = 0.5$ and the chosen hyper-parameters which are red-colored in Table I for CIFAR-10).

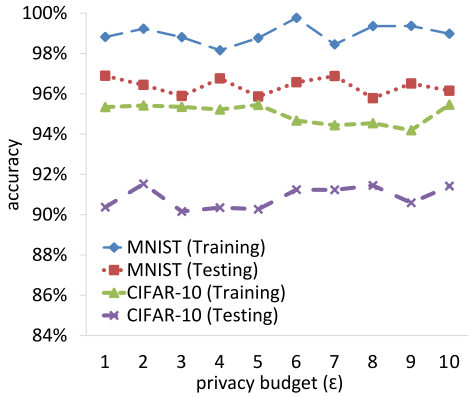

Fig. 11. Change of accuracy of LATENT against ε.

TABLE III
ADVANTAGES OF LATENT OVER THE EXISTING GDP METHODS FOR DL

GDP methods	LATENT
Always needs a trusted curator.	LATENT can be used for both trusted and untrusted settings.
For machine learning with cloud computing, original data needs to be uploaded to the server considering the server is trustable. However, the servers cannot always be trusted.	LATENT randomizes data before uploading them to the server in case the server is not trustable.
A higher privacy loss (a larger privacy budg et - ε) needs to be allocated to obtain a better utility.	LATENT provides excellent utility in terms of classification accuracy (more than 90%) even under an extreme level of randomization ($\varepsilon = 0.5$).
GDP runs either in client side or a server side. The distrust of the server might prevent the algorithm being run on a server. However, deep learning algorithms tend to be complex and can be complex for a general purpose personal computer, and privacy preservation techniques often add more complexity. This feature reduces the practicality of GDP algorithms for deep learning.	As LATENT is an LDP algorithm, it doesn't have an obligation to have a trusted curator. As the proposed architecture is already a distributed version which utilizes the computational power of data owners and the servers, LATENT is more practical compared to GDP approaches.

95%–96% training accuracy and about 90%-91% testing accuracy after 50 epochs. The significant feature space generated by LATENT, and the large input space created by image augmentation allow the final model to produce the corresponding excellent accuracies with high robustness of the DPFC model.

Fig. 11 shows the change of accuracy against ε values. As the figure depicts, accuracy is almost constant although ε is changed. Recall that the probability of randomization is loosely affected by small values of ε (when α is constant) due to the high sensitivity values. When α is kept constant ($= 7$), LATENT maintains a uniform level of randomization on each data set under each case of ε depicted in Fig. 11 and produces similar accuracy for smaller values of ε (<10).

B. Comparison of the Results of LATENT Against the Existing Approaches

We compare our results with two other existing differentially private mechanisms for DL, as shown in Table II. Although [SS15] [8] provides good accuracy; ε is presented as a parameter of the model. It can accumulate a large, unacceptable ε value at the end of model generation as there can be more than 1000 model parameters. For $\varepsilon = 2$ and $\delta = 10^{-5}$ the [ACG+16] [1] method provides good accuracy, yet the additive bound of δ can become unreliable when the method is used for much larger data sets. [ACG+16] has failed to generate acceptable accuracy for the CIFAR-10 data set in an extreme case like $\varepsilon = 0.5$. Our method provides much better accuracy for an extremely small privacy budget such as $\varepsilon = 0.5$. Also, the unavailability of additive bound ensures that our method has a low privacy leak when substantially large input data sets

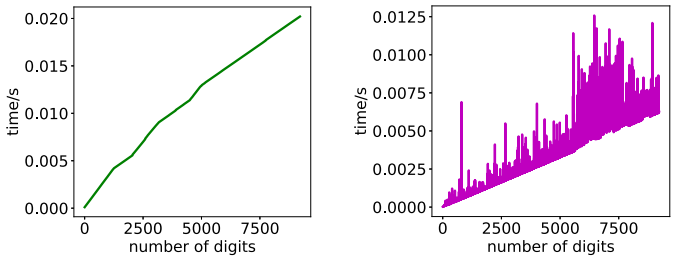

Fig. 12. Time consumption analysis for the (a) binary encoding and (b) randomization steps of LATENT against the number of digits generated by the CNM.

are presented to the method. Both [SS15] and [ACG+16] are based on global DP. Therefore, the availability of a trusted party is unavoidable. For a real-world scenario, often we do not have any trusted party. LATENT can be a much better solution in such cases, as it works with both untrusted and trusted curators. Table III sums up the advantages of LATENT over the existing GDP methods for DL.

C. Computational Complexity and Computational Burden on Generic Users

In this section, we investigate the computational complexity of the proposed algorithm and its burden on a resource-constrained computer. For these two experiments, we used a MacBook Pro (macOS Mojave, 13-inch, 2017) personnel computer (PC) with Intel Core i5 CPU (2.3 GHz), 8 GB RAM and 1536 MB GPU (Intel Iris Plus Graphics), with the assumption that a general-purpose computer is enough to work as an aggregator in a multisensor setup.

