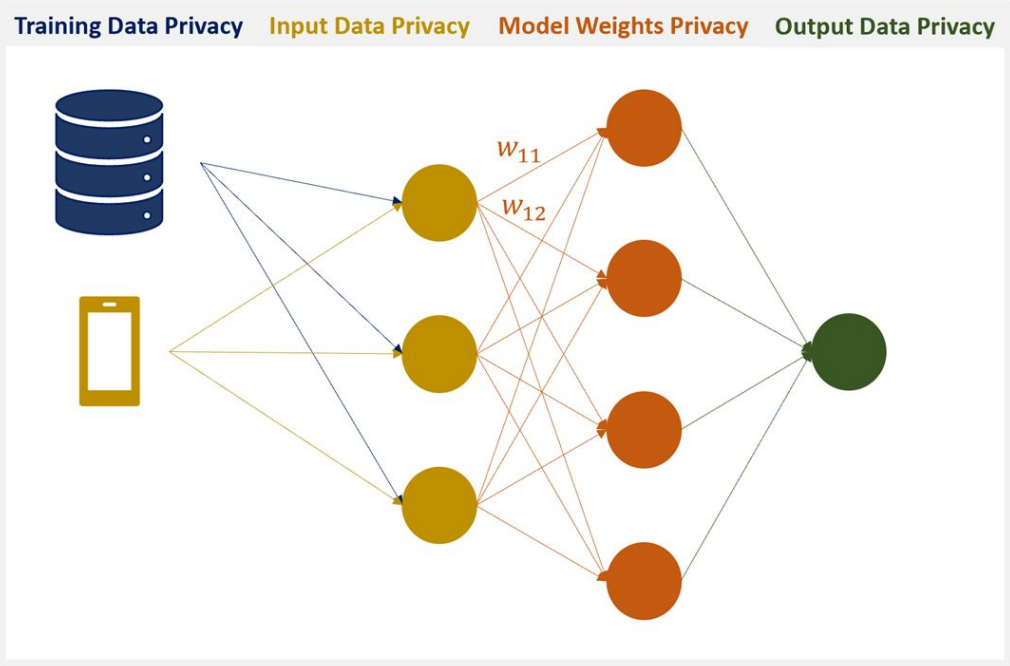
Concrete ML

Privacy Preserving Machine Learning Using
Fully Homomorphic Encryption Techniques

Privacy Goals

Privacy Goals Overview



Identity Privacy

Protecting the data contributor’s identity is necessary in many data collection processes, especially when the data collected may reveal political beliefs, sexual preferences, health conditions etc.

Achieving identity privacy can be seen as ensuring anonymity of the data contributors both from the model creators and the model owners, who are highly involved in the data collection stage of an ML process, but also from adversaries which in many cases are acting as model consumers.

Raw Dataset Privacy

Often, the raw dataset Z contains sensitive information, such as the X-ray images from the cancer diagnosis example, and if compromised the patient's privacy can be violated.

Often, compromising the privacy of the raw dataset could also result in compromising identity privacy, since data in Z can be used to identify a specific data contributor.

A common malpractice which eases the task of compromising Z is storing the dataset as plaintext instead of using state of the art encryption.

Feature Dataset Privacy

The privacy of the feature datasets is as important as the privacy of the raw dataset. This is because both the training set and the validation set are extracted from raw dataset, and in case of them being compromised, information about raw dataset could also be recovered.

Unfortunately, the privacy of the feature datasets is threatened in many ML models, which by default store the set training set (or the labelled set, in the case of supervised learning models) inside the model.

Model Privacy

Privacy of the model refers to the secrecy of the produced model and its parameter. Keeping the model secret is a common goal desired by the model owners, who want to stay ahead of their competitors.

