DEFENSE STRATEGIES

These adversarial examples are hard to defend because of the following reasons:

(1) **A theoretical model of the adversarial example crafting process is very difficult to construct.**

-complex optimization process which is non-linear and non-convex for most Machine Learning models.

-The lacking of proper theoretical tools to describe the solution to these complex optimization problems make it even harder to make any theoretical argument that a particular defense will rule out a set of adversarial examples.

(2) **Machine Learning models are required to provide proper outputs for every possible input.**

- A considerable modification of the model to incorporate robustness against the adversarial examples may change the elementary objective of the model.

Ref

1. Adversarial Attacks and Defences: A Survey, https://arxiv.org/pdf/1810.00069.pdf.

2. Ian J. Goodfellow, Jonathon Shlens, and Christian Szegedy. 2014. Explaining and Harnessing Adversarial Examples. CoRR abs/1412.6572 (2014). http://arxiv.org/abs/1412.6572

Challenges

1. Most of the current defense strategies are not adaptive to all types of adversarial attack as one

method may block one kind of attack but leaves another vulnerability open to an attacker

1. implementation of such defense strategies

may incur performance overhead, and can also degrade the prediction accuracy of the actual model.

Solution 1: Adversarial Training

[The primary objective of the adversarial training is to increase model robustness by injecting adversarial examples into the training set]

Adversarial training is a standard brute

force approach where the defender simply generates a lot of adversarial examples and augments

these perturbed data while training the targeted model.

The augmentation can be done either by feeding the model with both the original data and the crafted data, or by learning with a modified objective function given by [2]

$$\widetilde{J}(\theta, x, y) = \alpha J(\theta, x, y) + (1 - \alpha) J(\theta, x + \epsilon sign(\nabla_x J(\theta, x, y)), y)$$

Equation Details

$$\widetilde{J}(\theta, x, y) = \alpha J(\theta, x, y) + (1 - \alpha) J(\theta, x + \epsilon sign(\nabla_x J(\theta, x, y)), y)$$

Let $\boldsymbol{\theta}$ be the parameters of a model, $\boldsymbol{x}$ the input to the model, y the targets associated with $\boldsymbol{x}$ (for machine learning tasks that have targets) and $J(\boldsymbol{\theta}, \boldsymbol{x}, y)$ be the cost used to train the neural network. We can linearize the cost function around the current value of $\boldsymbol{\theta}$, obtaining an optimal max-norm constrained pertubation of

$$\boldsymbol{\eta} = \epsilon \operatorname{sign}\left(\nabla_{\boldsymbol{x}} J(\boldsymbol{\theta}, \boldsymbol{x}, y)\right).$$

- J : the original loss function.

The central idea behind this strategy is to increase model's robustness by ensuring that it will predict the same class for legitimate as well as perturbed examples in the same direction.

LINEAR EXPLANATION OF ADVERSARIAL EXAMPLES [2]

In many problems, the precision of an individual input feature is limited. For example, digital images often use only 8 bits per pixel so they discard all information below $1/255$ of the dynamic range. Because the precision of the features is limited, it is not rational for the classifier to respond differently to an input $\boldsymbol{x}$ than to an adversarial input $\tilde{\boldsymbol{x}} = \boldsymbol{x} + \boldsymbol{\eta}$ if every element of the perturbation $\boldsymbol{\eta}$ is smaller than the precision of the features. Formally, for problems with well-separated classes, we expect the classifier to assign the same class to $\boldsymbol{x}$ and $\tilde{\boldsymbol{x}}$ so long as $||\boldsymbol{\eta}||_\infty < \epsilon$, where ϵ is small enough to be discarded by the sensor or data storage apparatus associated with our problem.