LATENT involves two main steps that govern its computation complexity: 1) binary conversion and 2) randomization of the CNM output. Each digit in the CNM output is subjected to binary conversion according to (5). The parameters of the binary conversion [m and n, refer to (5)] are fixed at the beginning of Algorithm 1. Therefore, the computational complexity of the binary conversion step depends on the number of digits (l) in the convolutional output. The computational complexity of step 1 (binary conversion) can be derived as $O(l)$, i.e., is linear. This can be empirically proven by Fig. 12(a). The length of the binary string is determined by the values initialized to m and n. An l sized vector will produce a bit string of $l \times (m + n + 1)$ (refer to Section III-A3). Given that the randomization of each bit is independent, the complexity of the randomization step is also governed by l, which introduces a computational complexity of $O(l)$ as represented by Fig. 12(b).

In the proposed modular decomposition of the CNN architecture, the CNM and the LATENT module run on the data owner's machine. We need to make sure that the CNM and LATENT operations do not impose a substantial computational burden on the resource-constrained data owners. In order to check this, we measured the time consumption for the perturbation of a single record of MNIST and CIFAR-10 data sets separately. It took an average time of 0.1655 s to perturb a single record of MNIST data set while consuming 0.0374 s to perturb a single record of CIFAR-10 data set. This indicates that a general-purpose computer with moderate specification will suffice for generating the randomized data. We can conclude that the CNM and the randomization module of LATENT are feasible to be implemented in any IoT setting or equivalent in the cloud environment.

V. Related Work

Privacy-preserving data mining (PPDM) provides the capability of using the data mining techniques without disclosing private information of the participating entities in the underlying database. However, the main challenge in PPDM is countering the capabilities of skilled adversaries [28], [29]. To overcome this challenge, PPDM uses data modification (data perturbation) [30], [31] and encryption [32] techniques. Data encryption-based approaches provide good security and accuracy. However, cryptographic methods often suffer from high computational complexity, which makes encryption unsuitable for large-scale data mining [33]. Compared to encryption, data perturbation utilizes lower computational complexity, which makes it effective for big data mining [34]. Noise addition, geometric transformation, randomization, data condensation, hybrid perturbation (i.e., using several perturbation techniques together) are some examples of data perturbation techniques [35].

The data perturbation techniques preserve the original format of the input data or the output results. As a result of that, data perturbation techniques often allow some privacy leak [34]. A privacy model defines and identifies the limits of private information protection and disclosure of a certain perturbation mechanism [36]. Earlier privacy models include k-anonymity, l-diversity, (α, k)-anonymity, t-closeness [37], [38]. It was shown that these models are vulnerable to different attacks, such as minimality attack [39], composition-based attacks [40], and foreground knowledge [41]. These attacks exploit the perturbed data to reconstruct private information. DP is a strong privacy model that is trusted to provide a better level of privacy guarantee compared to previous privacy models [42], [43].

There are different data perturbation approaches to achieve DP. Laplace mechanism, Gaussian mechanism [19], geometric mechanism, randomized response [17], and staircase mechanisms [10] are a few of the fundamental mechanisms used to achieve DP. There are many practical examples where these fundamental mechanisms have been used to build differentially private algorithms. DP for SQL Queries [44], LDPMiner [17], PINQ [45], RAPPOR [14], Succinct histogram [46], and DL with DP [1] are a few examples of such practical applications.

Literature shows a few attempts to address the issue of privacy leaks in DL algorithms by imposing the private training [1], [8], [47]–[49]. Shokri and Shmatikov [8] developed a distributed multiparty learning mechanism (referred to as [SS15] in Table II) for a neural network without sharing input data sets. They parallelized the learning process, which is based on the stochastic gradient descent

optimization algorithm. In their method, the privacy loss is calculated per parameter of the model. This feature can entail a substantial privacy loss as there are many model parameters, often there can be thousands of such model parameters. Abadi *et al.* [1] introduced an efficient differentially private mechanism (referred to as [ACG+16] in Table II) based on global DP. Their model is capable of achieving high efficiency and performance under a modest privacy budget. Their algorithm is based on a differentially private version of stochastic gradient descent, which runs on the TensorFlow software library for ML. Further, they introduced a mechanism to track privacy loss, the moments accountant, which allows tight automated analysis of privacy loss. The additive bound δ of their (ε, δ)-DP mechanism may incur an unreliable level of privacy leak when the method is used for much larger data sets. Another shortcoming of the two methods [SS15] and [ACG+16] is the need for a trusted third party. Since both approaches are based on global DP, the necessity of having a trusted third party cannot be avoided. This can be considered as a significant issue in applying these methods to real-world scenarios, where trusted curators are not always available.

