ENCRYPTED MACHINE LEARNING

CRYPTOGRAPHY AND NETWORK SECURITY

ANUDIT NAGAR
E18CSE024
ABHINAV ROBINSON
E18CSE006
ASHOK KUMAR JAKKULA
E18CSE029

BENNETT UNIVERSITY

Introduction

While building Machine Learning applications, we often use data from multiple sources to train the machine learning model. In the health-care and finance domains this kind of data and the model can be extremely critical and sensitive. The model parameters are what companies work towards strengthening and optimizing and therefore are an important business asset whereas the data can contain private and personal information like medical records, financial history etc about clients or patients.

To solve this problem one approach is to encrypt the gradients of the machine learning and the data that the machine learning model uses to train on. This ensures personal information like data samples are not revealed for example, medical facilities would not be able to access patient records unless explicitly allowed to do so. There are a lot of encryption algorithms that exist, allowing for computation on encrypted data. The most common and widely adopted are,

1. Secure Multi-Party Computation (SMPC),
2. Homomorphic Encryption (FHE/SHE)
3. Functional Encryption (FE).

In our project we utilize a combination of Secure Multi-Party Computation and Fully Homomorphc encryption. We use open source libraries and crypto protocols like PyTorch, Syft, PyCrypto, SecureNN, SPDZ. These help us building an encrypted pipeline where we train and predict on encrypted data reliably and accurately.

Theoretical Details

Additively Homomorphic Encryption Algorithm

For any randomly chosen vectors ~r and ~g in a local dataset, using the public key B_{pk} and the message ~m ∈ {0, 1} n is encrypted by:

$$\vec{c} = E(\vec{m}) = \vec{m} + \vec{r} \cdot B_I + \vec{g} \cdot B_J^{pk},$$

Here, B_I is the basis of an ideal lattice L and ~m + ~r · B_I is the noise generator..

Additively Homomorphic Decryption Algorithm

Using a secret key B^J_{sk} , the cipher is decrypted as follows:

$$\vec{m} = \vec{c} - B_J^{sk} \cdot \lfloor (B_J^{sk})^{-1} \cdot \vec{c} \rceil \mod B_I,$$

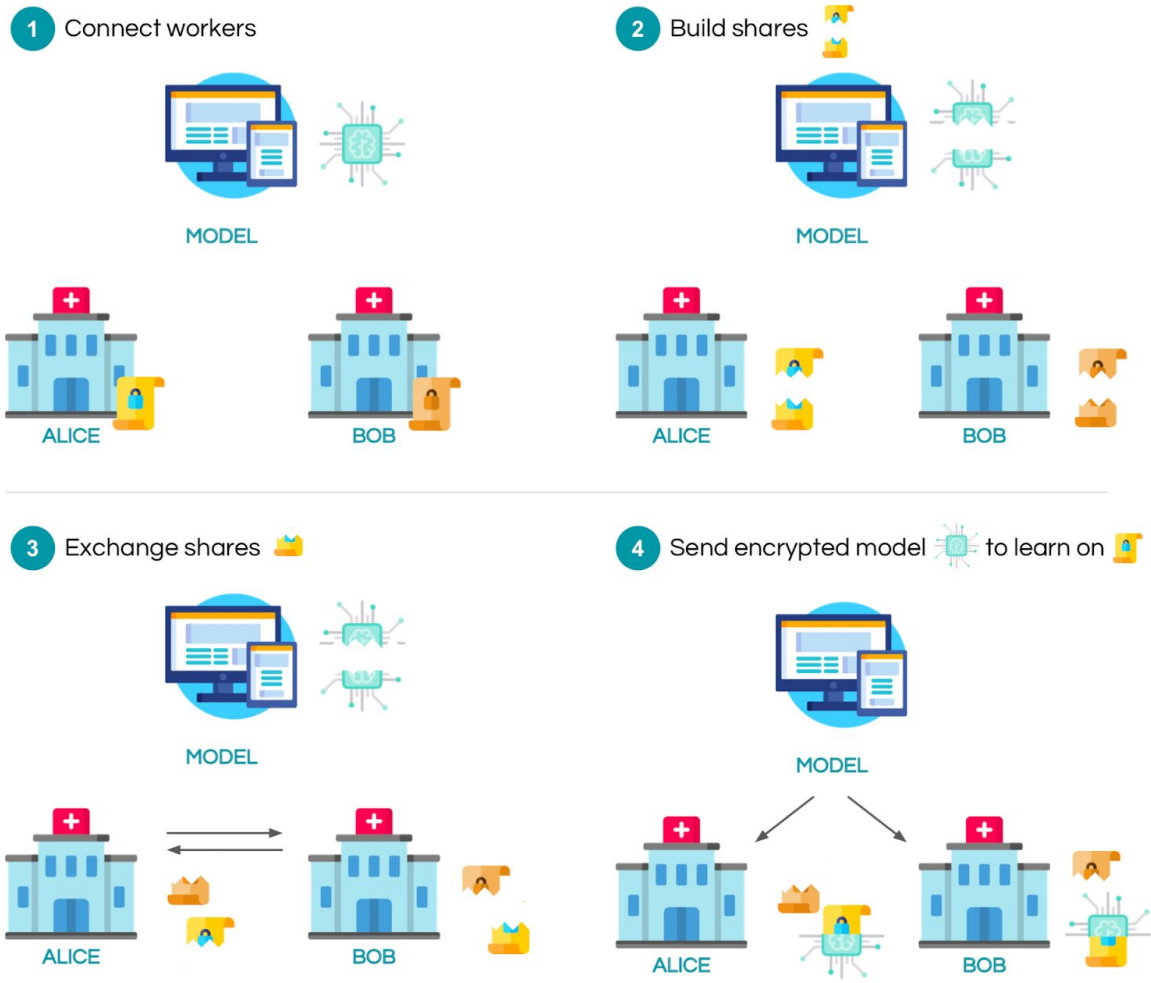
Here B·x is a nearest integer function for the coefficients of the vectorized training data.