Consider the dot product between a weight vector $\boldsymbol{w}$ and an adversarial example $\tilde{\boldsymbol{x}}$:

$$\boldsymbol{w}^\top \tilde{\boldsymbol{x}} = \boldsymbol{w}^\top \boldsymbol{x} + \boldsymbol{w}^\top \boldsymbol{\eta}.$$

LINEAR EXPLANATION OF ADVERSARIAL EXAMPLES [2]

- for problems with well-separated classes, we expect the classifier to assign the same class to x and x˜ so long as $||\eta||_\infty < , \varepsilon$.
- We can maximize this increase subject to the max norm constraint on η by assigning $\eta = \text{sign}(w)$.

- If w has n dimensions and the average magnitude of an element of the weight vector is m, then the activation will grow by εmn.

- Since $||\eta||_\infty$ does not grow with the dimensionality of the problem but the change in activation caused by perturbation by η can grow linearly with n, then for high dimensional problems, we can make many infinitesimal changes to the input that add up to one large change to the output

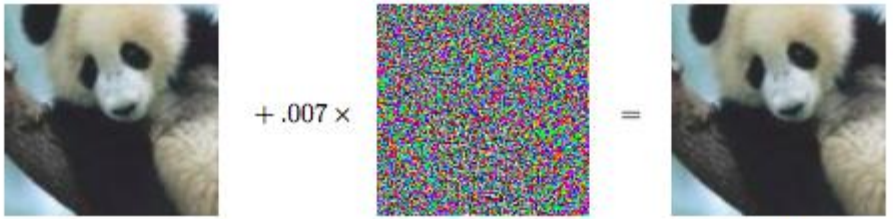


Figure 1: A demonstration of fast adversarial example generation applied to GoogLeNet (Szegedy et al., 2014a) on ImageNet. By adding an imperceptibly small vector whose elements are equal to the sign of the elements of the gradient of the cost function with respect to the input, we can change GoogLeNet's classification of the image. Here our ϵ of .007 corresponds to the magnitude of the smallest bit of an 8 bit image encoding after GoogLeNet's conversion to real numbers.

Let $\boldsymbol{\theta}$ be the parameters of a model, $\boldsymbol{x}$ the input to the model, y the targets associated with $\boldsymbol{x}$ (for machine learning tasks that have targets) and $J(\boldsymbol{\theta}, \boldsymbol{x}, y)$ be the cost used to train the neural network. We can linearize the cost function around the current value of $\boldsymbol{\theta}$, obtaining an optimal max-norm constrained pertubation of

$$\boldsymbol{\eta} = \epsilon\text{sign}\left(\nabla_{\boldsymbol{x}} J(\boldsymbol{\theta}, \boldsymbol{x}, y)\right).$$

Equation Details

$$\widetilde{J}(\theta, x, y) = \alpha J(\theta, x, y) + (1 - \alpha) J(\theta, x + \epsilon sign(\nabla_x J(\theta, x, y)), y)$$

J : the original loss function.

Modified the original cost function with adversarial inputs in training.

Additional instances are crafted using one or multiple following attack strategies

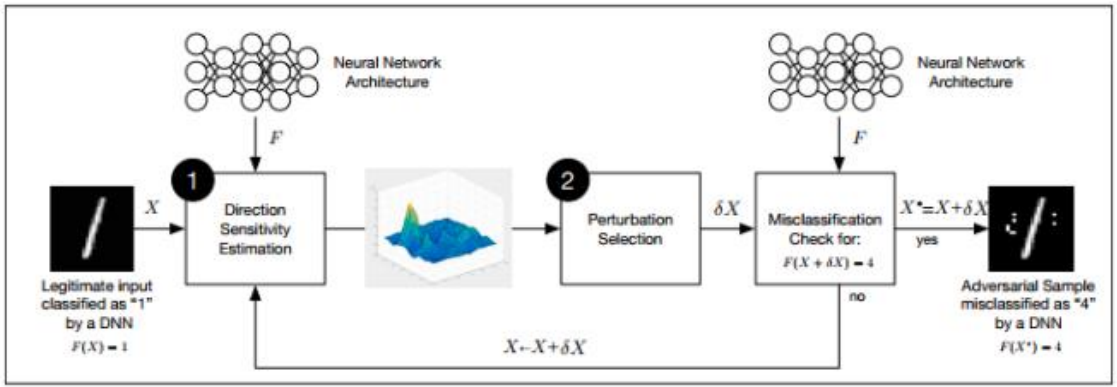
- How to modify samples so that a classification model yields an adversarial output?