LATENT is designed to be aligned with machine-learning-as-a-service scenario, which has become popular due to the capabilities offered by large Internet-based companies, such as Google and Amazon [3]. For example Google's cloud-based ML engine provides the ability to build the models with multiple ML frameworks, such as scikit-learn [50], XGBoost [51], Keras [25], and TensorFlow [3]. LATENT uses similar technologies for its implementation, replicating the technical settings of the environment offered by Google's cloud ML platform and other related services.

VI. Conclusion

We proposed a new LDP mechanism to train a deep neural network with high privacy and high accuracy. Our model exhibits excellent accuracy even under extremely low privacy budgets (e.g., $\varepsilon = 0.5$) compared to existing differentially private approaches. We achieve 95%–96% testing accuracy and 90%–91% testing accuracy for the MNIST data set and CIFAR-10 data set, respectively, with a high level of privacy (0.5-DP). Due to the large feature space created by LATENT during the randomization process, it generates better accuracy for the CIFAR-10 data set than the baseline CNN model without any privacy. Existing differentially private mechanisms are implemented using global DP, and so they need a trusted curator. The untrusted curator setting of our approach provides a higher level of privacy while leaving a low level of computational burden to the data owners. Moving the CNM to the data owners produces additional privacy even without the application of randomization as the CNM output is a 1-D dimension-reduced output. The distribution of the CNN structure between data owners and servers also increases the flexibility of data processing in the big data context. This distribution also helps LATENT be easily adapted to innovations, such as the amalgamation of SDN and NFV in edge-cloud interplay. When a large number of data owners communicate with a single server, the server has to be concerned only about generating the differentially private ANN model. The ability to use our method in the untrusted curator setting allows the private sharing of sensitive data and limits the privacy leak in distributed ML scenarios. Since the proposed method is based on LDP, we do not make any architectural modifications to the fully connected ANN component (which we call the FC module) of a convolutional network. Therefore, the input parameter selection (e.g., ε, α, and the number of input bits) of the differentially private component (LATENT) is independent of the tuning processes (e.g., regularization, image augmentation, and hyperparameter tuning) of the FC module in the CNN architecture. This allows easy training and tuning of the FC module with a higher level of accuracy and an extreme level of privacy, resulting in an outstanding balance between privacy and utility.

Our approach opens up many future research directions. Investigating the possibility of reducing data sensitivity will be a good research avenue. Low sensitivity will allow the selection of an appropriate ε value tailored to domain requirements. We would also like to test our method on other DL architectures such as recurrent networks with long short-term memory (LSTM) and test it for other large data sets to find its performance and generalizability.

Appendix A
Proof of Unary Encoding

Proof: Considering a sensitivity of 2, choose p and q as follows:

$$p = \frac{e^{\frac{\varepsilon}{2}}}{1 + e^{\frac{\varepsilon}{2}}} \tag{10}$$

$$q = \frac{1}{1 + e^{\frac{\varepsilon}{2}}} \tag{11}$$

$$\begin{aligned}\frac{\Pr[\boldsymbol{B}|v_1]}{\Pr[\boldsymbol{B}|v_2]} &= \frac{\prod_{i\in[d]} \Pr[\boldsymbol{B}[i]|v_1]}{\prod_{i\in[d]} \Pr[\boldsymbol{B}[i]|v_2]} \\ &\leq \frac{\Pr[\boldsymbol{B}[v_1] = 1|v_1] \Pr[\boldsymbol{B}[v_2] = 0|v_1]}{\Pr[\boldsymbol{B}[v_1] = 1|v_2] \Pr[\boldsymbol{B}[v_2] = 0|v_2]} \\ &= \frac{p}{q} \cdot \frac{1-q}{1-p} = e^{\varepsilon}. \end{aligned} \tag{12}$$

■

Appendix B
Proof of Optimized Unary Encoding

Proof: Considering a sensitivity of 2, define

$$p = \frac{1}{2} \tag{13}$$

$$q = \frac{1}{1 + e^{\varepsilon}} \tag{14}$$

$$\begin{aligned}\frac{\Pr[\boldsymbol{B}|v_1]}{\Pr[\boldsymbol{B}|v_2]} &= \frac{\prod_{i\in[d]} \Pr[\boldsymbol{B}[i]|v_1]}{\prod_{i\in[d]} \Pr[\boldsymbol{B}[i]|v_2]} \\ &\leq \frac{\Pr[\boldsymbol{B}[v_1] = 1|v_1] \Pr[\boldsymbol{B}[v_2] = 0|v_1]}{\Pr[\boldsymbol{B}[v_1] = 1|v_2] \Pr[\boldsymbol{B}[v_2] = 0|v_2]} \\ &= \frac{p}{q} \cdot \frac{1-q}{1-p} = e^{\varepsilon}. \end{aligned} \tag{15}$$