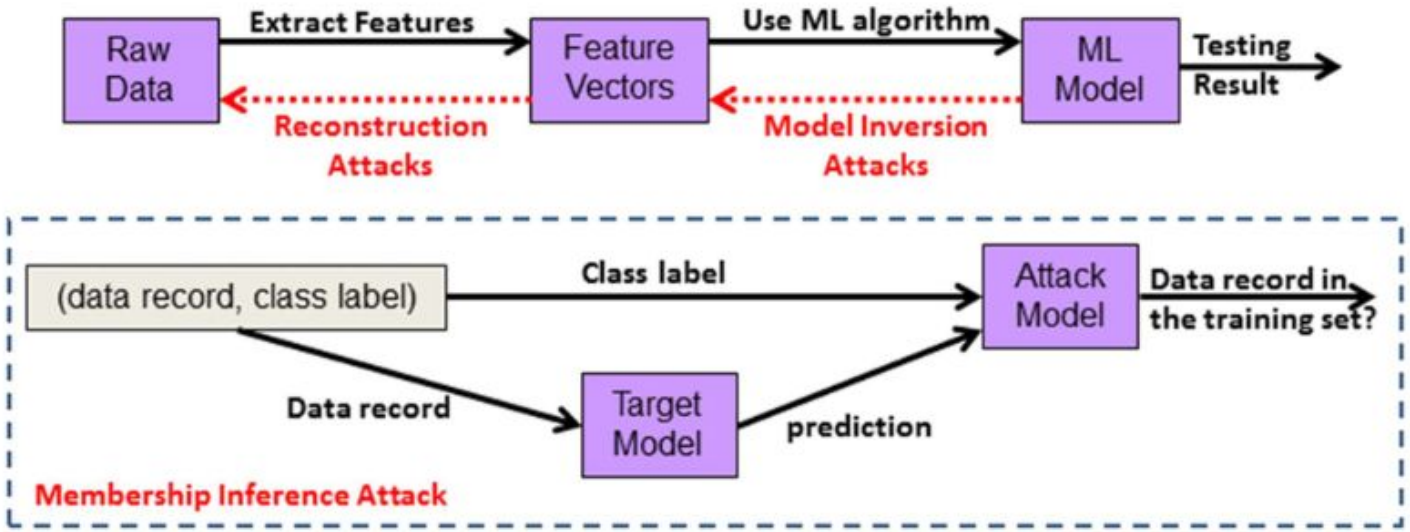
However, exposing part of the model's functionality to the model consumers is inevitable, since observing the responses to queries may reveal information about the internal structure of the model.

Input Privacy

Often the input (or part of it) to the model may be sensitive and needs to be protected. Therefore, only the output of the model is revealed to the model consumers.

For instance, consider the scenario of an ML model that takes as input medical records to predict if a certain disease can be inherited. In such cases, it is likely that the medical record contributors do not desire to disclose their medical records to each other.

Machine Learning Attack Vectors



Attack Vectors

Membership Inference Attacks

In this type of attack, the adversary wants to determine if a given data point z was part of the raw dataset Z, or if a particular data point x was part of the training set $X1$.

Membership inference attacks usually exploit that ML models often behave differently on the data used for their training versus unseen data.Membership inference attacks can compromise the raw dataset privacy or the feature datasets privacy.

For instance, this attack could be used by an adversary to learn whether a specific individual's record was used to train an ML model which determines the presence of a certain disease.

This attack can be performed by a black box adversary.

De-anonymization Attacks

This attack aims to identify an individual who has contributed his data into a dataset. ‘

Many model owners publish the raw dataset Z or the feature datasets $X1$ and $X2$, which may allow an adversary to compromise the identity privacy of the data contributors, even though the dataset has been first anonymised.

This attack requires white box access to the dataset. However, it can also be performed using black-box access by chaining other attacks

Reconstruction Attacks

The goal of this attack is to construct the raw dataset Z by reverse engineering the feature training dataset $X1$ or the validation dataset $X2$.

This attack can compromise the raw dataset privacy. In general, it requires white box access to a model that hard-codes the feature datasets inside it.

Model Extraction Attacks

The goal of this attack is to construct an approximation $\mathcal{M}\theta$ of the model $\mathcal{M}\theta$. The more accurate the approximation is, the more successful the attack is.

In this type of attack, the adversary has black box capabilities and trains his model $\mathcal{M}\theta$ by collecting pairs of queries and their responses received by the model $\mathcal{M}\theta$. This attack violates the model privacy.