Testing Phase Generation

How adversaries craft adversarial samples in a white-box setup.

Papernot et al. [62] introduced a general framework :

The framework is split into two phases: a) direction sensitivity estimation and b) perturbation selection



Direction Sensitivity Estimation: The adversary evaluates the sensitivity of a class change to each input feature by identifying directions in the data manifold around sample X in which the model F , learned by the DNN is most sensitive and likely to result in a class change.

Perturbation Selection: The adversary then exploits the knowledge of sensitive information to select a perturbation δX among the input dimensions in order to obtain an adversarial perturbation which is most efficient.

Both the steps are repeated by replacing X with X +δX before the start of each new iteration,until the adversarial goal is satisfied by the perturbed sample.

Total perturbation used for crafting the adversarial sample from a valid example needs to be as minimum as possible. This is necessary for the adversarial samples to remain undetected in human eyes.

Adversarial sample crafting using large perturbations is trivial. Thus, if one defines a norm $\|.\|$ to appropriately describe the differences between points in the input domain of the DNN model F, we can formalize the adversarial samples as a solution to the following optimization problem:

$$X_* = X + \arg\min_{\delta X} \{\|\delta X\| : F(X + \delta X) \neq F(X)\}$$

How to find an approximate solution to this optimization problem using each of the two steps?

Direction Sensitivity Estimation.

In this step, the adversary considers a legitimate sample X,an n-dimensional input vector.

The objective here is to find those dimensions of X which will produce an expected adversarial performance with the smallest selected perturbation.

This can be achieved by changing the input components of X and evaluating the sensitivity of the trained DNN model F for these changes.

Different Methods for Sensitivity Analysis

L-BFGS: Szegedy et al. [70] first introduced the term adversarial sample by formalizing the following minimization problem as the search for adversarial examples.

$$\arg\min_{r} f(x+r) = l \quad \text{s.t.} \ (x+r) \in D$$

- The input example x, which is correctly classified by f, is perturbed with r to obtain the resulting adversarial example $x* = x + r$.
- The perturbed sample remains in the input domain D, however, it is assigned the target label l.
- good performance, it is computationally expensive while calculating adversarial samples.

Why the name L-BFGS?

Broyden–Fletcher–Goldfarb–Shanno (BFGS) algorithm is an iterative method for solving unconstrained nonlinear optimization problems.

L indicates: Limited-memory BFGS

Fast Gradient Sign Method (FGSM):

fast gradient sign methodology which calculates the gradient of the cost function with respect to the input of the neural network.

adversarial examples are generated using the following equation:

$$X_* = X + \epsilon * sign(\nabla_x J(X, y_{true}))$$

Here, J is the cost function of the trained model, ∇_x denotes the gradient of the model with respect to a normal sample X with correct label y_{true}, and ϵ denotes the input variation

Target Class Method:

maximizes the probability of some specific target classt , which is unlikely the true class for a given example.

The adversarial example is crafted using the following equation:

$$X_* = X - \epsilon * sign(\nabla_x J(X, y_{target}))$$

Basic Iterative Method:

This method generates adversarial samples iteratively using small step size.

$$X_*^0 = X; \quad X_*^{n+1} = Clip_{X,e}\{X_*^n + \alpha * sign(\nabla_x J(X_*^n, y_{true}))\}$$

Here, α is the step size and $Clip_{X,e}\{A\}$ denotes the element-wise clipping of X. The range of $A_{i,j}$ after clipping belongs in the interval $[X_{i,j} - \epsilon, X_{i,j} + \epsilon]$. This method does not typically rely on any approximation of the model and produces additional harmful adversarial examples when run for more iterations.