■

APPENDIX C
PROOF OF THE UPPER BOUND THEOREM

Proof: Considering a sensitivity of 2, choose p and q of (7) according, respectively, to

$$p = \frac{\alpha e^{\frac{\varepsilon}{2}}}{1+\alpha e^{\frac{\varepsilon}{2}}} \tag{16}$$

$$q = \frac{1}{1+\alpha e^{\frac{\varepsilon}{2}}}. \tag{17}$$

According to (8), we can write

$$\varepsilon = \ln\left(\frac{\left(\frac{\alpha e^{\frac{\varepsilon}{2}}}{1+\alpha e^{\frac{\varepsilon}{2}}}\right)\left(1-\frac{1}{1+\alpha e^{\frac{\varepsilon}{2}}}\right)}{\left(1-\frac{\alpha e^{\frac{\varepsilon}{2}}}{1+\alpha e^{\frac{\varepsilon}{2}}}\right)\left(\frac{1}{1+\alpha e^{\frac{\varepsilon}{2}}}\right)}\right)$$

$$\varepsilon = \ln\left(\alpha^2 e^{\varepsilon}\right). \tag{18}$$

Therefore, $\mathrm{UB}(\varepsilon) = \ln(\alpha^2 e^{\varepsilon})$. ∎

APPENDIX D
PROOF OF MODIFIED OUE

Proof: Considering a sensitivity of 2, let

$$p = \frac{1}{1+\alpha} \tag{19}$$

$$q = \frac{1}{1+\alpha e^{\varepsilon}} \tag{20}$$

$$\begin{aligned}\frac{\Pr[\boldsymbol{B}|v_1]}{\Pr[\boldsymbol{B}|v_2]} &= \frac{\prod_{i\in[d]}\Pr[\boldsymbol{B}[i]|v_1]}{\prod_{i\in[d]}\Pr[\boldsymbol{B}[i]|v_2]} \\ &\le \frac{\Pr[\boldsymbol{B}[v_1]=1|v_1]\Pr[\boldsymbol{B}[v_2]=0|v_1]}{\Pr[\boldsymbol{B}[v_1]=1|v_2]\Pr[\boldsymbol{B}[v_2]=0|v_2]} \\ &= \frac{\left(\frac{1}{1+\alpha}\right)}{\left(\frac{\alpha}{1+\alpha}\right)}\cdot\frac{\left(\frac{\alpha e^{\varepsilon}}{1+\alpha e^{\varepsilon}}\right)}{\left(\frac{1}{1+\alpha e^{\varepsilon}}\right)} = e^{\varepsilon}.\end{aligned} \tag{21}$$

∎

APPENDIX E
PROOF OF ε-LDP FOR MOUE FOR HIGH SENSITIVITIES

Proof: Given that LATENT has a sensitivity of $r \times l$ (refer Section III-A1), the privacy budget (ϵ) needs to be divided by the sensitivity for each bit as proven by RAPPOR. For MOUE, $\Pr[\boldsymbol{B}[v_1]=1|v_1] = (1/1+\alpha)$ and $\Pr[\boldsymbol{B}[v_1]=1|v_2] = (\alpha/1+\alpha)$. Hence

$$\Pr[\boldsymbol{B}[v_1]=1|v_2] = \frac{\alpha e^{\frac{\varepsilon}{rl}}}{1+\alpha e^{\frac{\varepsilon}{rl}}} \tag{22}$$

$$\Pr[\boldsymbol{B}[v_2]=0|v_2] = \frac{1}{1+\alpha e^{\frac{\varepsilon}{rl}}}. \tag{23}$$

Therefore

$$\begin{aligned}\frac{\Pr[\boldsymbol{B}|v_1]}{\Pr[\boldsymbol{B}|v_2]} &= \frac{\prod_{i\in[d]}\Pr[\boldsymbol{B}[i]|v_1]}{\prod_{i\in[d]}\Pr[\boldsymbol{B}[i]|v_2]} \\ &\le \left(\frac{\Pr[\boldsymbol{B}[v_1]=1|v_1]\Pr[\boldsymbol{B}[v_2]=0|v_1]}{\Pr[\boldsymbol{B}[v_1]=1|v_2]\Pr[\boldsymbol{B}[v_2]=0|v_2]}\right)^{rl} \\ &= \left(\frac{\left(\frac{1}{1+\alpha}\right)}{\left(\frac{\alpha}{1+\alpha}\right)}\cdot\frac{\left(\frac{\alpha e^{\frac{\varepsilon}{rl}}}{1+\alpha e^{\frac{\varepsilon}{rl}}}\right)}{\left(\frac{1}{1+\alpha e^{\frac{\varepsilon}{rl}}}\right)}\right)^{rl} = e^{\varepsilon}.\end{aligned} \tag{24}$$