Available Implementations

Name	Scheme	Lang	Documentation	Libraries
HElib [Halevi and Shoup 2013b]	BGV [Brakerski et al. 2011]	C++	Yes [Halevi and Shoup 2013a]	NTL, GMP
libScarab [Perl et al. 2011a]	SV [Smart and Vercauteren 2010]	C	Yes [Perl et al. 2011b]	GMP, FLINT, MPFR, MPIR
FHEW [Ducas and Micciancio 2014]	DM14 [Ducas and Micciancio 2015]	C++	Yes [Ducas and Micciancio 2015]	FFTW
TFHE [Chillotti et al. 2017]	CGGI16 [Chillotti et al. 2016]	C++	Yes [Chillotti et al. 2016]	FFTW
SEAL [Laine et al. 2017]	FV12 [Fan and Vercauteren 2012b]	C++	Yes [Chen et al. 2017]	No external dependency

Project Architecture

In this Section we elaborate on our Project Architecture and the encrypted machine learning setup.



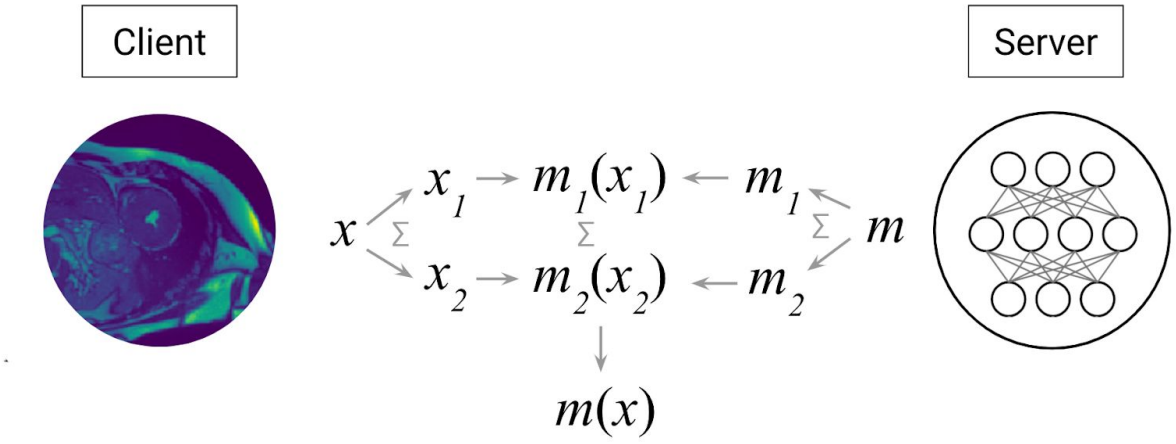
The algorithm is as follows,

- A server wants to train on local data stored on *n devices.*
- The server starts by sending the encrypted model to the n devices.
- The data on each device is encrypted using the homomorphic encryption algorithm described in the above section

- Now each device can start training on the data without sending the actual data to the servers. This brings massive security to the entire system.
- After the training is complete, all the trained model updates are sent back to the server where they are aggregated.