Jacobian Based Method

Perturbation Selection

An adversary may use the information about network sensitivity for input differences in order to evaluate the dimensions which are most likely to generate the target misclassification with minimum perturbation.

Perturb all the input dimensions:

Perturb every input dimensions but with a small quantity in the direction of the sign of the gradient calculated using the FGSM method.

This method efficiently minimizes the Euclidean distance between the original and the corresponding adversarial samples.

Perturb a selected input dimensions:

Involves saliency maps to select only a limited number of input dimensions to perturb.

The objective of using saliency map is to assign values to the combination of input dimensions which indicates whether the combination if perturbed, will contribute to the adversarial goals.

This method effectively reduces the number of input features perturbed while crafting adversarial examples.

For choosing the input dimensions which forms the perturbations, all the dimensions are sorted in decreasing order of adversarial saliency value.

The saliency value S(x,t)[i] of a component i of a legitimate example x for a target class t is evaluated using the following equation:

$$S(x,t)[i] = \begin{cases} 0, & \text{if } \frac{\partial F_t}{\partial x_i}(x) < 0 \text{ or } \sum_{j \neq t} \frac{\partial F_j}{\partial x_i}(x) > 0 \\ \frac{\partial F_t}{\partial x_i}(x) \left| \sum_{j \neq t} \frac{\partial F_j}{\partial x_i}(x) \right|, & \text{otherwise} \end{cases}$$

where $\left[\frac{\partial F_j}{\partial x_i}\right]_{ij}$ could be easily calculated using the Jacobian Matrix J_F of the learned model

Jacobian matrix is a matrix of partial derivatives.

Jacobian is the determinant of the jacobian matrix.

The matrix will contain all partial derivatives of a vector function.

$$\mathbf{J} = \begin{bmatrix} \dfrac{\partial \mathbf{f}}{\partial x_1} & \cdots & \dfrac{\partial \mathbf{f}}{\partial x_n} \end{bmatrix} = \begin{bmatrix} \nabla^{\mathrm{T}} f_1 \\ \vdots \\ \nabla^{\mathrm{T}} f_m \end{bmatrix} = \begin{bmatrix} \dfrac{\partial f_1}{\partial x_1} & \cdots & \dfrac{\partial f_1}{\partial x_n} \\ \vdots & \ddots & \vdots \\ \dfrac{\partial f_m}{\partial x_1} & \cdots & \dfrac{\partial f_m}{\partial x_n} \end{bmatrix}$$

where $\nabla^{\mathrm{T}} f_i$ is the transpose (row vector) of the gradient of the i component.

The saliency value S(x,t)[i] of a component i of a legitimate example x for a target class t is evaluated using the following equation:

$$S(x,t)[i] = \begin{cases} 0, & \text{if } \frac{\partial F_t}{\partial x_i}(x) < 0 \text{ or } \sum_{j \neq t} \frac{\partial F_j}{\partial x_i}(x) > 0 \\ \frac{\partial F_t}{\partial x_i}(x) \left| \sum_{j \neq t} \frac{\partial F_j}{\partial x_i}(x) \right|, & \text{otherwise} \end{cases}$$

where $\left[\frac{\partial F_j}{\partial x_i}\right]_{ij}$ could be easily calculated using the Jacobian Matrix J_F of the learned model

F. Input components are added to perturbation δx in the decreasing order of the saliency value $S(x,t)[i]$ until the resulting sample $x_* = x + \delta x$ is misclassified by the model F.

Advantages and Drawbacks

The first method (FDGM) is well fitted for the fast crafting of many adversarial samples but with a relatively large perturbation and thus is potentially easier to detect.

The second method (Saliency) reduces the perturbations at the expense of a higher computing cost.

Adversarial examples in a black-box setting.

in the case of an adaptive black-box scenario, the adversary does not have access to a large dataset and thus augments a partial or randomly selected dataset by selectively querying the target model as an oracle.