∎

APPENDIX F
PROOF OF THE ε-LDP OF THE UTILITY ENHANCING RANDOMIZATION STEP OF LATENT

Proof: Considering a sensitivity of rl, choose the randomization probabilities according to

$$p(B[i]v) = \begin{cases} \Pr[\boldsymbol{B}[v_1]=1|v_1] = \frac{\alpha}{1+\alpha}, & \text{if } i \in 2n;\ n \in \mathbb{N} \\ \Pr[\boldsymbol{B}[v_2]=0|v_1] = \frac{\alpha e^{\frac{\varepsilon}{rl}}}{1+\alpha e^{\frac{\varepsilon}{rl}}} & \text{,,} \\ \Pr[\boldsymbol{B}[v_1]=1|v_1] = \frac{1}{1+\alpha^3}, & \text{if } i \in 2n+1 \\ \Pr[\boldsymbol{B}[v_2]=0|v_1] = \frac{\alpha e^{\frac{\varepsilon}{rl}}}{1+\alpha e^{\frac{\varepsilon}{rl}}} & \text{,,} \end{cases} \tag{25}$$

$$\begin{aligned}\frac{\Pr[\boldsymbol{B}|v_1]}{\Pr[\boldsymbol{B}|v_2]} &= \frac{\prod_{i\in[d]}\Pr[\boldsymbol{B}[i]|v_1]}{\prod_{i\in[d]}\Pr[\boldsymbol{B}[i]|v_2]} \\ &= \frac{\prod_{i\in 2n}\Pr[\boldsymbol{B}[i]|v_1]}{\prod_{i\in 2n}\Pr[\boldsymbol{B}[i]|v_2]} \times \frac{\prod_{i\in 2n+1}\Pr[\boldsymbol{B}[i]|v_1]}{\prod_{i\in 2n+1}\Pr[\boldsymbol{B}[i]|v_2]} \\ &\le \left(\frac{\Pr[\boldsymbol{B}[v_1]=1|v_1]\Pr[\boldsymbol{B}[v_2]=0|v_1]}{\Pr[\boldsymbol{B}[v_1]=1|v_2]\Pr[\boldsymbol{B}[v_2]=0|v_2]}\right)^{\frac{rl}{2}}_{i\in 2n} \\ &\quad\times\left(\frac{\Pr[\boldsymbol{B}[v_1]=1|v_1]\Pr[\boldsymbol{B}[v_2]=0|v_1]}{\Pr[\boldsymbol{B}[v_1]=1|v_2]\Pr[\boldsymbol{B}[v_2]=0|v_2]}\right)^{\frac{rl}{2}}_{i\in 2n+1} \\ &= \left(\frac{\left(\frac{\alpha}{1+\alpha}\right)}{\left(\frac{1}{1+\alpha}\right)}\cdot\frac{\left(\frac{\alpha e^{\frac{\varepsilon}{rl}}}{1+\alpha e^{\frac{\varepsilon}{rl}}}\right)}{\left(\frac{1}{1+\alpha e^{\frac{\varepsilon}{rl}}}\right)}\right)^{\frac{rl}{2}}\left(\frac{\left(\frac{1}{1+\alpha^3}\right)}{\left(\frac{\alpha^3}{1+\alpha^3}\right)}\cdot\frac{\left(\frac{\alpha e^{\frac{\varepsilon}{rl}}}{1+\alpha e^{\frac{\varepsilon}{rl}}}\right)}{\left(\frac{1}{1+\alpha e^{\frac{\varepsilon}{rl}}}\right)}\right)^{\frac{rl}{2}} \\ &= e^{\varepsilon}.\end{aligned} \tag{26}$$

∎

REFERENCES

[1] M. Abadi *et al.*, "Deep learning with differential privacy," in *Proc. ACM SIGSAC Conf. Comput. Commun. Security (CCS)*, Vienna, Austria, 2016, pp. 308–318.

[2] R. Shokri, M. Stronati, C. Song, and V. Shmatikov, "Membership inference attacks against machine learning models," in *Proc. IEEE Symp. Security Privacy (SP)*, San Jose, CA, USA, 2017, pp. 3–18.

[3] M. Abadi *et al.*, "TensorFlow: A system for large-scale machine learning," in *Proc. OSDI*, vol. 16. Savannah, GA, USA, 2016, pp. 265–283.

[4] 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 Endow*, vol. 5, no. 8, pp. 716–727, 2012.

[5] C. Song, T. Ristenpart, and V. Shmatikov, "Machine learning models that remember too much," in *Proc. ACM SIGSAC Conf. Comput. Commun. Security*, Dallas, TX, USA, 2017, pp. 587–601.