Model Inversion Attacks

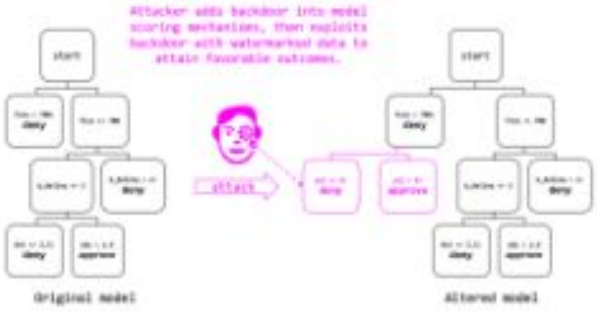
Model inversion attacks aim at inferring properties about the training dataset $X1$, even when it is not explicitly stored in the model $\mathcal{M}\theta$.

This type of attack can even be performed by a limited black box attacker who can interact with the model by querying it and collecting its responses.

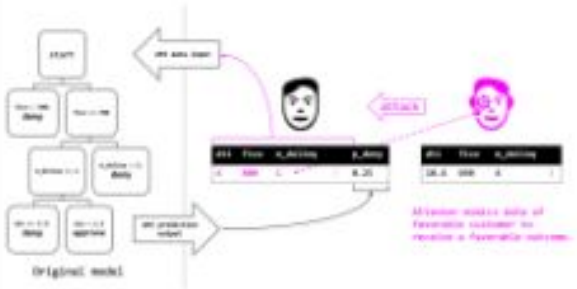
It is also common that an adversary may first perform a model extraction attack and then utilise the model $\mathcal{M}\theta$ to accomplish a model inversion attack.

Machine Learning Attack Cheatsheet

Backdoors and Watermarks



Impersonation



Adversarial Examples

Data Poisoning

Model Inversion and Stealing

Membership Inference

Primitive Methods

Limitations of Data Anonymization

- Goal is to remove personally identifiable information before publishing data
- First approach is to strip attributes that uniquely identify an individual
  - Susceptible to linkage attacks - uniquely linking a record in the anonymized dataset to an identified record in the public dataset
- Second approach is k-anonymity where a set of attributes are defined as quasi identifiers. Suppress / generalize attributes and / or add dummy records to make every record in the dataset indistinguishable from at least k-1 other records with respect to the quasi identifiers.
  - Variants of k-anonymity (t-closeness, l-diversity) tried to address the issues but require to modify the original data even more which destroys the utility.

De-anonymization is still a possibility!

In high-dimensional and sparse datasets, any combination of attributes is a potential PII that can be exploited using appropriate auxiliary knowledge

- De-anonymization of Netflix dataset protected with k-anonymity using a few public ratings from IMDB [Narayanan and Shmatikov, 2008]
- De-anonymization of Twitter graph using Flickr [Narayanan and Shmatikov, 2009]
- 4 spatio-temporal points uniquely identify most people [de Montjoye et al., 2013]

Aggregate Statistics is not safe

- Problem 1: differencing attacks, i.e. combining aggregate queries to obtain precise information about specific individuals
- Problem 2: membership inference attacks, i.e. inferring presence of known individual in a dataset from (high-dimensional) aggregate statistics
- Problem 3: reconstruction attacks, i.e. inferring the dataset from the output of many aggregate queries