-train a local substitute model which approximates the decision boundary of the target model. Once the local model is trained with high confidence, any of the white-box attack strategies can be applied.

Eg: One of the popular method of dataset augmentation- **Jacobian based Data Augmentation**.

Jacobian based Data Augmentation.

An adversary could potentially make an infinite number of queries to get the Oracle’s output O(x) for any input x.

This would provide the adversary a copy of the oracle. However, the process is not tractable considering the continuous domain of an input to be queried.

Furthermore, making a significant number of queries presents the adversarial behavior easy to detect.

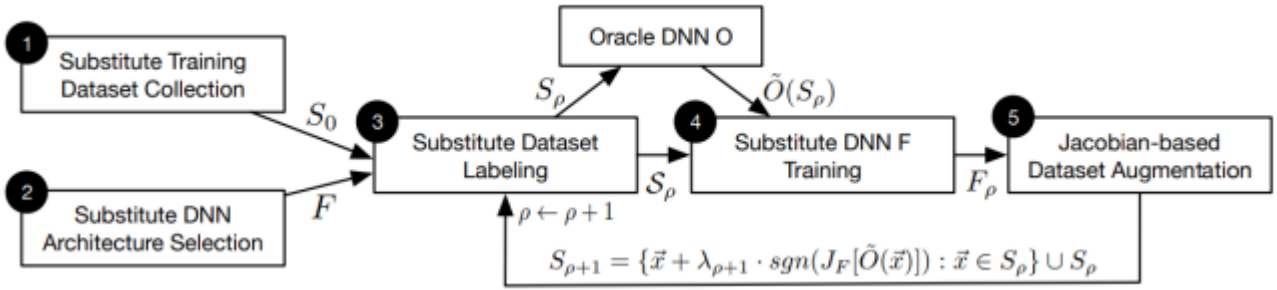
Hence, the greedy heuristic that an adversary follow is to prioritize the samples while querying the oracle for labels to get a substitute DNN F approximating the decision boundaries of the Oracle.

These directions can be identified with the **substitute DNN's Jacobian matrix JF** , which is evaluated at several input points x.

Precisely,the adversary evaluates the sign of the Jacobian matrix dimension corresponding to the label assigned to input x by the oracle, denoted by sдn (JF (x) [O(x)]).

The term λ∗sдn (JF (x) [O(x)]) is added to the original datapoint, to obtain a new synthetic training point.

The iterative data augmentation technique can be summarized using the following equation.



where S_n is the dataset at n^{th} step and S_{n+1} is the augmented dataset.

*https://dl.acm.org/doi/pdf/10.1145/3052973.3053009

Algorithm 1 - Substitute DNN Training: for oracle $\tilde{O}$, a maximum number max_ρ of substitute training epochs, a substitute architecture F, and an initial training set S_0.

Input: $\tilde{O}$, max_ρ, S_0, λ

Define architecture F
for $\rho \in 0 \,..\, max_\rho - 1$ **do**
    // Label the substitute training set
    $D \leftarrow \left\{ (\vec{x}, \tilde{O}(\vec{x})) : \vec{x} \in S_\rho \right\}$
    // Train F on D to evaluate parameters θ_F
    $\theta_F \leftarrow \text{train}(F, D)$
    // Perform Jacobian-based dataset augmentation
    $S_{\rho+1} \leftarrow \{\vec{x} + \lambda \cdot \text{sgn}(J_F[\tilde{O}(\vec{x})]) : \vec{x} \in S_\rho\} \cup S_\rho$
end for
return θ_F

*https://dl.acm.org/doi/pdf/10.1145/3052973.3053009

Here, the limitation is placed on the substitute:

it must model a differentiable function—to allow for synthetic data to be generated with its Jacobian matrix.