We ensure that the computations run are secure in an honest-but-curious adversarial model.

In the aggregation phase, encrypted gradients received from the devices are averaged by the following method.



This ensures that corrupted updates from an agent cannot bring down the accuracy of the entire network.

Project Implementation

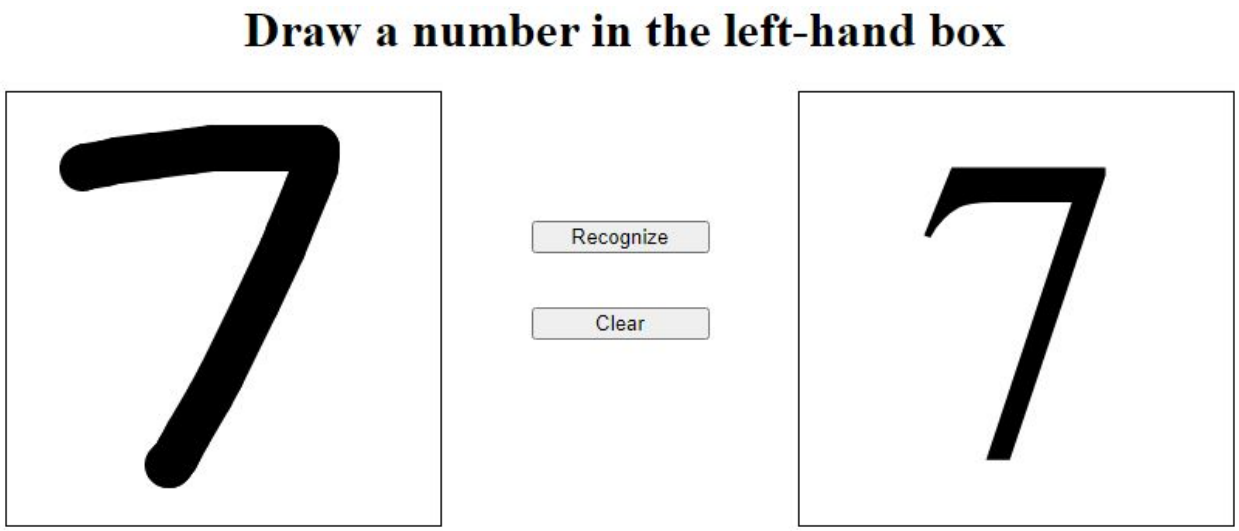
Our Neural network architecture is as follows,

```python
class Net(nn.Module):
    def __init__(self):
        super(Net, self).__init__()
        self.conv1 = nn.Conv2d(1, 32, 3, 1)
        self.conv2 = nn.Conv2d(32, 64, 3, 1)
        self.dropout1 = nn.Dropout(0.25)
        self.dropout2 = nn.Dropout(0.5)
        self.fc1 = nn.Linear(9216, 128)
        self.fc2 = nn.Linear(128, 10)

    def forward(self, x):
        x = self.conv1(x)
        x = F.relu(x)
        x = self.conv2(x)
        x = F.relu(x)
        x = F.max_pool2d(x, 2)
        x = self.dropout1(x)
        x = torch.flatten(x, 1)
        x = self.fc1(x)
        x = F.relu(x)
        x = self.dropout2(x)
        x = self.fc2(x)
        output = F.log_softmax(x, dim=1)
        return output
```

With this setup we were able to achieve an accuracy of **98.3%** with 12 Epochs and 10 Devices training on encrypted data.

Project Demo



Project Demo :
https://encrypt-pred.herokuapp.com/

Project Codebase:
https://github.com/REGATTE/Crypto_Sem5_Project

Conclusion

We create an encrypted machine learning architecture that can be adapted for any task and performs reliably and accurately on encrypted data improving security of the traditional machine learning paradigm.