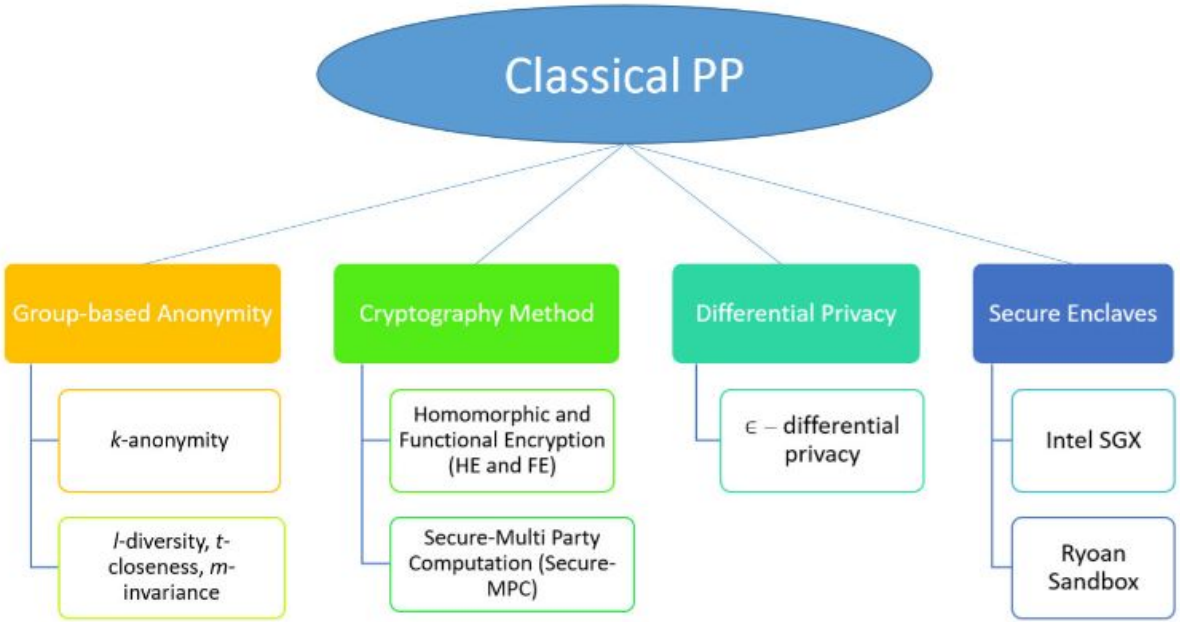
Machine learning models can be considered as elaborate kind of aggregate statistics!

Models are susceptible to membership inference attacks where presence of known individuals in the training set can be extracted!

Models are also susceptible to reconstruction attacks where the sensitive texts are extracted from large language models