Gradient Hiding

A natural defense against gradient-based attacks [e.g FGSM too]:

- could consist in hiding information about the model's gradient from the adversary if the model is non-differentiable
- (e.g, a Decision Tree, a Nearest Neighbor Classifier, or a Random Forest),
- gradient-based attacks are rendered ineffective in such cases.

[However,this defense are easily fooled by learning a surrogate Black-Box model having gradient and crafting examples using it.]

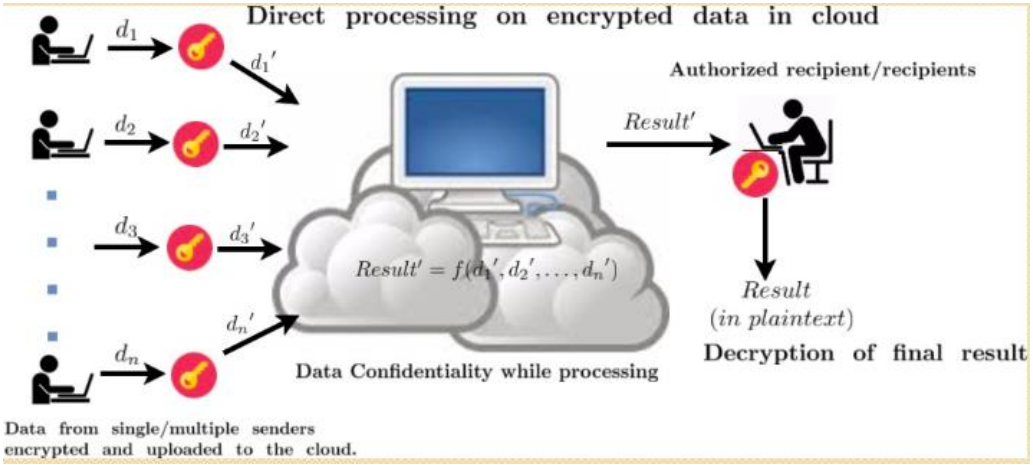
Privacy-Preserving Machine Learning

According to member inference attack mentioned], malicious users in the training might use the plaintext gradient to train a shadow model to compromise the data security of other users.

Thus, homomorphic encryption against this attack, allows one to perform calculations on encrypted gradients without decrypting it.

The privacy-preserving issue is one of the most practical problems for machine learning recently.

Fully homomorphic encryption (FHE) is one appropriate tool for privacy-preserving machine learning (PPML) to ensure strong security in the cryptographic sense.

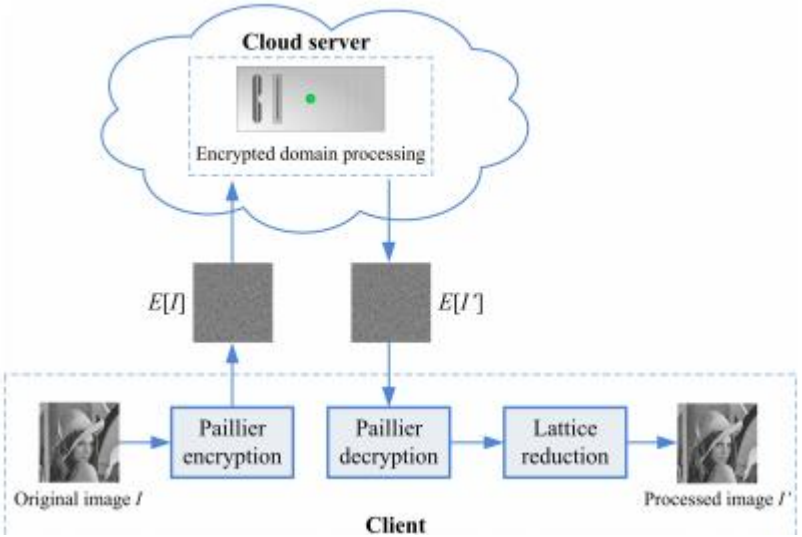


What is homomorohism?