[6] M. Fredrikson, S. Jha, and T. Ristenpart, "Model inversion attacks that exploit confidence information and basic countermeasures," in *Proc. 22nd ACM SIGSAC Conf. Comput. Commun. Security*, Denver, CO, USA, 2015, pp. 1322–1333.

[7] C. Dwork and A. Roth, "The algorithmic foundations of differential privacy," *Found. Trends® Theor. Comput. Sci.*, vol. 9, no. 3–4, pp. 211–407, 2014.

[8] R. Shokri and V. Shmatikov, "Privacy-preserving deep learning," in *Proc. 22nd ACM SIGSAC Conf. Comput. Commun. Security*, Denver, CO, USA, 2015, pp. 1310–1321.

[9] X. Xiao and Y. Tao, "Output perturbation with query relaxation," *Proc. VLDB Endow.*, vol. 1, no. 1, pp. 857–869, 2008.

[10] P. Kairouz, S. Oh, and P. Viswanath, "Extremal mechanisms for local differential privacy," in *Proc. Adv. Neural Inf. Process. Syst.*, Montreal, QC, Canada, 2014, pp. 2879–2887.

[11] J. A. Fox, *Randomized Response and Related Methods: Surveying Sensitive Data*, vol. 58. Los Angeles, CA, USA: SAGE, 2015,

[12] M. A. P. Chamikara, P. Bertok, D. Liu, S. Camtepe, and I. Khalil, "An efficient and scalable privacy preserving algorithm for big data and data streams," *Comput. Security*, vol. 87, Nov. 2019, Art. no. 101570.

[13] T.-H. H. Chan, M. Li, E. Shi, and W. Xu, "Differentially private continual monitoring of heavy hitters from distributed streams," in *Proc. Int. Symp. Privacy Enhanc. Technol. Symp.*, Vigo, Spain, 2012, pp. 140–159.

[14] Ú. Erlingsson, V. Pihur, and A. Korolova, "RAPPOR: Randomized aggregatable privacy-preserving ordinal response," in *Proc. ACM SIGSAC Conf. Comput. Commun. Security*, Scottsdale, AZ, USA, 2014, pp. 1054–1067.

[15] S. L. Warner, "Randomized response: A survey technique for eliminating evasive answer bias," *J. Amer. Stat. Assoc.*, vol. 60, no. 309, pp. 63–69, 1965.

[16] Y. Wang, X. Wu, and D. Hu, "Using randomized response for differential privacy preserving data collection," in *Proc. EDBT/ICDT Workshops*, vol. 1558, 2016, pp. 1–8.

[17] Z. Qin, Y. Yang, T. Yu, I. Khalil, X. Xiao, and K. Ren, "Heavy hitter estimation over set-valued data with local differential privacy," in *Proc. ACM SIGSAC Conf. Comput. Commun. Security (CCS)*, Vienna, Austria, 2016, pp. 192–203.

[18] M. Bun and T. Steinke, "Concentrated differential privacy: Simplifications, extensions, and lower bounds," in *Proc. Theory Cryptography Conf.*, 2016, pp. 635–658.

[19] T. Chanyaswad, A. Dytso, H. V. Poor, and P. Mittal, "MVG mechanism: Differential privacy under matrix-valued query," in *Proc. ACM SIGSAC Conf. Comput. Commun. Security*, Oct. 2018, pp. 230–246.

[20] J. Schmidhuber, "Deep learning in neural networks: An overview," *Neural Netw.*, vol. 61, pp. 85–117, Jan. 2015.

[21] Y. Kim, "Convolutional neural networks for sentence classification," in *Proc. Conf. Empir. Methods Nat. Lang. Process. (EMNLP)*, Doha, Qatar, 2014, pp. 1746–1751. [Online]. Available: https://doi.org/10.3115/v1/D14-1181

[22] A. Krizhevsky, I. Sutskever, and G. E. Hinton, "Imagenet classification with deep convolutional neural networks," in *Proc. Adv. Neural Inf. Process. Syst.*, 2012, pp. 1097–1105.

[23] L. M. Vaquero and L. Rodero-Merino, "Finding your way in the fog: Towards a comprehensive definition of fog computing," *ACM SIGCOMM Comput. Commun. Rev.*, vol. 44, no. 5, pp. 27–32, 2014.

[24] K. Kaur, S. Garg, G. Kaddoum, F. Gagnon, N. Kumar, and S. H. Ahmed, "An energydriven network function virtualization for multi-domain software defined networks," in *Proc. INFOCOM IEEE Conf. Comput. Commun. Workshops (INFOCOM WKSHPS)*, Paris, France, 2019, pp. 121–126, doi: 10.1109/INFCOMW.2019.8845314.