Machine learning models are also susceptible to differencing attacks

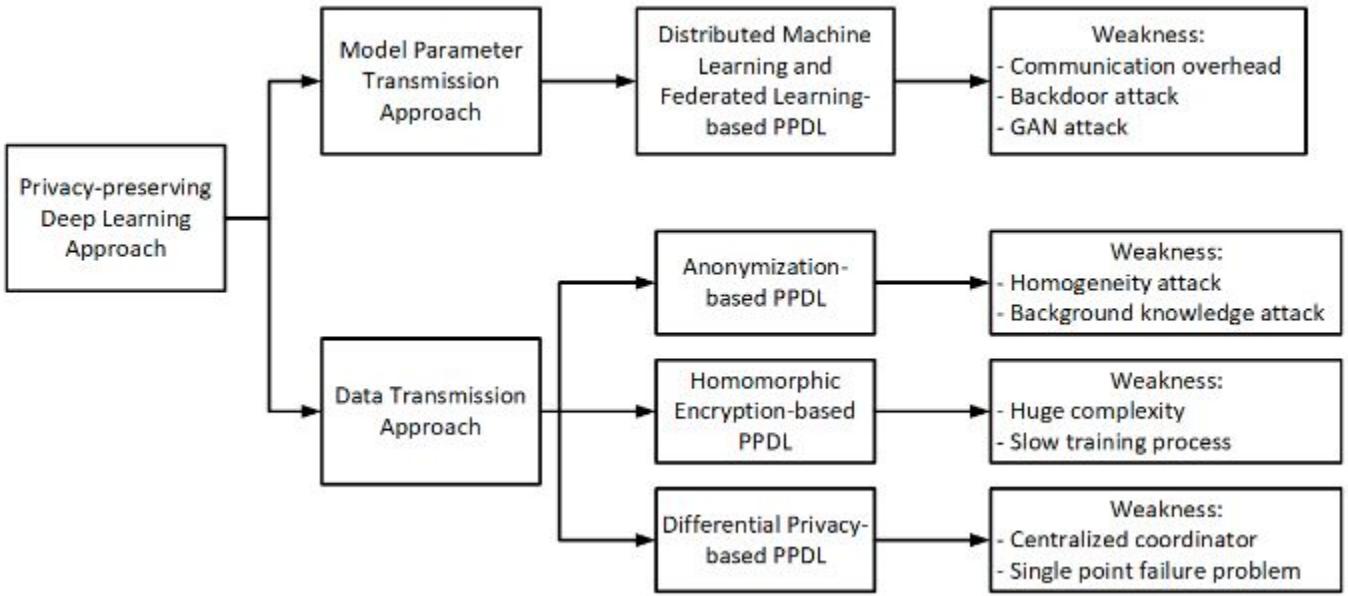
PPML Methods



PPML Approaches

- Noise perturbation techniques - An additive or multiplicative noise is included in the data aiming at hiding the original value
- Differential Privacy - A probabilistic approach that aims at maximizing the information provided by queries from a database while minimizing the chance of identifying the individual records. Relies on sanitization techniques.
  - Input / output perturbation - add noise to inputs or computation results
  - K-anonymization - Verify that at least k elements provide the same attributes
- Federated Machine Learning - Updating a master model with incremental updates from every client owning a subset of the training set
- Distributed Machine Learning - The exchanged information is not raw data and does not reveal any information concerning individual users
- Partially Homomorphic Cryptography - Homomorphic encryption schemes guarantee some equivalence between operations performed on the clear and encrypted domain
- Fully Homomorphic Cryptography - Schemes that guarantee the equivalence between any operation performed on the clear and encrypted domain
- Secure Multiparty Computation - Every user only gets as information the output of a function on its own data