- A group is an algebraic object consisting of a set together with a single binary operation, satisfying certain axioms.
- If G and H are groups, a homomorphism from G to H is a function $f : G \to H$ such that $f(g_1 * g_2) = f(g_1) * f(g_2)$ for any elements $g_1,\ g_2 \mathrm{in} G$, where $*$ denotes the operation in G and $*$ denotes the operation in H.

Homomorphism and Encryption

- Example in **Unpadded RSA (1977)** :
  If the RSA public key is modulus m and exponent e, then the encryption of a message x is given by $\mathcal{E}(x) = x^e \bmod m$. The homomorphic property is then $\mathcal{E}(x_1) \cdot \mathcal{E}(x_2) = x_1^e x_2^e \bmod m = (x_1 x_2)^e \bmod m = \mathcal{E}(x_1 \cdot x_2)$.
- **Hence, concept of homomorphism in encryption is not new**.



Paillier Encryption

Key Generation: KeyGen(p, q)

Input: $p, q \in \mathbb{P}$

Compute $n = pq$

Choose $g \in \mathbb{Z}^*_{n^2}$ such that

$$\gcd(L(g^\lambda \bmod n^2), n) = 1 \text{ with } L(u) = \frac{u-1}{n}$$

Output: (pk, sk)

public key: $pk = (n, g)$

secret key: $sk = (p, q)$

Encryption: Enc(m, pk)

Input: $m \in \mathbb{Z}_n$

Choose $r \in \mathbb{Z}^*_n$

Compute $c = g^m \cdot r^n \bmod n^2$

Output: $c \in \mathbb{Z}_{n^2}$

Decryption: Dec(c, sk)

Input: $c \in \mathbb{Z}_{n^2}$

Compute

$$m = \frac{L(c^\lambda \bmod n^2)}{L(g^\lambda \bmod n^2)} \bmod n$$

- **Homomorphic addition of plaintexts**

  The product of two ciphertexts will decrypt to the sum of their corresponding plaintexts,

  $$D(E(m_1, r_1) \cdot E(m_2, r_2) \bmod n^2) = m_1 + m_2 \bmod n.$$

  The product of a ciphertext with a plaintext raising g will decrypt to the sum of the corresponding plaintexts,

  $$D(E(m_1, r_1) \cdot g^{m_2} \bmod n^2) = m_1 + m_2 \bmod n.$$

Application in Federated Learning*

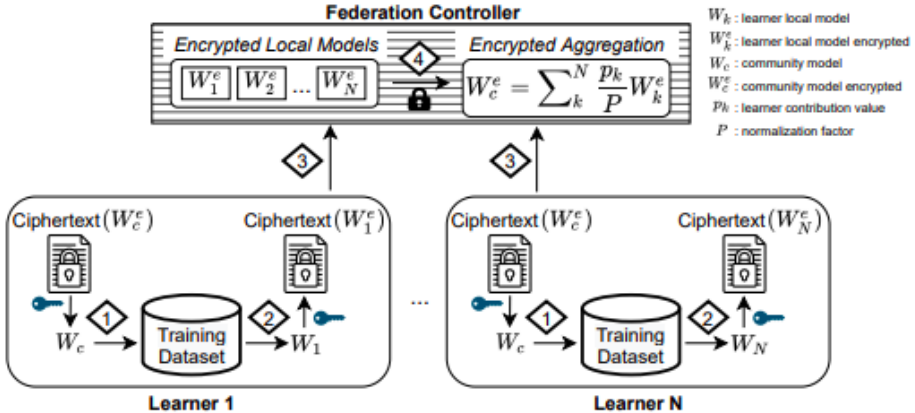


Figure Federated System Architecture with Encryption

***Secure Neuroimaging Analysis using Federated Learning with Homomorphic Encryption, SIPAIM 2021**

Code References

https://github.com/data61/python-paillier/blob/master/examples/alternative_base.py

https://github.com/data61/python-paillier/blob/master/examples/federated_learning_with_encryption.py

Resnet 20: How to compute in ciphertext domain?