[25] F. Chollet *et al.*. (2015). *Keras: Deep Learning Library for Theano and Tensorflow*. [Online]. Available: https://keras.io/k

[26] T. Wang, J. Blocki, N. Li, and S. Jha, "Locally differentially private protocols for frequency estimation," in *Proc. 26th USENIX Security Symp. (USENIX Security)*, 2017, pp. 729–745.

[27] Y. Lecun, L. Bottou, Y. Bengio, and P. Haffner, "Gradient-based learning applied to document recognition," *Proc. IEEE*, vol. 86, no. 11, pp. 2278–2324, Nov. 1998.

[28] K. Yang, Q. Han, H. Li, K. Zheng, Z. Su, and X. Shen, "An efficient and fine-grained big data access control scheme with privacy-preserving policy," *IEEE Internet Things J.*, vol. 4, no. 2, pp. 563–571, Apr. 2017.

[29] D. Vatsalan, Z. Sehili, P. Christen, and E. Rahm, "Privacy-preserving record linkage for big data: Current approaches and research challenges," in *Handbook of Big Data Technologies*. Cham, Switzerland: Springer, 2017, pp. 851–895.

[30] K. Chen and L. Liu. (2005). *A Random Rotation Perturbation Approach to Privacy Preserving Data Classification*. [Online]. Available: https://corescholar.libraries.wright.edu/knoesis/916/

[31] K. Chen and L. Liu, "Geometric data perturbation for privacy preserving outsourced data mining," *Knowl. Inf. Syst.*, vol. 29, no. 3, pp. 657–695, 2011.

[32] F. Kerschbaum and M. Härterich, "Searchable encryption to reduce encryption degradation in adjustably encrypted databases," in *Proc. IFIP Annu. Conf. Data Appl. Security Privacy*, 2017, pp. 325–336.

[33] K. Gai, M. Qiu, H. Zhao, and J. Xiong, "Privacy-aware adaptive data encryption strategy of big data in cloud computing," in *Proc. IEEE 3rd Int. Conf. Cyber Security Cloud Comput. (CSCloud)*, Beijing, China, 2016, pp. 273–278.

[34] H. Xu, S. Guo, and K. Chen, "Building confidential and efficient query services in the cloud with RASP data perturbation," *IEEE Trans. Knowl. Data Eng.*, vol. 26, no. 2, pp. 322–335, Feb. 2014.

[35] M. A. P. Chamikara, P. Bertok, D. Liu, S. Camtepe, and I. Khalil, "Efficient data perturbation for privacy preserving and accurate data stream mining," *Pervasive Mobile Comput.*, vol. 48, pp. 1–9, Aug. 2018.

[36] A. Machanavajjhala and D. Kifer, "Designing statistical privacy for your data," *Commun. ACM*, vol. 58, no. 3, pp. 58–67, 2015.

[37] M. A. P. Chamikara, P. Bertok, D. Liu, S. Camtepe, and I. Khalil, "Efficient privacy preservation of big data for accurate data mining," *Inf. Sci.*, pp. 1–24, May 2019. [Online]. Available: https://doi.org/10.1016/j.ins.2019.05.053

[38] N. Li, T. Li, and S. Venkatasubramanian, "t-Closeness: Privacy beyond *k*-anonymity and *l*-diversity," in *Proc. IEEE 23rd Int. Conf. Data Eng. (ICDE)*, Istanbul, Turkey, 2007, pp. 106–115.

[39] L. Zhang, S. Jajodia, and A. Brodsky, "Information disclosure under realistic assumptions: Privacy versus optimality," in *Proc. 14th ACM Conf. Comput. Commun. Security*, Alexandria, VA, USA, 2007, pp. 573–583.

[40] S. R. Ganta, S. P. Kasiviswanathan, and A. Smith, "Composition attacks and auxiliary information in data privacy," in *Proc. 14th ACM SIGKDD Int. Conf. Knowl. Disc. Data Min.*, Las Vegas, NV, USA, 2008, pp. 265–273.

[41] R. C.-W. Wong, A. W.-C. Fu, K. Wang, P. S. Yu, and J. Pei, "Can the utility of anonymized data be used for privacy breaches?" *ACM Trans. Knowl. Disc. Data (TKDD)*, vol. 5, no. 3, pp. 1–24, 2011.

[42] C. Dwork, "The differential privacy frontier," in *Proc. Theory Cryptography Conf.*. 2009, pp. 496–502.

[43] N. Mohammed, R. Chen, B. C. M. Fung, and P. S. Yu, "Differentially private data release for data mining," in *Proc. 17th ACM SIGKDD Int. Conf. Knowl. Disc. Data Min.*, San Diego, CA, USA, 2011, pp. 493–501.

[44] N. Johnson, J. P. Near, and D. Song, "Towards practical differential privacy for sql queries," *Proc. VLDB Endow.*, vol. 11, no. 5, pp. 526–539, 2018.