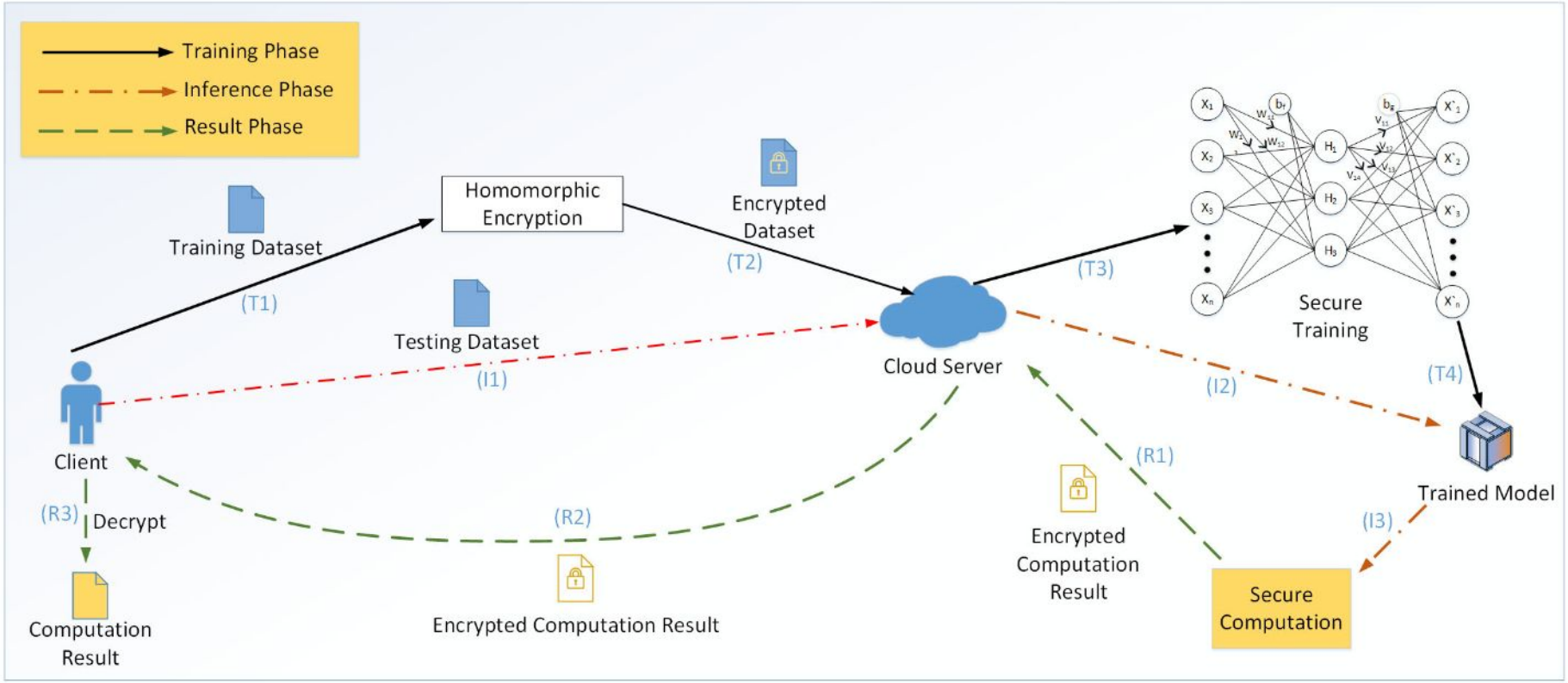
Privacy Preserving Deep Learning



Homomorphic Encryption based PPDL

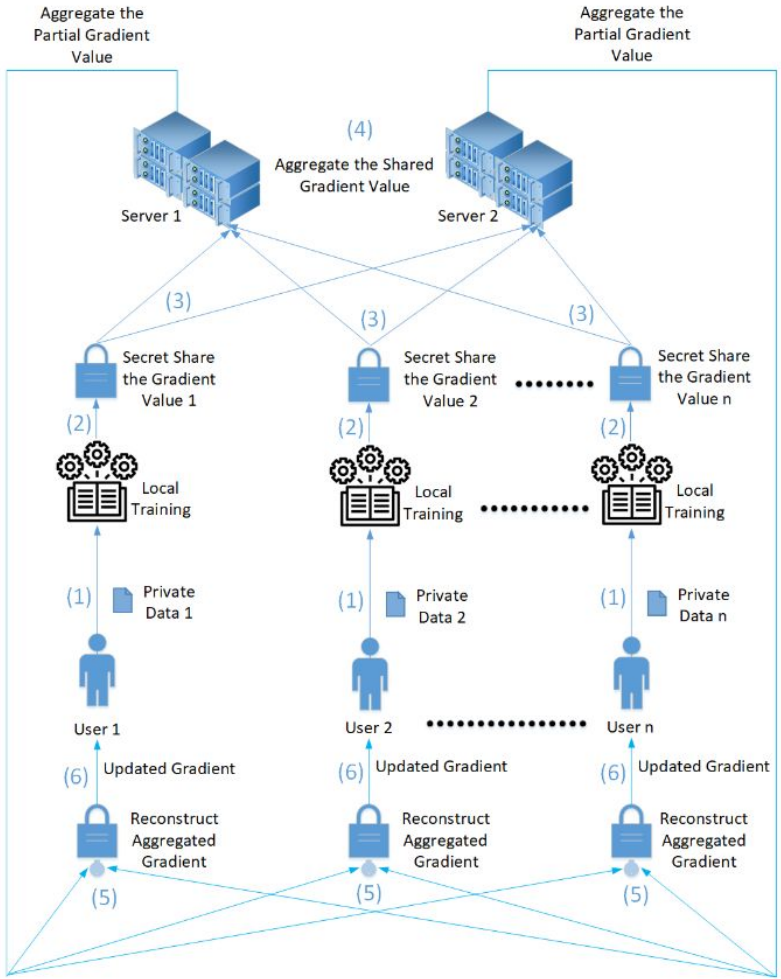
Fully homomorphic encryption enables the computation on encrypted data, with operations such as addition and multiplication that can be used as basis for more complex arbitrary functions.

Due to the high cost associated with frequently bootstrapping the cipher text (refreshing the cipher text because of the accumulated noise), additive homomorphic encryption schemes are mostly used in PPML approaches.

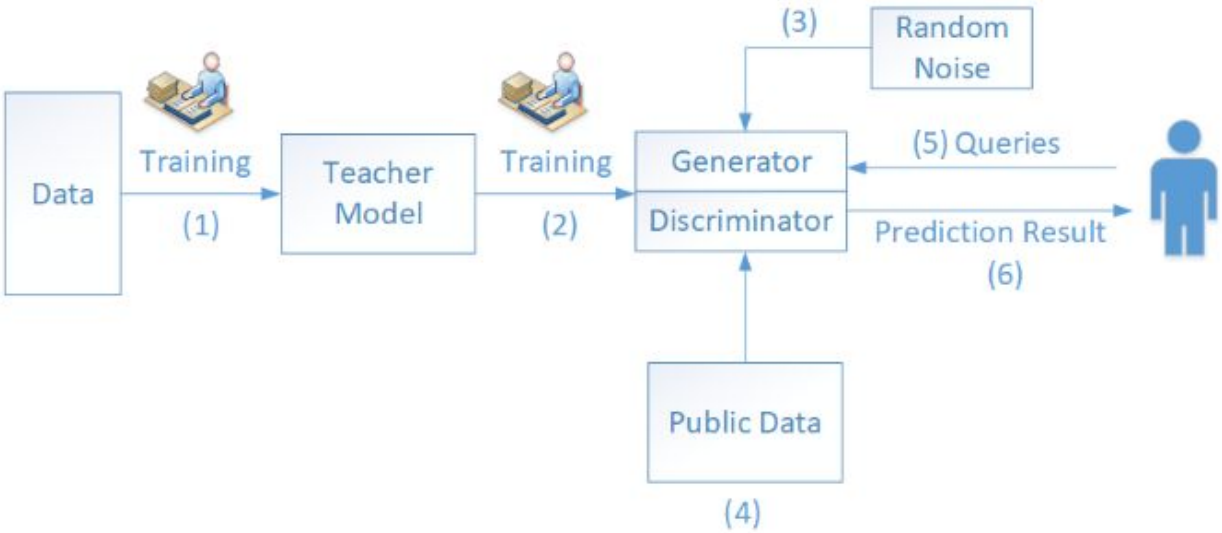

Training Phase
Inference Phase
Result Phase
Training Dataset
(T1)
Homomorphic Encryption
Encrypted Dataset
(T2)
(T3)
Secure Training
(T4)
Testing Dataset
(I1)
Cloud Server
(I2)
Client
(R3)
Decrypt
Computation Result
(R2)
Encrypted Computation Result
(R1)
Encrypted Computation Result
Secure Computation
(I3)
Trained Model

Secure Multiparty Computation based PPDL

- Firstly, users perform local training using their private data.
- Then, the gradient result from the training process is secret-shared.
- The shared gradient is transmitted to each server.
- After that, the server aggregates the shared gradient value from users.
- The aggregated gradient value is transmitted from each server to the client
- Each client reconstructs the aggregated gradient value for the next training process.



Differential Privacy based PPDL



Key steps in the Differential Privacy PPDL

First the training data are used to train the teacher model

Then the teacher model is used to train the student model

Student model is devised as a GAN with generator and discriminator

Random noise is added to the generator to generate fake data

On the other hand, the teacher model trains the student model using the public data

The student model runs the zero sum game between the generator and discriminator

Comparison of PPDL Approaches

Method	Main Limitation	How to Overcome
HE-based PPDL	More layer leads to more complexity because of the property of HE.	Adding batch normalization layer between pooling layer and activation layer.
	Accuracy highly dependents on the approximation of the activation function.	Using the polynomial with lowest possible degree as the activation function.
SecureMPC-based PPDL	Most of the current publications only guarantee privacy of clients based on honest-but-curious adversary, but they do not have protection against malicious adversary.	Designing a protocol that each participant shares their secret data, then the server runs secure computation to generate the output. By doing this, even if the server is malicious, it cannot see either the input or output value.
Differential Privacy-based PPDL	Differential privacy is computationally very expensive because it requires many interactions between server and client when adding the noise.	Distributed differential privacy can be a good solution to reduce the interactions. Since the architecture of previous differential privacy-based PPDL techniques are centralized, distributed differential privacy will be an interesting option.
Secure Enclaves-based PPDL	There is a data leakage issue due to side channel attack.	Utilizing differential privacy so that the adversary cannot get the real data, even if the side channel attack is successful.

Concrete ML

Concrete ML Approach

Automatically turn models into their FHE equivalent

Train models on encrypted data

Pre-process encrypted data through data frames

Concrete ML Capabilities

Training on encrypted data provides the highest level of privacy but is slower than training on clear data.

Federated learning is an alternative approach, where data privacy can be ensured by using a trusted gradient aggregator, coupled with optional differential privacy instead of encryption.

Concrete ML can import linear models, including logistic regression, that are trained using federated learning using the from_sklearn function.