[45] F. D. McSherry, "Privacy integrated queries: An extensible platform for privacy-preserving data analysis," in *Proc. ACM SIGMOD Int. Conf. Manag. Data*, Providence, RI, USA, 2009, pp. 19–30.

[46] R. Bassily and A. Smith, "Local, private, efficient protocols for succinct histograms," in *Proc. 47th Annu. ACM Symp. Theory Comput.*, Portland, OR, USA, 2015, pp. 127–135.

[47] P. Li *et al.*, "Multi-key privacy-preserving deep learning in cloud computing," *Future Gener. Comput. Syst.*, vol. 74, pp. 76–85, Sep. 2017.

[48] N. Papernot, M. Abadi, Ú. Erlingsson, I. J. Goodfellow, and K. Talwar, "Semi-supervised knowledge transfer for deep learning from private training data," in *Proc. 5th Int. Conf. Learn. Represent. (ICLR)*, Toulon, France, Apr. 2017. [Online]. Available: https://openreview.net/forum?id=HkwoSDPgg

[49] S. A. Osia *et al.*, "A hybrid deep learning architecture for privacy-preserving mobile analytics," *arXiv preprint arXiv:1703.02952*, 2017.

[50] F. Pedregosa *et al.*, "Scikit-learn: Machine learning in python," *J. Mach. Learn. Res.*, vol. 12, pp. 2825–2830, Oct. 2011.

[51] T. Chen and C. Guestrin, "XGBoost: A scalable tree boosting system," in *Proc. 22nd ACM SIGKDD Int. Conf. knowl. Disc. Data Min.*, San Francisco, CA, USA, 2016, pp. 785–794.

Pathum Chamikara Mahawaga Arachchige received the M.Phil. degree in computer science from the University of Peradeniya, Kandy, Sri Lanka, in 2015.

He is a Ph.D. Researcher of computer science and software engineering with the School of Science, RMIT University, Melbourne, VIC, Australia. He is also a Researcher with CSIRO Data61, Melbourne. His research interests include information privacy and security, data mining, artificial neural networks, and fuzzy logic.

Peter Bertok received the Ph.D. degree in computer engineering from the University of Tokyo, Tokyo, Japan.

He is an Associate Professor with the School of Science, RMIT University, Melbourne, VIC, Australia, where he is a member of the Cyberspace and Security Group. His research interests include access control, privacy protection, and communication security.

Ibrahim Khalil received the Ph.D. degree from the University of Bern, Bern, Switzerland, in 2003.

He was also worked for EPFL, Lausanne, Switzerland, the University of Bern, and Osaka University in Japan. He is an Associate Professor with the School of Science, RMIT University, Melbourne, VIC, Australia. He has several years of experience in Silicon Valley-based companies working on Large Network Provisioning and Management Software. His research interests are in scalable efficient computing in distributed systems, network and data security, and secure data analysis, including big data security, steganography of wireless body sensor networks and highspeed sensor streams, and smart grids.

Dongxi Liu received the Ph.D. degree in computer science and engineering from Shanghai Jiao Tong University, Shanghai, China.

He was a Researcher with the University of Tokyo, Tokyo, Japan, from February 2004 to March 2008, and a Research Fellow with the National University of Singapore, Singapore, from December 2002 to December 2003. In March 2008, he joined CSIRO Data61, Melbourne, VIC, Australia, where he is a Senior Research Scientist. His current research focuses on lightweight encryption for IoT security and encrypted data processing for cloud security.

Seyit Camtepe received the Ph.D. degree in computer science from Rensselaer Polytechnic Institute, Troy, NY, USA, in 2007.

He is a Principal Research Scientist with CSIRO Data61, Melbourne, VIC, Australia. From 2007 to 2013, he was with Technische Universitaet Berlin, Berlin, Germany, as a Senior Researcher and a Research Group Leader of security. From 2013 to 2017, he worked as a Lecturer with the Queensland University of Technology, Brisbane, QLD, Australia. His research interests include mobile and wireless communication, pervasive security and privacy, and applied and malicious cryptography.

Mohammed Atiquzzaman (SM'94) received the M.S. and Ph.D. degrees in electrical engineering and electronics from the University of Manchester, Manchester, U.K.

He currently holds the Edith Kinney Gaylord Presidential Professorship with the School of Computer Science, University of Oklahoma, Norman, OK, USA. His research has been funded by the National Science Foundation, National Aeronautics and Space Administration, U.S. Air Force, Cisco, and Honeywell. His research interests are in communications switching, transport protocols, wireless and mobile networks, satellite networks, and optical communications.

Dr. Atiquzzaman is the Editor-in-Chief of the *Journal of Networks and Computer Applications*, the founding Editor-in-Chief of *Vehicular Communications* and has served/serving on the editorial boards of various IEEE journals and co-chaired numerous IEEE international conferences, including IEEE Globecom.