Lifecycle of a Concrete ML Model

- Concrete ML models can be trained on clear or encrypted data, then deployed to predict on encrypted inputs.
- During deployment data pre-processing can be done on encrypted data.
- Data can be encrypted during the entire lifecycle of the machine learning model

Concrete ML Workflow

- The model is trained on unencrypted (plaintext) data using scikit-learn. As FHE operates over integers, Concrete ML quantizes the model to use only integers during inference.
- The quantized model is compiled to an FHE equivalent. Under the hood, the model is first converted to a Concrete Python program, then compiled. Inference can then be done on encrypted data.
- Encrypted inference can be implemented in the model-development phase.
- Alternatively, during deployment in a client/server setting, the data is encrypted by the client, processed securely by the server, and then decrypted by the client.

Concrete ML Model Development

- Training: A model is trained either using plaintext, non-encrypted, training data, or encrypted training data.
- Quantization: The model is converted into an integer equivalent using quantization. Concrete ML performs this step either during training (Quantization Aware Training) or after training (Post-training Quantization), depending on model type.
- Quantization converts inputs, model weights, and all intermediate values of the inference computation to integers.

Concrete ML Model Simulation

- Testing FHE models on very large data-sets can take a long time. Furthermore, not all models are compatible with FHE constraints out of the box.
- Simulation allows you to execute a model that was quantized, to measure the accuracy it would have in FHE, but also to determine the modifications required to make it FHE compatible.

Concrete ML Model Compilation

- Once the model is quantized, simulation can confirm it has good accuracy in FHE.
- The model then needs to be compiled using Concrete's FHE Compiler to produce an equivalent FHE circuit.
- This circuit is represented as an MLIR program consisting of low level cryptographic operations.

Concrete ML Model Compilation Process

- Tracing the NumPy program and creating a Concrete op-graph
- Checking the op-graph for FHE compatibility
- Producing machine code for the op-graph
- This step automatically determines cryptographic parameters

Concrete ML Inference

- The compiled model can then be executed on encrypted data, once the proper keys have been generated.
- The model can also be deployed to a server and used to run private inference on encrypted inputs.

Current Limitations

- To make a model work with FHE, the only constraint is to make it run within the supported precision limitations of Concrete ML (currently 16-bit integers).
- Thus, machine learning models must be quantized, which sometimes leads to a loss of accuracy versus the original model, which operates on plaintext.
- Additionally, Concrete ML currently only supports training on encrypted data for some models, while it supports *inference* for a large variety of models.
- Finally, there is currently no support for pre-processing model inputs and post-processing model outputs.

Concepts

Variants of Homomorphic Encryption

Homomorphism means elements of one set are transformed to elements of a second set while maintaining the relationships between the elements of the two sets.

Crypto systems enabling to add or to multiply ciphertexts (but not both operations) are called partially homomorphic encryption schemes; examples include the ElGamal cryptosystem (1985) or the Paillier cryptosystem (1999).

We say of a scheme that it is “fully” homomorphic when it supports both addition and multiplication of ciphertexts, as any program can be represented as a circuit of additions and multiplications.

Types of Homomorphic Encryption

- Fully Homomorphic Encryption (FHE)
  - Arbitrary functions with any operations
  - But computationally expensive
- Somewhat Homomorphic Encryption (SWHE)
  - Can evaluate specific functions with limited processing (+ and *)
  - Could be more computationally expensive
- Partially Homomorphic Encryption (PHE)
  - Can evaluate functions with very limited operations (+ or *)
  - Computationally cheap

Primitives for Homomorphism

Security of the these schemes are based on a hard lattice problem called learning with errors. Accordingly, the resulting ciphertexts must contain a certain level of noise to guarantee the security of the encryption.

This issue however is that computing homomorphically will increase the noise level in the ciphertext. As long as the noise is below a certain threshold, the ciphertext can be decrypted. However, if the noise grows too much, it will overflow on the data itself, rendering decryption impossible.

Bootstrapping and Homomorphism

To prevent the issues from noise happening, a special noise-reduction operation called bootstrapping can be applied to the ciphertext, effectively resetting the noise to a nominal level.

In Gentry's original FHE scheme, bootstrapping is done by homomorphically evaluating the decryption circuit, resulting in another ciphertext that encrypts the same plaintext. Since decryption removes noise, the noise present in a bootstrapped ciphertext is reset to a nominal level (i.e., it only contains the noise coming from the bootstrapping).

A possible basic strategy to get a fully homomorphic encryption scheme is therefore to perform a bootstrap operation after each addition or multiplication on ciphertexts using a somewhat homomorphic encryption (SHE).